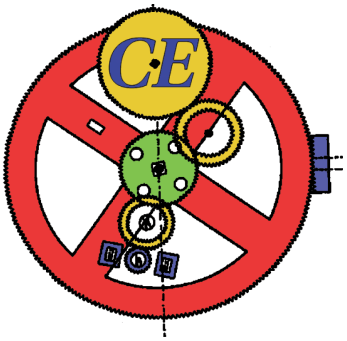


Computer Engineering
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ce.et.tudelft.nl/

2019

MSc THESIS

Monocular Visual Inertial Odometry for Underwater Vehicle Navigation, Optimized on Embedded System

Georgios Giannakaras

Q&CE-CE-MS-2019-19

Abstract

Underwater robots are widely used in various operations below the surface of the sea. In many of these operations the precise localization and positioning is quite important, especially in cases that the robot must navigate between man-made installations. Although underwater vehicles can carry a number of different sensors, many of them can not be exploited for efficient navigation, due to costs, complexity, weight and inaccuracy. Most of robots feature at least one camera, a sensor that can be cheap, small, with low power requirements and that offers vast amount of information. The advances in computer vision and embedded systems have set cameras as an attractive mean for navigation. This combination allows computer vision as a viable alternative to provide data in real-time. This data can be used locally as a positioning source for the own robot control system, or to inform the navigation coordinates to an observer or another system. Visual Odometry (VO) is the process of estimating the position and orientation of a robot based completely on data acquired by cameras. The camera-only system can be enhanced with the fusion of an Inertial Measurement Unit (IMU) to estimate motion even more accurately, resulting in Visual Inertial Odometry (VIO).

This thesis researches various approaches for underwater navigation with edge computing and focuses on systems that utilize a camera and an IMU. A VIO system architecture is proposed, comprised by a machine vision camera, an IMU, a micro-controller and the embedded system Jetson Xavier from NVIDIA. Some of the most recent VO/VIO algorithms are evaluated on a number of underwater datasets and the impact of the IMU on their performance is assessed. Design issues are discussed and challenges related to the camera - IMU fusion are analyzed. The Visual-Inertial ORB-SLAM (ORB-SLAM + IMU) algorithm is deployed on the proposed VIO system and is optimized on the embedded GPU. The whole framework is evaluated on an artificial underwater structured environment which was created in an indoor tank. While the outcome of the algorithm on motion estimation is examined, the computational performance of the embedded system is profiled for various power modes as well. Results show that the proposed VIO system is able to estimate the underwater robot's traversed trajectory in the tank with adequate accuracy and that can execute the Visual-Inertial ORB-SLAM in real-time with sufficient speed.

TU Delft
Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

Monocular Visual Inertial Odometry for Underwater Vehicle Navigation, Optimized on Embedded System

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

EMBEDDED SYSTEMS

by

Georgios Giannakaras
born in Volos, Greece

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Monocular Visual Inertial Odometry for Underwater Vehicle Navigation, Optimized on Embedded System

by Georgios Giannakaras

Abstract

Underwater robots are widely used in various operations below the surface of the sea. In many of these operations the precise localization and positioning is quite important, especially in cases that the robot must navigate between man-made installations. Although underwater vehicles can carry a number of different sensors, many of them can not be exploited for efficient navigation, due to costs, complexity, weight and inaccuracy. Most of robots feature at least one camera, a sensor that can be cheap, small, with low power requirements and that offers vast amount of information. The advances in computer vision and embedded systems have set cameras as an attractive mean for navigation. This combination allows computer vision as a viable alternative to provide data in real-time. This data can be used locally as a positioning source for the own robot control system, or to inform the navigation coordinates to an observer or another system. VO is the process of estimating the position and orientation of a robot based completely on data acquired by cameras. The camera-only system can be enhanced with the fusion of an IMU to estimate motion even more accurately, resulting in VIO.

This thesis researches various approaches for underwater navigation with edge computing and focuses on systems that utilize a camera and an IMU. A VIO system architecture is proposed, comprised by a machine vision camera, an IMU, a micro-controller and the embedded system Jetson Xavier from NVIDIA. Some of the most recent VO/VIO algorithms are evaluated on a number of underwater datasets and the impact of the IMU on their performance is assessed. Design issues are discussed and challenges related to the camera - IMU fusion are analyzed. The Visual-Inertial ORB-SLAM (ORB-SLAM + IMU) algorithm is deployed on the proposed VIO system and is optimized on the embedded GPU. The whole framework is evaluated on an artificial underwater structured environment which was created in an indoor tank. While the outcome of the algorithm on motion estimation is examined, the computational performance of the embedded system is profiled for various power modes as well. Results show that the proposed VIO system is able to estimate the underwater robot's traversed trajectory in the tank with adequate accuracy and that can execute the Visual-Inertial ORB-SLAM in real-time with sufficient speed.

Laboratory : Computer Engineering
Codenumber : Q&CE-CE-MS-2019-19

Committee Members :

Advisor: Dr.ir. J.S.S.M. Wong, CE, TU Delft

Advisor: Dr.ir. A.J. van Genderen, CE, TU Delft

Chairperson:	Dr.ir. J.S.S.M. Wong, CE, TU Delft
Member:	Dr.ir. C.J.M Verhoeven, Microelectronics, TU Delft
Member:	Dr. Robert Andersson, Supervisor, Fugro
Member:	Dr. Diego Carvalho, Supervisor, Fugro

Dedicated to my family: my parents who always support me, my sister who is always by my side, and my beloved grandmother.

Contents

List of Figures

List of Tables

List of Acronyms

AHRS	Attitude and Heading Reference System
A-KAZE	Accelerated-KAZE
ANN	Approximate Nearest Neighbours
APE	Absolute Position Error
API	Application Programming Interface
AUV	Autonomous Underwater Vehicle
BA	Bundle Adjustment
BoW	Bag of Words
BRIEF	Binary Robust Independent Elementary Features
BRISK	Binary Robust Invariant Scalable Keypoints
CPU	Central Processing Unit
DoG	Difference of Gaussian
DPPTAM	Dense Piecewise Planar Tracking and Mapping
CenSurE	Center Surround Extremas
DA	Data Association
DoF	Degrees of Freedom
DSO	Direct Sparse Odometry
DTAM	Dense Tracking And Mapping
DVL	Doppler Velocity Log
EKF	Extended Kalman Filter
FAST	Features from Accelerated Segment Test
FOG	Fiber Optical Gyroscopes
FOV	Field of View
FPS	Frames Per Second
FREAK	Fast Retina Keypoint
GBA	Global Bundle Adjustment

GPIO	General-Purpose Input/Output
GPS	Global Positioning System
GPU	Graphics Processing Unit
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
iSAM	Incremental Smoothing And Mapping
KLT	Kanade-Lucas-Tomasi
LBA	Local Bundle Adjustment
LBL	Long Baseline
LoG	Laplacian of Gaussian
LSD-SLAM	Large-Scale Direct Monocular SLAM
NASA	National Aeronautics and Space Administration
NCC	Normalized Cross Correlation
OF	Optical Flow
ORB	Oriented FAST and Rotated BRIEF
PTAM	Parallel Tracking and Matching
RANSAC	Random Sample Consensus
REMODE	REgularized MOnocular Depth Estimation
RoI	Regions of Interest
ROS	Robot Operating System
ROV	Remotely Operated underwater Vehicle
RPE	Relative Position Error
SAD	Sum of Absolute Differences
SBL	Short Baseline
SIFT	Scale Invariant Feature Transform
SLAM	Simultaneous localization and mapping
SOM	Self-Organizing Maps
SSD	Sum of Square Differences

SURF	Speeded-Up Robust Features
SVD	Singular Value Decomposition
SVO	Semi-Direct Visual Odometry
TMS	Tether Management System
ToF	Time of Flight
UGV	Unmanned Ground Vehicle
USBL	Ultra Short Baseline
U-SURF	Upright-SURF
UWSim	Underwater Simulator
VIO	Visual Inertial Odometry
VI-SLAM	Visual Inertial SLAM
VO	Visual Odometry
V-SLAM	Visual Simultaneous Localization And Mapping
ZNCC	Zero-mean Normalized Cross-Correlation

Acknowledgements

The completion of this MSc thesis marks the end of two intense years in TU Delft and the achievement of a significant milestone. Studying for the MSc in Embedded Systems was a great experience and there could not be another better way than this thesis to finish it. The subject of this thesis is multidisciplinary and the learning curve was steep throughout the whole process. To accomplish successfully this MSc degree a lot of diverse knowledge had to be acquired and a huge amount of effort was required.

I would like to thank my parents that always support me to fulfill my dreams and accomplish my goals, my sister that is always by my side and my beloved grandmother. I am glad that they encouraged me to pursue this degree at TU Delft. Moreover, I would like to thank my dear friend Alex with whom we had a great time and a remarkable journey during this MSc with endless hours of studying and laughing. Special thanks to my girlfriend for her support and patience throughout the whole MSc. In addition, I would like to express my gratitude to my supervisors from Fugro, to Diego Carvalho for his endless support, patience and his great insight, and to Robert Andersson who believed in me, significantly helped me to form the topic of this thesis, and for his guidance. Last but not least, I would like to thank my supervisors from TU Delft, Dr. Arjan van Genderen and Dr. Stephan Wong for their support, guidance, advice and for enabling me to do this thesis topic in the industry.

Georgios Giannakaras
Delft, The Netherlands
September 20, 2019

1 Introduction

In the last decades mobile systems and robotics have been an integral part of the industry and they are involved in various domains. Especially, with the recent blooming of powerful mobile computing devices and embedded systems, robotics have become an even more hot topic, attracting the attention of many research groups and companies. While the market share of robotics has been rapidly growing for commercial use, it is also aiming for the automation of various challenging tasks that the people in industry have to accomplish. A lot of these tasks regard the underwater domain, especially if we consider that the water covers approximately the 71% of the earth's surface. The need to operate underwater comes from many applications, such as oil & gas extraction, underwater structures/pipeline inspection, archaeology, marine science, ship maintenance and infrastructures deployment for renewable energy. All the aforementioned applications are very important, however some of them are critical, involving high risks. The oil & gas domain belongs to the most crucial ones, taking into account that there is a trend to expand its operations in deeper regions of the sea, to exploit the vast mount of the unexploited resources.

Some common actions that take place in underwater resources extraction is the survey of the seabed and the study of its morphology, deployment of huge man-made structures, and drilling. It is well known that the tasks that have to be accomplished underwater are way more challenging compared to the onshore, due to the harsh environment of the sea, with high pressure, lack of light, strong currents and the highly dynamic conditions. On top of these factors, there is the fact that the regions around oil & gas extraction might get in risk, because even the smallest mistake can lead to severe disasters, while various kinds of operations have to be accomplished. Such kind of operations require situation awareness, perception, detailed organization and high dexterity. Even though specialized divers can be utilized for the underwater tasks, they are limited to work up to a maximum of approximately 500 meters. For this reason Remotely Operated Vehicles (ROV) are employed, in order to conduct tasks in bigger depths, while keeping the workers safe and away of the underwater risks. An ROV typically is an underwater remotely operated robot, featuring thrusters for motion, buoyancy parts, various sensors like cameras, Inertial Measurement Unit (IMU), sonar, and it communicates with a vessel through an umbilical cable.

Utilization of ROVs is convenient, but is not always a simple procedure. In most cases a vessel is employed along with a specialized and large crew. Furthermore, in contrast to an Autonomous Underwater Vehicle (AUV), an ROV requires a trained pilot to manipulate it and a whole infrastructure to be deployed (Figure 1.1). It is already evident that to conduct such a mission is very costly and it demands a significant number of people to be involved. Not only that, but due to the great risk of a possible accident near an oil extraction area, a lot of money is spent in insurances, while the

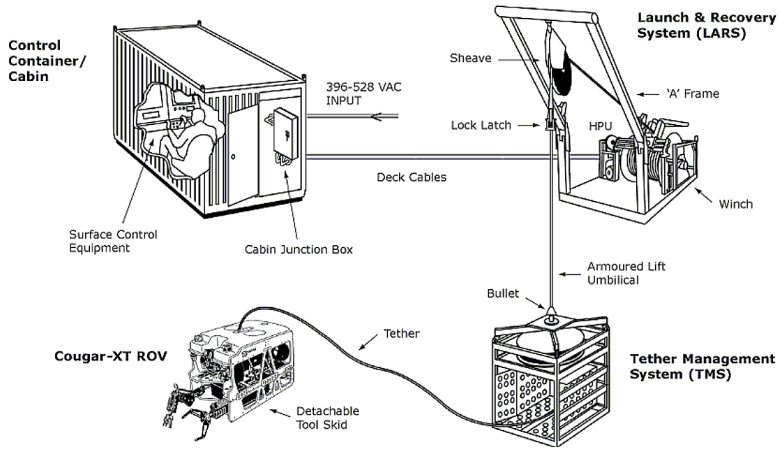


Figure 1.1: ROV mission infrastructure. (Courtesy of SAAB)

level of the coordination required is quite significant. One way to simplify the operations and to reduce the costs is to transfer a big part of the mission onshore. Figure 1.2 illustrates such a case. Ideally, most of the crew would be onshore along with the ROV pilot and offshore a smaller vessel with only the essential amount of personnel would be employed. To achieve this concept a communication link via a satellite should be involved, but most importantly the ROV should feature autonomy capabilities. Remote ROV manipulation requires the commands to be sent over the satellite to the vessel, and from the vessel through a tether to the underwater vehicle. It is evident that this process suffers from transmission overhead and notable delay is introduced. For this reason the pilot should only control the ROV for essential motions, while the robot should be able to autonomously perform actions, such as station keeping against currents and moving efficiently around man-made structures. Moreover, the robot should be able to navigate itself away from dangerous regions and to avoid obstacles in case the communication with the pilot is totally lost.

Edge computing and embedded systems play a significant role for the accomplishment of the aforementioned goals. In a lot of ROV missions the robot collects data underwater and transmits them via a tether on the vessel, where all the processing is done on servers. This involves the overhead of saving and transmitting the data, while in case that the connection via the tether is lost the vehicle is out of control and unable to operate. A promising solution is to relocate all the processing on the edge, by utilizing powerful embedded systems and by providing further intelligence to the robot.

With edge computing integrated into the ROV, the robot is able to carry out some vital tasks, such as navigation, positioning and localization. Although accurate localization is critical for most of the subsea operations, it is very difficult to do it precisely, in contrast to the onshore applications. Many of the sensors and techniques that are widely used on aerial and ground robots are not accurate or do not work underwater. For example it is impossible to use Global Positioning System (GPS) underwater, as the sea water absorbs the radio signals. One of the most common methods for underwater navigation is the use of acoustic positioning systems, however they are expensive systems,

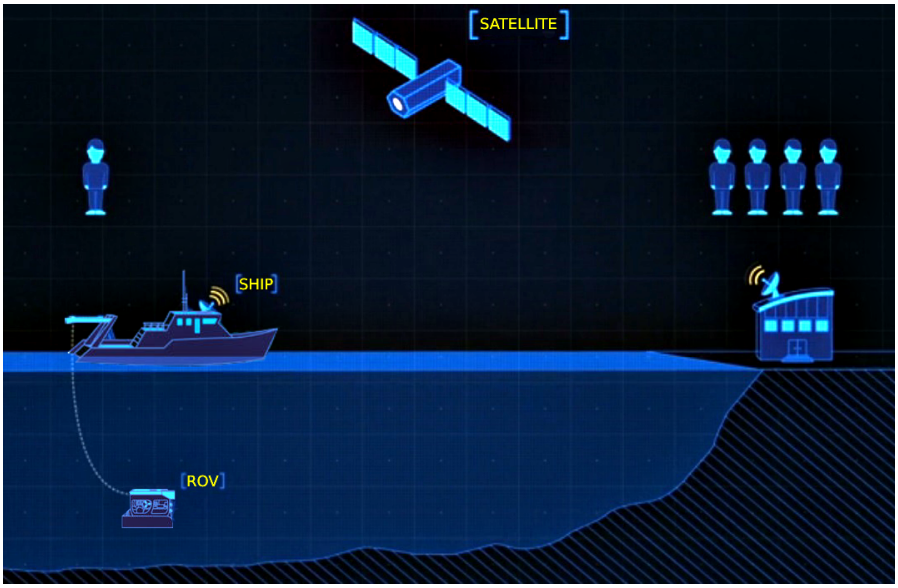
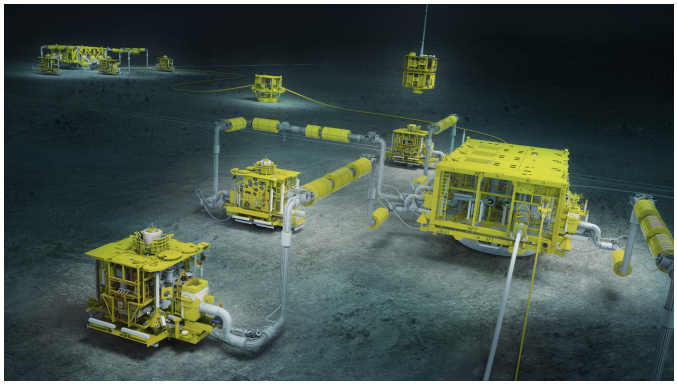


Figure 1.2: Illustration of remote ROV operation. (Courtesy of DexROV)

which require precise calibration, time consuming deployment and they provide noisy measurements, deviating from cm to many meters from the real position. Other means of navigation are the Doppler Velocity Log (DVL), the Fiber Optical Gyroscopes (FOG) and sonars, which are also expensive (more than 100.000 €), heavy (many kg) and might be unsuitable for some kinds of operations.

This thesis examines the case of underwater environments surrounding oil & gas infrastructures, which are occupied by man-made installations. In addition, we consider the phase in which the ROV has reached the seabed and it is about to operate around a structured area, such as the depicted ones in Figure 1.3. Taking into account that most of the ROVs feature at least one camera and vast information is provided by images, a promising option would be to utilize computer vision algorithms along with the camera sensor for navigation. Computer vision has been around many decades and is involved in various applications. In the recent years significant advances have taken place, including autonomous navigation and perception for robotics. Especially, computer vision is known for being used in autonomous driving for cars and in applications with drones. One of the most popular missions that it is known for, is the NASA mission in Mars, featuring rovers that navigate and localize by utilizing computer vision algorithms. Despite the advances, visual motion estimation has not been widely explored for the underwater cases, and it is far less researched, in contrast to land and air applications. This is due to the amount of challenges that are met underwater, the demand for expertise and the high costs for conducting experiments.

The most researched topics in visual motion estimation are the Visual Odometry (VO) and Visual Simultaneous Localization And Mapping (V-SLAM). These two methods utilize images captured from cameras to gradually estimate motion and eventually to compute a traversed trajectory. The motion estimation is done by initially extracting salient features or pixel intensities from a frame, and by finding afterwards



(a)

(b)

Figure 1.3: (a) Underwater structured area (Courtesy of AkerSolutions). (b) Man-made structure deployed underwater. [1]

their new position in the next frames. VO and V-SLAM are strongly connected, and specifically the first one is part of the latter. V-SLAM introduces, on top of VO, the construction and maintenance of a global map. It also utilizes this map to detect loop closures, i.e. to recognize places that have been visited before. A common combination of sensors, also known as sensor fusion, is a camera with an IMU. The IMU is another typical sensor for the underwater robots and it provides inertial navigation measurements, such as linear accelerations and angular velocities. If these two sensors are used individually they introduce drift during motion estimation. However, if they are fused, one can compensate for the drift of the other, increasing the level of accuracy for motion estimation. When an IMU is utilized along with a camera, the aforementioned terms are usually referred as Visual Inertial Odometry (VIO) and Visual Inertial SLAM (VI-SLAM). Despite the advantages that the IMU brings, it comes also with some difficulties, such as the synchronization that is required to operate along with a camera.

1.1 Scope & Contributions

This MSc thesis has as an objective the motion estimation for a ROV with state-of-the-art hardware and software, aiming to enhance its awareness and navigation accuracy in subsea structured areas. It is important for a ROV to navigate around man-made installations with an accuracy of centimetre, while there are a lot of limitations in the current means of motion estimation, due to the challenges that an underwater environment introduces. This thesis examines the fields of navigation, positioning and localization by investigating various approaches to implement and evaluate a system for efficient motion estimation in underwater environments, that feature man-made installations. During the study, potential techniques are researched, regarding the hardware and the software parts, including a promising sensor fusion approach that contains a machine vision camera and an IMU. The goals are to examine and utilize an appropriate VIO architecture with recently published hardware, to research and evaluate the latest VIO and VI-SLAM

algorithms, and to optimize them on the implemented VIO system, aiming the underwater environment. Additional to the efficiency, the other criteria that are considered for choosing the hardware are the modules to be recently released in the market, to examine their performance and suitability for VIO and introduce the results to the research community.

The range of study is quite broad, as it covers a complete system implementation, from hardware to software, including research and prototyping for both. Moreover, the study is multidisciplinary, featuring various research fields, such as computer vision, robotics, control, sensors, embedded systems and algorithms optimization.
To achieve these goals:

- This thesis initially evaluates a wide number of popular feature detectors and descriptors by enhancing a VO algorithm, to support them and utilize them for motion estimation, in C++ with OpenCV. Additionally, a real-time visualizer was implemented in Python to illustrate the algorithm's estimated trajectory against the groundtruth.

- Some of the most recent VI-SLAM algorithms, featuring the evaluated feature detectors/descriptors, are studied, configured, and assessed for underwater cases. For the evaluation of the algorithms for an underwater environment, most of the online available underwater datasets are utilized. In contrast to available land and air datasets, the underwater datasets along with their quality and documentation are quite limited. For this reason these datasets are gathered, analyzed, and evaluated, while the VI-SLAM algorithms are properly configured for them.

- The impact of image processing filters on underwater video datasets is examined. To evaluate if the performance of the VI-SLAM algorithms is improved with an enhanced underwater sequence, image processing filters are investigated and applied by utilizing C++ and OpenCV.

- The studied algorithms for motion estimation offer a wide range of parameters to configure. In this thesis the affect of each of them on the accuracy and performance is evaluated and interpreted, for underwater cases.

- Visual-motion-estimation system setups are investigated and a VIO system architecture is evaluated and improved, including a machine vision camera from *FLIR*, an IMU from *SBG* and an embedded system from *NVIDIA*. To utilize the camera in the system, a low level interface between the machine vision camera and the embedded system is implemented in C++. Combining the camera and the IMU results into sensor fusion, which is deployed by timestamping and synchronizing the frames and the messages produced from the two sensors. Sensor fusion is quite challenging, hence some limitations and difficulties that were faced during its deployment are analyzed. Also the impact of the camera and IMU fusion quality on the VI-SLAM algorithms is reviewed.

- The most suitable of the examined algorithms is implemented on a VIO system along with Robot Operating System (ROS). The algorithm was optimized by utilizing CUDA accelerated parts of a similar algorithm that uses only a camera for motion

estimation. The accelerated parts are mostly based on optimized methods provided by OpenCV. The accelerated VIO algorithm utilizes the embedded system Graphics Processing Unit (GPU) and its enhanced performance is analyzed. In addition, the accelerated algorithm is evaluated on all the available NVIDIA power profiles/modes for the embedded system and their performance is compared.

- For the system to operate, the camera and the IMU have to be calibrated properly. Algorithms and methods for calibration are investigated, and their performance and suitability for the underwater cases is assessed. The complete VIO system, including the hardware and software part, is tested on air and in water.
- Up to the author's knowledge, all the online available underwater datasets feature a downward looking camera. This thesis proposes a new camera orientation, pointing forward with a slight inclination downwards. The rationale behind it is that since the examined cases feature man-made installations on the seabed, features can be extracted from the sand and from the structures at the same time. For the reason that there are no underwater datasets with such a camera orientation, an artificial underwater environment, featuring man-made installations, is set-up in an indoor tank and experiments are conducted. Underwater datasets are produced in this artificial environment, featuring also the proposed camera orientation, and are formatted into a ROS bag, to be used conveniently along with VI-SLAM algorithms and ROS.
- Design issues regarding the VIO hardware for the underwater domain are discussed.
- Finally, the performance of the complete VIO system is evaluated in the field for underwater station keeping and trajectory estimation.

1.2 Organization

This MSc thesis is organized in six chapters. In Chapter 2 the principles of VO and V-SLAM are presented, including the camera-setup options to deploy such a system, and the analysis of computer vision fundamentals for visual motion estimation. Furthermore, the related works in the domain of visual motion estimation are presented. Chapter 3 provides information and visualization of the utilized datasets, along with details for the parameters that are used for the algorithms and the hardware configuration. In Chapter 4 the employed experimental setup is presented, while the reason that specific components were selected is thoroughly justified. Chapter 5 contains the architecture and details of the implemented VIO system, a brief description of the examined VI-SLAM algorithms and the results of the conducted experiments, regarding the online and the proprietary underwater datasets. Last but not least, in Chapter 6 the formed conclusions are mentioned, followed by a discussion for possible future work.

In Appendix A the VO algorithm, that was initially examined, and the implemented improvements on it are described, while the results of its execution with all the examined feature detectors/descriptors are presented. While this chapter contains details about the improvement of the VO algorithm and some conclusions about the performance of the feature detectors/descriptors, it was placed in the appendices, because it did not produce usable results for the underwater case.

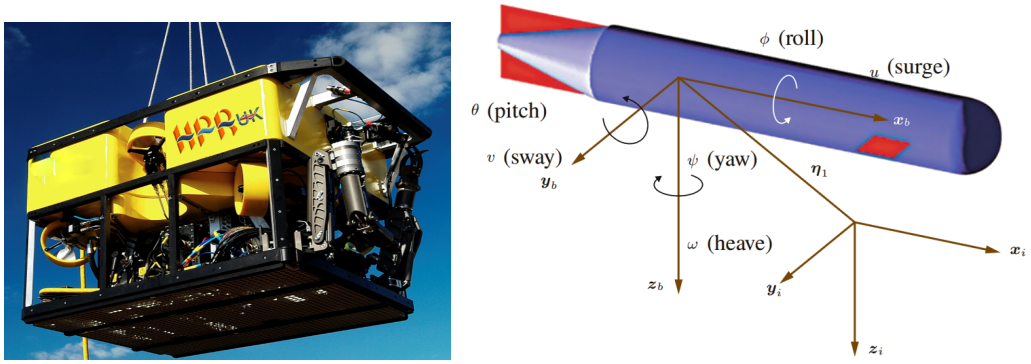
Background and Literature Review

2

In this chapter a literature review is conducted, while the research of this thesis is compared to other related works. Since the examined domain is multidisciplinary with a vast amount of information available, the most fundamental parts and terms, that also contributed to the author's knowledge to conduct this research, are discussed. Initially, some basic information about the Remotely Operated underwater Vehicle (ROV)s and their sensors is presented. Subsequently, the underwater optical imaging challenges and the difficulties that are met in subsea are discussed. A description of the utilized camera model follows along with information for camera calibration. The next part is an extensive discussion on Visual Odometry (VO). At the start, the types of camera systems and their characteristics are analyzed. Subsequently the motion estimation fundamentals and the scale uncertainty issue are presented. Afterwards, the relation of VO, Visual Inertial Odometry (VIO) and Visual Simultaneous Localization And Mapping (V-SLAM), Visual Inertial SLAM (VI-SLAM) is described, while feature-based and outliers rejection techniques are presented. A discussion is following on direct and indirect methods, on the appearance-based techniques and on the bundle adjustment. Finally, the researched related works are mentioned.

2.1 Remotely Operated underwater Vehicle (ROV)

A ROV is an underwater vehicle physically linked to a host ship via a tether, and it is controlled by an operator located on a ship/floating platform or on a proximate land. The tether is utilized for providing power and communication to the underwater vehicle and to close the manned control loop. The ROV is accompanied by the Tether Management System (TMS), which is a garage-like platform that includes the vehicle and is used for lowering the ROV into the sea. Additionally, the TMS is utilized for shortening and lengthening the tether, so that the cable drag effect due to underwater currents is minimized. On the other side, the term Autonomous Underwater Vehicle (AUV) refers to vehicles that are totally autonomous and depend on their on-board power system and computational capabilities. These two types of underwater vehicles are very popular and are used in various domains like in military, oil & gas extraction, underwater structures/pipeline inspection, archaeology, marine science, ship maintenance and infrastructures deployment for renewable energy. The ROV usually features thrusters for control and navigation, buoyancy elements, lights, a wide range of sensors and maybe a robotic arm. Figure 2.1a illustrates a ROV, while Figure 2.1b depicts its motion notation. Below some of the most common sensors that ROVs use for navigation are presented.



(a) (b)

Figure 2.1: (a) Illustration of a ROV. (Courtesy of HPR UK) (b) ROV frames and motion notation. [2]

2.1.1 Sensors

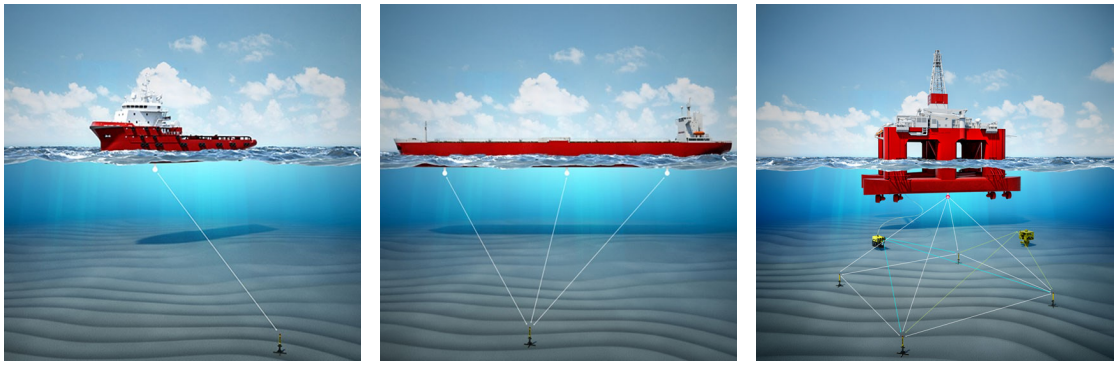
Acoustic Transponders

This category includes techniques that perform navigation by measuring the Time of Flight (ToF) of signals produced by acoustic beacons. A transponder is mounted on the ROV and communicates with a group of baseline stations, which are located on the vessel or the seabed, by receiving and transmitting signals. The stations have to be deployed prior to an operation. The operating frame rate depends on the water depth and commonly range at 0.3 - 1 Hz. The three most common methods are the Ultra Short Baseline (USBL), Short Baseline (SBL) and Long Baseline (LBL) (Figure 2.2).

In the case of USBL a set of transducers is compacted in a stand-alone device, which is mounted on the hull. The vessel then queries the transponder mounted on the ROV and estimates its position. In SBL three transducers with space between them are mounted on the vessel's hull. Also in this case a transponder is installed on the ROV and the vessel queries it. Finally, in LBL a set of transponders is deployed on the seabed in known positions. During the operation the ROV interrogates the transponders and calculates its distance to each of them by measuring the ToF. The acoustic positioning systems are expensive, they require time and staff to be deployed and the position error in deep waters might reach up to several meters.

Doppler Velocity Log (DVL)

The DVL captures the bottom tracking and estimates the velocity vector of a ROV, that moves across the seabed, by utilizing acoustic measurements. The heave, sway and surge velocities of the ROV are determined by measuring the Doppler shifted returns from pulses that are transmitted towards the seabed. DVLs usually consist of four or more beams. DVLs are quite expensive and they exhibit bad accuracy too close to the seabed.



(a) (b) (c)

Figure 2.2: (a) USBL, (b) SBL, (c) LBL. (Courtesy of Kongsberg)

Pressure Sensor

Pressure sensors provide the depth of the ROV by measuring the water pressure. These sensors are generally reliable and can achieve high accuracy, around 0.1 m. The measurements might be less accurate in the wave zone and due to salinity change [2].

Inertial Measurement Unit (IMU)

An IMU typically features three gyroscopes, three accelerometers and three magnetometers, providing measurements for linear acceleration, angular velocity and magnetic field components. The acquired data are combined to estimate the ROV's attitude. Some IMUs are small and inexpensive, thus they can be mounted on any ROV or AUV. An IMU usually operates at a range of 100 - 1000 Hz. It is very important to calibrate an IMU for higher accuracy. Despite the advances in their technology, IMUs suffer from noise and drift.

2.2 Underwater Optical Imaging and Challenges

With the advance of hardware and cameras the interest for using them underwater is constantly increasing. Optical imaging is very important for understanding the underwater environment and for various underwater applications, like robot navigation, seabed surveying, shipwrecks inspection, mapping of subsea areas and maintenance of man-made structures. With the contribution of computer vision a wide range of tasks is accomplished, like cable or pipe tracking, motion estimation, station keeping and localization.

However, subsea is not an ideal environment for computer vision, as inherently its nature negatively affects the image quality. While computer vision is highly based on features, underwater environment lacks of salient points on seabed and from structures in contrast to onshore. In addition, the natural light is not present in great depths, so most of the applications require artificial lighting, which by its turn introduces negative effects. Also, there are various unpredictable and uncontrollable factors like the clarity of the sea, conditions of surface and turbidity of water. Another fact is that the artificial

light source mounted on the robot attracts fish close to the ROV, resulting to partial all total loss of view. This can have severe impact on applications like motion estimation or station keeping. One more factor that has to be considered, is the presence of other ROVs in the field, meaning that their light source might reflect on the cameras of other ROVs, and as a consequence part or the whole camera view might be lost.

More specifically, according to [3] four significant issues that are commonly met underwater are the attenuation, scattering, image processing and distortion. The water rapidly absorbs the light, while this phenomenon is intensified as the depth increases until the natural light is completely eliminated. A side effect of this is the degradation of colors and especially of the red one. Scattering is caused by floating particles or bubbles that deflect the rays of light, resulting in a non-straight trajectory of the rays between the observed object and the camera. Scattering is further classified in forward and back scattering. Back-scattering is the phenomenon, during which light from the source is reflected back to the camera lens, and it has a negative impact on feature extraction and on the intensity of contrast. On the other hand, forward-scattering takes place when the light rays are deflected from their original path by a slight angle, which results in contrast degradation and blurriness. Another important factor is the introduced distortion. Due to the high levels of pressure underwater there is the need of appropriate housing for cameras and along with it, suitable depth-rated lenses. These lenses might suffer from design and production imperfection leading to distortion. Moreover, the refraction of light, caused by the difference of transmitting mediums (air, lens material, water), leads to non-linear image distortions. To compensate for the various types of introduced distortion, there is the need for camera calibration to find the camera intrinsic parameters and to use tangential and radial models, so that undistorted images can be produced. On top of that, the properties of underwater environment affect the post-processing of images as well. Due to the high light absorption and the other forms of noise, images must be captured close to the observed objects, so that a clearer view is acquired. This has as a consequence the limited available Field of View (FOV). Moreover, the use of artificial light source, mounted on the robot, causes non-uniform distribution of illumination, that produces moving shadows. All the aforementioned reasons along with the lack of salient features underwater, add extra requirements during post image processing, making harder to process underwater data and setting the algorithms more computationally expensive.

2.3 Camera Model and Calibration

2.3.1 Pinhole Camera Model

One of the most popular camera models, that is also considered in this thesis, is the so-called pinhole model. The pinhole camera model represents the mathematical formulation that relates the coordinates of a point in the 3D space and its projection on a 2D image plane of an ideal pinhole camera, in which no lens is used and the aperture is considered to be small, represented by a point. An inverted scene is projected on the image plane, as light rays are driven through the small aperture (see Figure 2.3). It is worth mentioning that the pinhole model does not consider lens distortion and to

represent a real camera accurately, also the radial and tangential lens distortion must be taken into account. This model represents a simple camera and it is used only for first order approximation for projecting a 3D scene to a 2D image. However, some effects that are not considered can be compensated and others can be neglected, setting the pinhole model as an adequate description of how a 3D scene is depicted by a camera and it is widely used in computer graphics and vision [4].

The perspective transformation used to project 3D points on an image plane is given from the following formula:

$$sm' = A[R|t]M' \tag{2.1}$$

or

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.2}$$

where:

- s is the scale
- (X, Y, Z) are the 3D coordinates of a point in world
- (u, v) are the 2D coordinates of the projected point in pixels
- A is the intrinsic matrix and contains the intrinsic camera parameters
- (c_x, c_y) is the principal point, usually located in the image center
- f_x, f_y are the focal lengths in pixel units
- $[R|t]$ is the extrinsic matrix, which contains the translation (t) and rotation (R) of the camera in the world.

It should be noted that the 2D and 3D coordinates are represented by homogeneous coordinates, that is why we observe an extra 1 along with them. The intrinsic matrix is independent from the observed scene. This means that once it is calculated it can be re-used if the focal length remains constant. The extrinsic matrix can be used to derive the camera motion in relevance to a static scene, and vice versa the rigid motion of an object in relevance to the camera. The product of intrinsic and extrinsic matrices is called camera matrix. The world points coordinates can be transformed to camera coordinates by using the extrinsic matrix. Afterwards, the camera coordinates can be transformed to image plane coordinates by utilizing the intrinsic matrix. More specifically, the extrinsic matrix $[R|t]$ transforms the coordinates of a 3D point (X, Y, Z) to the camera's coordinate system, and this transformation is equivalent to the following (when z $\neq$ 0):

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t \tag{2.3}$$

and by considering the representation in Figure 2.4 and geometrical relations we can derive that:

$$x' = \frac{x}{z}, \qquad y' = \frac{y}{z} \tag{2.4}$$

and

$$u = f_x * x' + c_x, \qquad v = f_y * y' + c_y \tag{2.5}$$

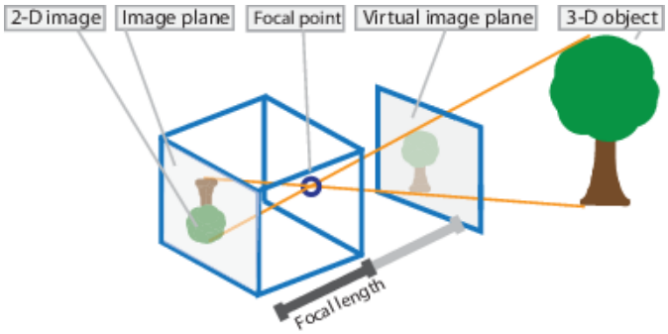


Figure 2.3: Pinhole camera model illustration. (Courtesy of MathWorks)

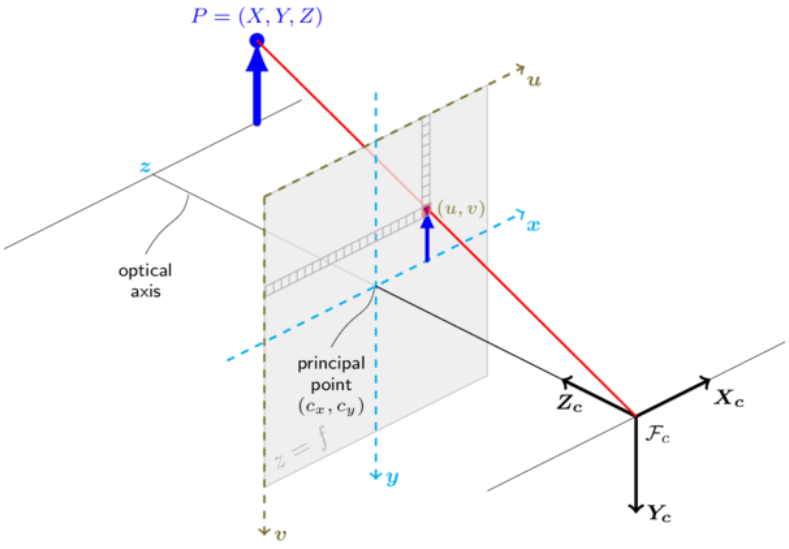
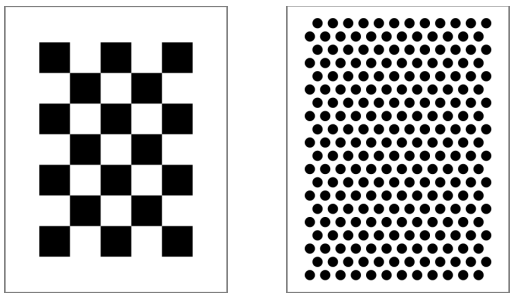


Figure 2.4: Detailed pinhole camera model representation. (Courtesy of OpenCV)

2.3.2 Camera Calibration

Camera calibration is the process, during which the lens and image sensor parameters of a camera are estimated. These parameters can be used for lens distortion correction, to estimate the relation between the real world units (meters) and the camera's units (pixels), to calculate the size of an object in real units and to determine the camera's position in a scene. All these tasks are usually applied in various domains, including robotics.

With calibration the camera matrix is estimated, and to achieve this we need 3D world points and their corresponding 2D image points and their coordinates. The required correspondences can be acquired by using calibration patterns such as in Figure 2.5, and by taking pictures of them from different viewpoints. Each snapshot corresponds to a number of equations which are combined to solve a system and to determine the camera matrix. Due to the reason that the solution might be biased, various viewpoints are utilized so that the conducted optimization converges to a more accurate solution



(a) (b)

Figure 2.5: Calibration patterns. (a) Chessboard pattern. (b) Circle Hexagonal Grid pattern.

and a more representative camera matrix.

For a real camera the radial and tangential distortion have to be considered. For the radial factor the following formulas are considered:

$$\begin{aligned} x_{corrected} &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ y_{corrected} &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{aligned} \tag{2.6}$$

with

$$r^2 = x^2 + y^2 \tag{2.7}$$

where k_1, k_2, k_3 are the radial distortion coefficients of the lens. The radial distortion is observed in the form of "barrel" or "fish-eye" effect. On the other hand, the tangential distortion occurs, because the camera lens is not completely parallel to the image plane. For the tangential factor the following formulas are considered:

$$\begin{aligned} x_{corrected} &= x + [2p_1 xy + p_2(r^2 + 2x^2)] \\ y_{corrected} &= y + [p_1(r^2 + 2y^2) + 2p_2 xy] \end{aligned} \tag{2.8}$$

where p_1, p_2 are the tangential distortion coefficients of the lens. Hence, given the initial pixel point at (x, y) the new $(x_{corrected}, y_{corrected})$ coordinates can be calculated. The distortion coefficients are presented by a one row matrix with five columns as follows:

$$Distortion_{coefficients} = (k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3) \tag{2.9}$$

It is worth mentioning that the distortion coefficients do not depend on the observed scene. Hence, they also belong in the intrinsic camera parameters and they remain the same no matter the image resolution. That practically means that if for example a camera has been calibrated with 320×240 resolution images, the same distortion coefficients will be valid also for 640×480 resolution images from the same camera, keeping in mind that f_x, f_y, c_x, c_y must be scaled properly [5].

2.4 Visual Odometry

VO is the process of estimating the position and orientation of an agent (such as vehicle, human, robot), and is based completely on data acquired through a single or multiple

cameras. Many applications utilize VO methods, including robotics, automotive, wearables and mobile computing, and augmented reality. The term was introduced for the first time in 2004 by Nister [6]. The term was inspired from wheel odometry, which estimates the distance that was traversed by accumulating the amount of wheel turns over time. Similarly, VO estimates the pose of the camera, and therefore of the agent, by observing the changes that take place from frame to frame, caused by the motion. For the VO to exhibit satisfactory performance, it is important to function under adequate illumination and in the environment there should be enough features or texture to derive the ongoing motion. In addition, there should be a good balance between dynamic and static scenes, with the latter ones being the dominant, while the successive frames should have ample overlap.

Wheel odometry exhibits bad performance and drift when used in slippery or rough terrain. On the other hand, VO has much better performance in such cases, resulting in very accurate trajectories, with an error in relative position ranging between $0.1-2\%$. For the VO to function, just a camera and a processing unit are required, elements that most of the vehicles and devices feature nowadays. Considering also the low cost of the optical sensors, VO is a great supplement to wheel odometry, and to other navigation systems like the IMU, the Global Positioning System (GPS), and laser odometry. However, the ultimate importance of VO is certainly in GPS denied environments, like subsea and in isolated aerial or ground places. [7]

The problem of estimating a vehicle's position and orientation by utilizing a camera dates back to 1980, and was described by Moravec during his PhD [8]. Moravec's work is considered a milestone in the development of the VO domain. He conducted experiments with a planetary rover equipped with a sliding camera to estimate its motion. In the subsequent years the VO research was dominated by National Aeronautics and Space Administration (NASA), while they were preparing their planetary rovers for the mission to Mars, aiming to provide their vehicles with the ability to estimate their 6 Degrees of Freedom (DoF) motion in all types of terrains. Still nowadays VO is most known for the NASA mission to Mars.

Additionally to the Mars mission, VO has been successfully applied to many other domains. It is commonly used for navigation, for target tracking and for obstacle avoidance. Furthermore, VO has been utilized in the aerial domain, for autonomous landing and take off. Another very important domain, that this thesis also addresses, is the underwater. Underwater environment is a GPS denied case and all the available navigation means are inaccurate or expensive to deploy. Therefore, VO plays an important role for underwater navigation and positioning, for hovering and obstacle avoidance. It has been utilized for coral-reef inspection and mapping, ship-wrecks observation, subsea caves and archaeological areas inspection and for surveying near the seabed. The automotive is another domain that has extensively used VO algorithms, for autonomous driving, parking assistance, vision based assisted breaking systems and localization when GPS is unavailable (e.g. in tunnels). In addition, ground robots make use of VO for indoor localization or in GPS denied environments, for good accuracy with low costs and when the robots are too small to carry extra heavy weight. Finally, VO has been exploited also in agriculture, in which robots estimate their position in relevance to the crops. [9]

2.4.1 Types of camera setups for VO

Stereo VO

Most of the research on VO domain is related to the stereo vision. Back in 1980 Moravec worked on a planetary rover with a single sliding camera. Although only one camera was used, his implementation is classified as a stereo vision algorithm. This was due to the way the frames were captured, i.e. the rover moved in a stop-and-go manner and in every position of the rover the single camera was sliding to capture nine images at equidistant intervals. However, the stereo vision systems consist of two cameras, which are spatially separated by a constant and known distance, called baseline (Figure 2.6a). The advantage of a stereo setup is that in every single time step, 3D information (depth) can be estimated by triangulation, by observing the same features in the left and right images. The motion can be estimated with only two consecutive frames, by observing features in the left and right image. In addition, the scale can be estimated as well. From Figure 2.6b it is derived:

$$\frac{Z+f}{Z} = \frac{X_l + X_r + B}{B} \tag{2.10}$$

or,

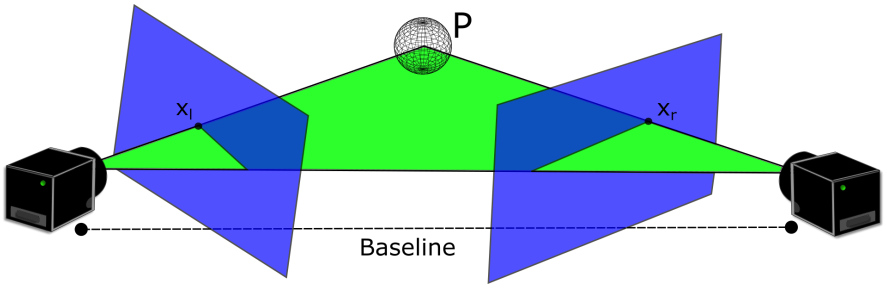
$$Z = \frac{fB}{X_l + X_r} \tag{2.11}$$

where X_l and X_r are the image coordinates of a 3D point for the left and right image correspondingly and $X_l + X_r$ is the disparity. In a stereo setup the focal length f of the cameras and the baseline B between them are known. If the disparity $X_l + X_r$ is computed, the depth Z can be calculated from 2.11. It is worth mentioning that the disparity and depth are inversely proportional. This practically means that an object located close to the stereo system will result in a large disparity, while if it is far away from the system the disparity will be small between the left and right images [10].

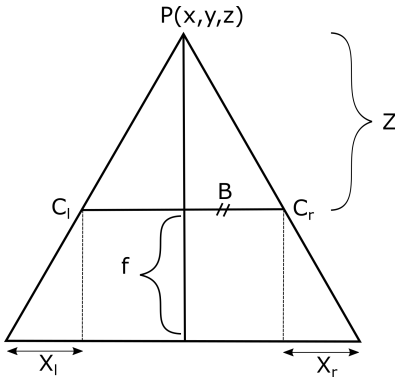
Many works have been proposed based on a stereo setup. The stereo vision approach is very popular, because it offers the absolute scale without additional measurements from other sensors. The fact that there is no need for other sensors and that it offers a decent accuracy, makes it an attractive solution, and it has been used extensively in ground, underwater and aerial robots. However, it comes with some drawbacks, like bigger mass, more electronics, need for synchronization, higher power needs and larger amount of data to be processed. A stereo camera is illustrated in Figure 2.7b.

Monocular VO

The alternative to the stereo setup is the monocular approach. The monocular setup is the simplest approach, as only a camera sensor is required to implement VO. It is also known as bearing-only, because only bearing information is available, in the sense that an amount of light passes through the focal center of the lens and corresponds to a specific pixel on the image plane [11]. The 3D scene is projected on the image plane, leading to the loss of depth information. However, if multiple poses of a single camera are combined, then the poses and the 3D structure of the environment is possible to be estimated. A



(a)



(b)

Figure 2.6: (a) Stereo vision setup, with two cameras separated by the baseline. (b) Stereo geometry. B = Baseline, f = Focal length, C_l and C_r = Camera centers, X_l and X_r = Image coordinates of P in left and right image, Z = depth [10].

major issue is that the motion and the 3D structure are estimated up to a relative scale. Usually, a scale is predefined between the first two frames, and subsequently the scale of the reconstructed 3D points and motion is relative to this. Furthermore, an initialization procedure is required to use the monocular approach. At least three consecutive frames are needed, to observe features in the first, re-observe the features and triangulate 3D points in the second, and finally estimate the motion in the third (see Figure 2.12a and Section 2.4.3). To derive the absolute scale an extra measurement is required from other sensors, like an IMU, a pressure sensor, or the real dimensions of an item placed in the scene have to be known. A worth mentioning point is that the stereo approach is limited to cases, in which the distance of the observed scene can not be much larger than the stereo baseline. If the distance to the scene is significantly larger, then the stereo approach degrades to the monocular one, as triangulation from the two cameras can not be achieved with sufficient accuracy [7].

Several applications utilize the monocular approach, due to its simplicity and low cost, such as simple mobile robots, cellphones and wearable gadgets. In [12] a monocular implementation for motion tracking with a cellphone is presented, in [13] a drone utilizes a monocular camera to navigate and avoid trees in a forest, while many other successful works have been proposed based on monocular setup. A monocular camera is illustrated

in Figure 2.7a.

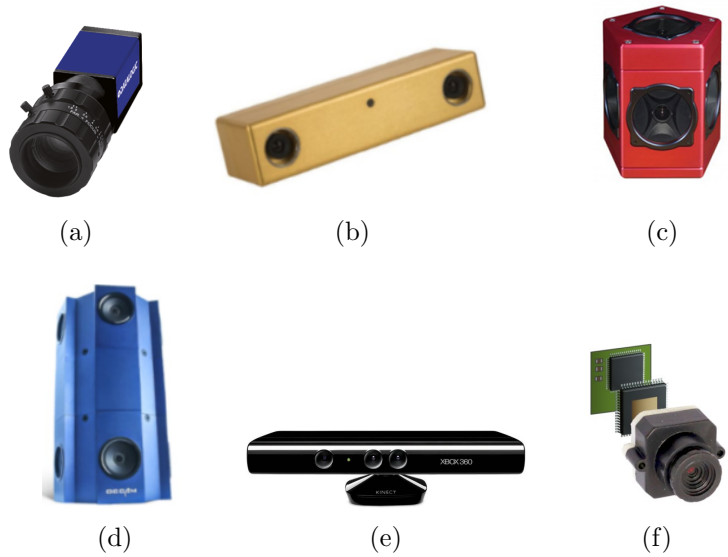
Other camera types

The stereo and monocular approaches are the most commonly used in VO. However, there are some other well established and some experimental approaches as well.

- **Omnidirectional cameras:** Omnidirectional cameras are cameras with a FOV greater than 180°, with the most common and popular ones reaching the 360°. Omnidirectional cameras are classified into the following designs: (a) Rotating camera systems, which utilize mechanical parts to rotate a single camera to capture a panoramic image, (b) Multiple camera systems, which are composed of two or more cameras tightly placed together in a circular manner and (c) Catadioptric systems, which utilize a single conventional camera facing a mirror that reflects the surrounding scene [14]. Omnidirectional cameras offer the advantage that a higher number of features can be observed and can be tracked for a longer period of time. Many approaches have utilized omnidirectional cameras for VO as well. A monocular and a stereo omnidirectional camera are illustrated in Figures 2.7c and 2.7d correspondingly.

- **RGB-D cameras:** RGB-D cameras are sensors that combine RGB camera images with depth information. The RGB-D cameras use ToF or active stereo [15] sensing to determine the depth for a large amount of pixels. The previous years these sensors were usually custom-built and they were expensive. Only the last years they have been massively produced with low cost, mainly from gaming and entertainment industry, making them attractive for the vision research community. The RGB-D cameras offer good accuracy in mid-resolution appearance and depth information at sufficiently high data rates [16]. The most popular sensor in this category is the Kinect (Figure 2.7e) produced by Microsoft, offering 3D point cloud information. In [16] the researchers implemented one of the initial approaches for RGB-D mapping utilizing the Kinect, while in [17, 18] they presented a depth only mapping implementation.

- **Event-based cameras:** The event-based cameras are inspired by the human visual system, in which cells in the eyes transmit signals to the brain whenever a change in the scene is observed, i.e. an event takes place. Similarly, whenever an event occurs the event-based cameras capture the change in pixel intensities asynchronously. Compared to the conventional cameras that capture frames in a fixed rate, with the event-based cameras a series of events is produced, encoding information about location, time and sign of intensity changes. The event-based cameras bring many advantages, such as very low latency, low power consumption, low redundant data, very high dynamic range and no motion blur. However, the traditional computer vision algorithms do not apply to this type of cameras, so the development of new algorithms is required. Additionally, similar to any other sensor the event-based cameras are prone to noise, and these noisy and non-ideal cases have to be modeled, so that more useful information can be acquired from events [19]. In Figure 2.7f an event-based sensor is presented.

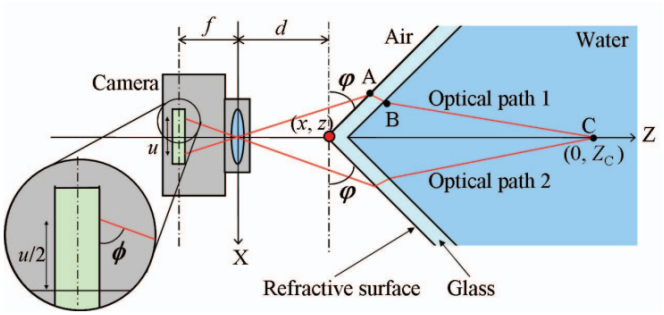


(a) (b) (c)

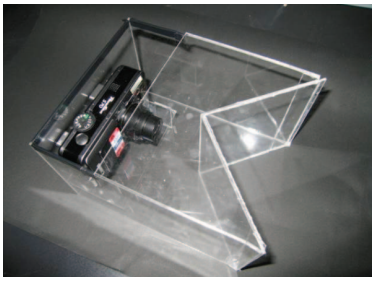
(d) (e) (f)

Figure 2.7: Various types of cameras for VO. (a) Monocular camera (property of Datalogic), (b) Stereo camera (property of VOLTRIUM), (c) Monocular omnidirectional, multiple camera system (property of FLIR), (d) Stereo omnidirectional, multiple camera system (property of Occam), (e) RGB-D camera, Kinect (property of Microsoft), (f) Event-based camera (property of iniLabs).

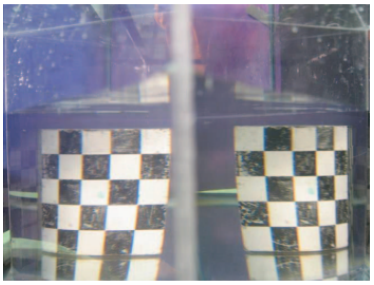
- **Experimental camera setups:** The aforementioned types of cameras are well established and massively produced. The two following designs are experimental, but interesting. With the proposed method in [20] a monocular setup is transformed into a stereo. This method exploits the refraction of light in different mediums of transmission, i.e. water and air. When a camera is placed in a housing, like the proposed in Figure 2.8b, image distortion occurs by the refraction of light on the boundary between air and water, and different optical paths are produced, obtaining the result shown in Figure 2.8c. The advantage of this approach is that the optical paths are produced with a very low cost housing and by exploiting the two different mediums of transmission. From the acquired image the depth is possible to be estimated, similar to a stereo setup. The accuracy of the depth estimation depends on the angle ϕ of the refractive surface. The optimal angle of the refractive surface depends on the distance Z_c between the camera and the object (Figure 2.8a). However, the distance between the camera and the object is unknown before the experiments and the conducted measurements. Therefore, we consider that the angle should be optimized in a certain range of distances, that are very likely to be observed. Another approach for obtaining stereo system images with a single camera is proposed in [21]. This system is a cheap implementation with no moving parts and composed by a beam splitter and three mirrors (Figure 2.9a). The obtained result is presented in Figure 2.9b. Similar to the previous proposed approach, the depth can be estimated, as in a stereo system, from the acquired image. According to the authors, this structure can be constructed in various dimensions, simulating different types of baseline lengths,



(a)

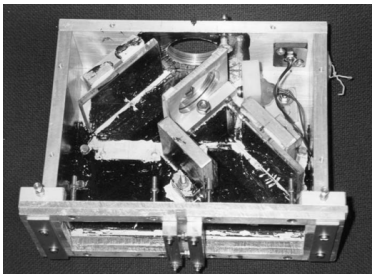


(b)

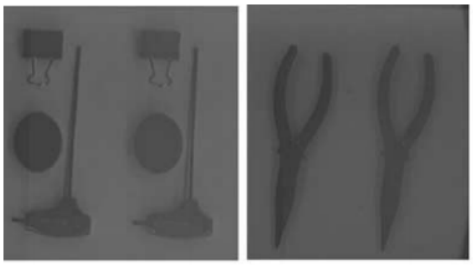


(c)

Figure 2.8: (a) Geometry and parameters of the refractive surface. (b) Camera housing that transforms monocular to a stereo system. The case is made of acrylic plates. (c) The obtained result when the camera with the case are placed into water.



(a)



(b)

Figure 2.9: (a) Apparatus that transforms a monocular to a stereo system. It is composed by a beam splitter and three mirrors. (b) The obtained image using the apparatus in the left picture.

while it is aiming to robot applications. The two aforementioned experimental designs could be especially interesting and inspiring for companies or researchers that design and produce their own housings for their robot applications.

2.4.2 Definition of VO

According to [7] VO is defined as the motion estimation from an amount of consecutive camera frames, considering the camera to be rigidly mounted on a robot. In monocular setup the set of frames is denoted as $I_{0:n} = \{I_0, ..., I_n\}$, where its frame is captured at the instant $k = 0...n$. For the stereo setup there are two set of frames, for the left and for the right camera, denoted as $I_{l,0:n} = \{I_{l,0}, ..., I_{l,n}\}$ and $I_{r,0:n} = \{I_{r,0}, ..., I_{r,n}\}$ correspondingly. For the purpose of simplicity, it is assumed that the robot's coordinate system is the same as its camera coordinate system. In the case of the stereo system, the left camera can be used as the reference point. The relation between two consecutive frames, acquired at time instants $k-1$ and k, is defined by the transformation $T_{k,k-1} \in \mathbb{R}^{4\times4}$, given from the form:

$$T_{k,k-1} = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix}, \tag{2.12}$$

where $R_{k,k-1} \in SO(3)$ is the rotation matrix, and $t_{k,k-1} \in \mathbb{R}^{3\times1}$ is the translation vector. All the consecutive motions are represented by the set $T_{1:n} = \{T_{1,0}, ..., T_{n,n-1}\}$. The set $C_{0:n} = \{C_0, ..., C_n\}$ includes all the camera poses with respect to the initial pose C_0, which can be set by the user arbitrarily. A pose C_n in time instant $k = n$, is calculated by the product of all the previous transformations T_k, where $k = 1...n$, hence $C_n = C_{n-1}T_n$. The motion transformations and the poses are illustrated with an example in Figure 2.10, where the poses C_1, C_2, C_3 denote a trajectory different from the $C1, C2'$. The purpose of VO is to estimate all the subsequent transformations, to calculate a trajectory. This means that a trajectory is calculated incrementally pose after pose. After this step a refinement of the last m poses can be performed to estimate more accurately the last part of the trajectory. This is achieved by a method called windowed-bundle adjustment, which is described in a subsequent section.

VO can be implemented with two different approaches, by using feature-based or appearance-based methods, which are described in the subsequent Sections (2.4.6, 2.4.11). Feature-based methods are based on salient points in an image, while appearance-based methods on pixel intensities. The feature-based methods are more robust and accurate, being invariant to rotation, scale and illumination changes, and in different viewpoints and occlusions as well. For this reason many VO algorithms are based on feature-based methods. A basic VO pipeline is presented in Figure 2.11. Initially, frames are acquired and features are detected in both of them. Afterwards, image correspondences in these two consecutive frames are located. A correspondence is defined as the case in which two 2D features in different frames are reprojections of the same 3D point. The correspondences are matched or tracked (Section 2.4.7) between two subsequent frames. In the next step the relative motion is estimated between the frames using one of the three approaches, 2D to 2D, 3D to 3D, 3D to 2D, which are described in the Section 2.4.3. Finally, bundle adjustment is applied to estimate the local trajectory with higher accuracy (Section 2.4.12).

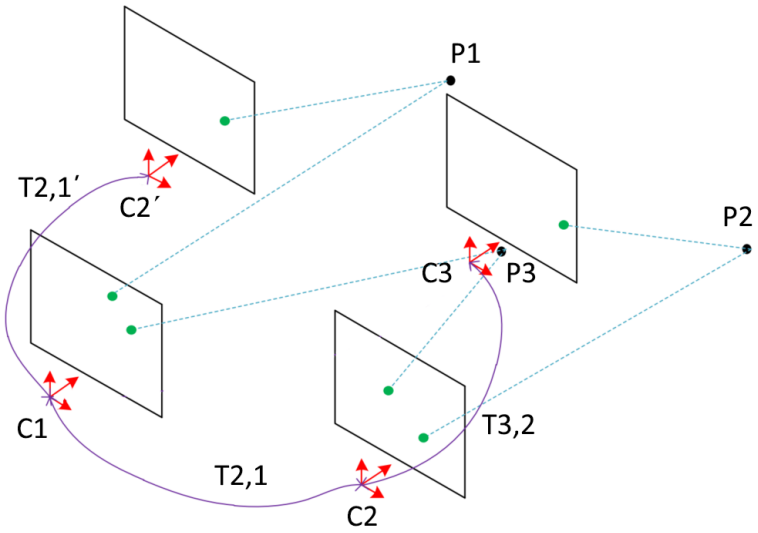


Figure 2.10: Illustration of VO formulation. T denotes the transformations, C the poses, and P the 3D points. The cyan dashed lines depict the triangulation of 3D points from different poses, while the purple lines depict the motion of the agent. [22]

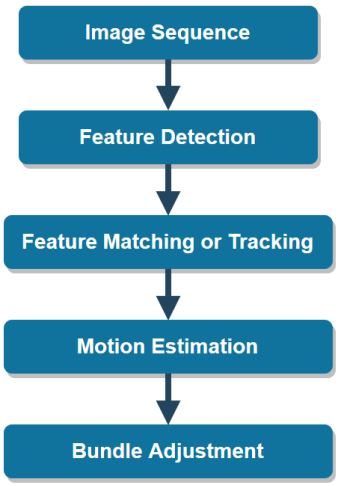


Figure 2.11: A basic VO pipeline. [7]

2.4.3 Motion Estimation

2D to 2D

This approach is utilized when 3D information is not available. This happens in the case of a monocular system, in which during the initialization the only available data are the 2D features from the first two frames. These 2D data are utilized to derive the transformation between the two initial frames. Afterwards, the 2D data are associated and the first 3D points are calculated by triangulation. This procedure is depicted in Figure 2.12a, with the first two frames corresponding to the camera poses C_1, C_2.

The essential matrix E describes the geometric relations between two frames and is given by the following formula:

$$E = [t] \times R, \tag{2.13}$$

where R is the rotation matrix, t the translation vector given by:

$$t = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}, \tag{2.14}$$

and

$$[t] = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & t_y \\ -t_y & t_x & 0 \end{bmatrix}. \tag{2.15}$$

Using epipolar geometry and the 2D feature correspondences the essential matrix can be calculated. Afterwards, rotation and translation can by derived by decomposing the essential matrix E. Practically, given the matrix E and the coordinates of a feature in one frame, we can estimate the feature's location in the other frame using the epipolar constraint equation:

$$u_i'^T E u_i = 0 \tag{2.16}$$

where u_i is a feature's location in one frame and u_i' the location of its corresponding feature in the other frame (Figure 2.12b). More details can be found in [7].

3D to 3D

This approach requires 3D information for both of the two consecutive frames, therefore it is applicable only for stereo systems. The transformation T is estimated by triangulating 3D feature points in successive frames and by minimizing the 3D Euclidean distance between the associated 3D points using the following formula:

$$T = \underset{T}{\operatorname{argmin}} \sum_i |X_i - T X_i'|^2 \tag{2.17}$$

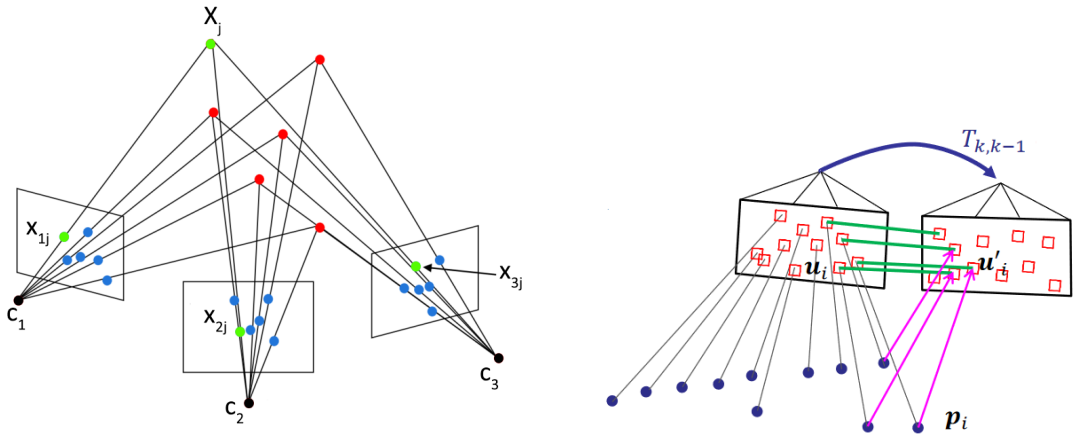
where X_i is the 3D point observed by the current frame, while the X_i' is its correspondence observed by the previous frame. The i denotes the ith feature.

3D to 2D

This approach is similar to the previous one, however it is more accurate, as it minimizes the 2D re-projection error, instead of minimizing the 3D to 3D feature point position error. The formula that minimizes the re-projection error and computes the transformation T is given by:

$$T = \underset{T}{\operatorname{argmin}} \sum_i |z - f(T, X_i')|^2 \tag{2.18}$$

where z is the 2D feature observed in the current frame and $f(T, X_i')$ is the re-projection function of its 3D corresponding point from the previous frame, on which the transformation T is applied. In Figure 2.12a the 2D to 2D motion estimation is used for the two



(a) (b)

Figure 2.12: (a) Illustration of 3D to 2D motion estimation, and representation of initialization procedure for monocular approach. X_j represents the feature in 3D and the x_{1j}, x_{2j}, x_{3j} in 2D coordinates. C_1, C_2, C_3 depict three different poses [23]. (b) 3D to 2D re-projection. $u_i, u_i\prime$ are the 2D coordinates of the 3D points in two subsequent frames. The transformation is denoted as $T_{k,k-1}$ [24].

Table 2.1: Motion Estimation Approaches

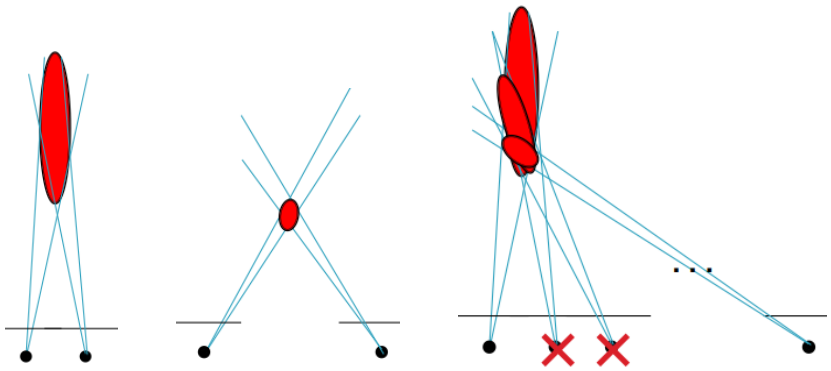
Motion Estimation Approach	Monocular	Stereo
2D to 2D	✓	✓
3D to 3D		✓
3D to 2D	✓	✓

initial poses C_1, C_2. After the 3D points coordinates have been calculated, the 3D to 2D approach is used for the third pose C_3. The final step is also illustrated more detailed in Figure 2.12b.

The minimum number of features required to calculate the transformation, in the aforementioned approaches, is based on the system's DoF and the utilized model. If more features than the minimum required are used, the accuracy increases against the computational efficiency [22]. In Table 2.1 the motion estimation approaches and their relation to monocular and stereo setups are presented.

2.4.4 Scale and Depth Estimation

To calculate a trajectory, the successive transformations $T_{0:n}$ have to be concatenated. To achieve this the relative scales have to be computed as well. While with a stereo vision setup the absolute scale can be estimated immediately with triangulation, in the case of monocular setup a relative scale is considered. To estimate the absolute scale there is the need of an additional measurement from another sensor, for example an IMU. However, as it is stated in [7], it is possible to compute the relative scales of con-



(a) (b) (c)

Figure 2.13: The red ellipse indicates the depth uncertainty. A 3D point could be located anywhere inside the ellipse. (a) High depth uncertainty. (b) Low depth uncertainty. (c) Keyframes selection. [24]

secutive transformations by using 3D points coordinates observed from two subsequent viewpoints, with the following equation:

$$r = \frac{\|X_{k-1,i} - X_{k-1,j}\|}{\|X_{k,i} - X_{k,j}\|} \tag{2.19}$$

where X_i and X_j are two 3D points observed from two subsequent frames in steps $k-1$ and k. Their relevant distance is calculated for both of the viewpoints and the ratio r is calculated. For a more robust result the ratio r is calculated by considering many 3D points and the mean is used. Afterwards, the translation t can be scaled with the derived distance ratio r. With this method common points between frames are utilized to propagate the scale factor. It is worth mentioning that the scale of translation can vary between frame pairs. Furthermore, it has to be noted that errors are gradually accumulated and lead to scale drift.

When 3D points are triangulated from two frames to estimate the scale, it is important to choose the most suitable frames that will result in the least possible uncertainty. If we use frames that are very close to each other compared to the scene distance, high uncertainty is introduced. In Figure 2.13a the red ellipse depicts the large depth uncertainty, i.e. the 3D point could be located anywhere in the red ellipse. On the other hand in Figure 2.13b the utilized frames have adequate distance relevant to the scene and the depth uncertainty is significantly reduced. A proper solution to this problem would be to skip frames until the average depth uncertainty is lower than a certain threshold (Figure 2.13c). The frames that are eventually chosen are called keyframes. According to [24] an indicative rule to choose the proper keyframes is given by the following equation:

$$\frac{keyframe\ distance}{average\ depth} > threshold \tag{2.20}$$

In the case of a monocular approach the absolute scale could be estimated by triangulating 3D points using keyframes and measurements from an extra sensor. Similarly

to the stereo setup, where the absolute scale is calculated immediately from the two cameras, two keyframes can be used to triangulate 3D points, while the real distance (baseline) between these keyframes can be estimated using an IMU.

2.4.5 VO, VIO and V-SLAM, VI-SLAM

VO and V-SLAM are two terms that are closely related. As it is described in [7] the purpose of V-SLAM is to estimate and maintain a global and consistent robot trajectory. This requires environment mapping, even if a map is not needed, because the robot must have the ability to realize if it is located in a place that has already been visited before. The process of recognizing already visited areas is called loop closure and it is a basic component of V-SLAM. If a loop closure takes place, a new constraint is introduced and causes a reduction in path drift, while it refines the global trajectory and the map as it is illustrated in Figure 2.15. In Figure 2.15a it is depicted that the robot started its trajectory from the green area in bottom and moved to the right. We can observe that a significant accumulative error and drift are introduced. However, if V-SLAM is utilized, when the robot revisits the green area and realizes that it has been there before, an iterative back-propagation refinement for the whole previous trajectory and map is executed, resulting in the path depicted in Figure 2.15b. Here we notice the difference between VO and V-SLAM, because in VO only a local refinement takes place with Bundle Adjustment (BA) (described in Section 2.4.12).

VO constitutes a component of V-SLAM as it is also noted in Figure 2.14 and in the following equation:

$$V\text{-}SLAM = VO + loop\ detection + global\ map\ optimization. \tag{2.21}$$

It is worth mentioning that V-SLAM can still be used instead of VO even if only the robot path estimation is the main goal. V-SLAM offers greater accuracy and less drift in trajectory estimation. However, V-SLAM comes with a risk, that if a place is falsely recognized as revisited it will introduce severe drift. In the end, it is up to the user to decide if VO or V-SLAM is suitable for a problem, considering a trade-off between consistency, performance and implementation complexity. VO might be more suitable for a real-time performance, because it does not keep track of a map or the whole past trajectory. In the literature review and in the implementation for this thesis, both VO and V-SLAM algorithms are considered. Here it has to be pointed out that V-SLAM has many other aspects to consider as well, like efficient map building, however these are out of the scope of this thesis.

The terms VIO and VI-SLAM refer to the case where VO or V-SLAM is combined with an inertial sensor, like an IMU. The inertial sensor might be used to compute the scale for the monocular cases, or even to increase the accuracy of the algorithms by combining data from both, the camera and the IMU.

2.4.6 Feature-based Methods

The feature-based methods are also known as sparse methods. They are based on feature detectors and descriptors. A detector searches for salient features, like corners, edges or

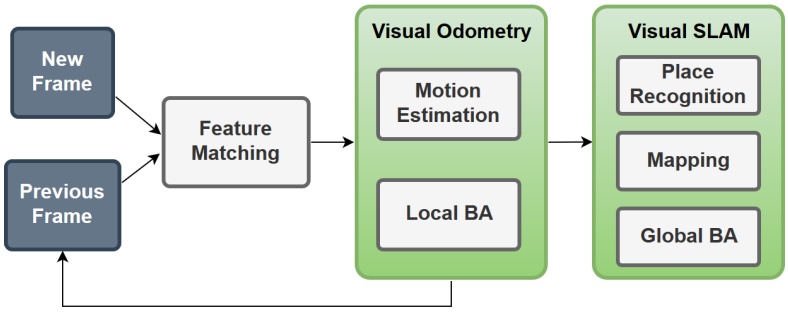
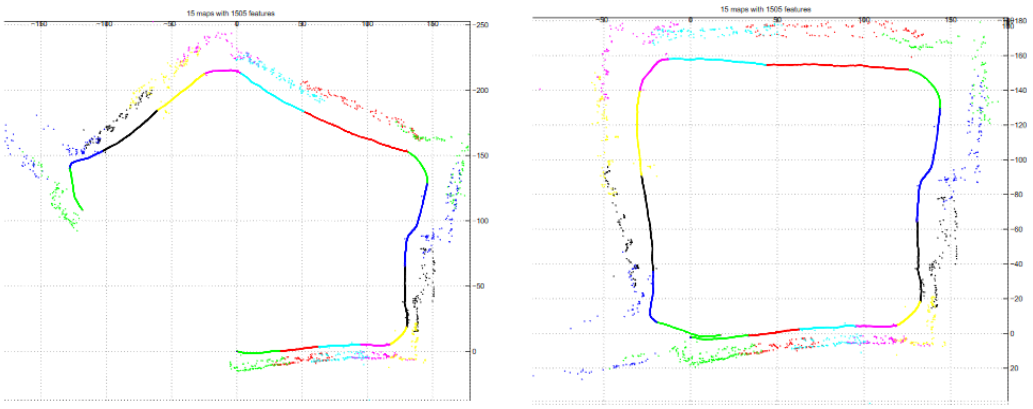


Figure 2.14: VO and V-SLAM relation. [25]



(a) (b)

Figure 2.15: (a) Trajectory with VO. (b) Trajectory with V-SLAM. [26]

blobs, and extracts them. Subsequently, a descriptor is used to describe the characteristics of the area surrounding the feature, in a way that the feature can be identified again in another image. There are various implementations of detectors and descriptors, each one with its advantages and disadvantages. The most robust descriptors are considered to be those that are invariant against a variety of changes, like in illumination, scale, viewpoint, occlusions, rotation and noise. As usual there is a trade-off between performance and accuracy. Some detectors or descriptors exhibit exceptional accuracy, however they are computationally hungry, and in most cases they are inefficient to be used, due to the high demand of power and the inability to function in real-time. Some implementations come with a detector and descriptor, while some others are only descriptors or detectors. It is commonly seen that many implementations are based on a various combinations of descriptors and detectors, leading to a wide range of different performances. The combination of a detector-descriptor should be chosen in a way, so that they operate optimally together. Some researchers claim that the built-in detectors of some descriptors are the optimal ones, however experiments have proved that a careful blend can lead to satisfactory performance. There is not an ideal combination nor a perfect descriptor or detector for all the cases. The performance of the most promising ones has to be examined for every individual case, due to the variety of the environments and the requirements or

limitations of performance and accuracy. Choosing and deploying detectors and descriptors is not a trivial task. A significant challenge is to properly tune the parameters of the algorithms so that a desirable balance can be achieved between a tolerable accuracy and computational efficiency. Various parameters might affect the performance, like the number of features tracked or the Random Sample Consensus (RANSAC) iterations. It is also important that the features are uniformly distributed across the whole image to avoid biased motion estimations. The most popular feature detectors and descriptors are described below.

- **Harris Corner:** The Harris corner detector [27] is one of the most successful and established methods for point detection, and it is based on the Moravecs algorithm [8]. In the Harris algorithm, for every pixel in the image a matrix, that represents the curvatures of image intensities, is formed. These curvatures are proportional to the matrix eigenvalues and represent how quickly the image intensity changes in relevance to small position changes. The eigenvalues are utilized to determine if a point is part of an edge, a corner, or a uniform region. A point is considered to be a corner if it is characterized by two large eigenvalues, corresponding to a high intensity change in both directions.

- **Shi-Tomasi Corner:** The Shi-Tomasi corner detector [28] is another classic method for feature extraction and it is based on Harris detector, but with a slight variation in the way that the corners are defined. The Shi-Tomasi performs better than the Harris, due to its improved approach on the scoring function (selection criteria).

- **Scale Invariant Feature Transform (SIFT):** SIFT is known as one of the most robust methods for feature extraction and it was proposed in 1999 by David Lowe [29]. It detects blobs, which are something different from corners and edges. Blobs are image patterns that are characterized by color, intensity and texture, that are different from the adjacent neighborhood. Its robustness is due to its invariance against scale, rotation and up to some level against illumination, viewpoint and occlusions. The main disadvantage is that the algorithm is very computationally expensive. SIFT features are detected by applying the Difference of Gaussian (DoG) and by searching for local extremas at different scales. The found extremas with high local contrast that are not edges, are considered to be the keypoints. SIFT has also its own descriptor, which is derived by computing the gradients orientation and magnitude in a 16×16 window around each keypoint, which is then divided in 4×4 sub-regions. In these sub-regions, weighted histograms of 8 bins are formed. These histograms are then gathered and the information is stored in a $4 \times 4 \times 8 = 128$ element vector. An example of a SIFT descriptor is shown in Figure 2.16.

- **Speeded-Up Robust Features (SURF):** SURF is highly inspired by the SIFT algorithm and is a blob detector and descriptor as well. SURF was introduced in 2006 [31] and it was aimed to be faster than SIFT. Compared to SIFT that uses the DoG to approximate the Laplacian of Gaussian (LoG), SURF uses box filters to estimate the second-order derivative and integral images to approximate the Hessian matrix. The Hessian matrix is analyzed to locate extremas in different scales. SURF

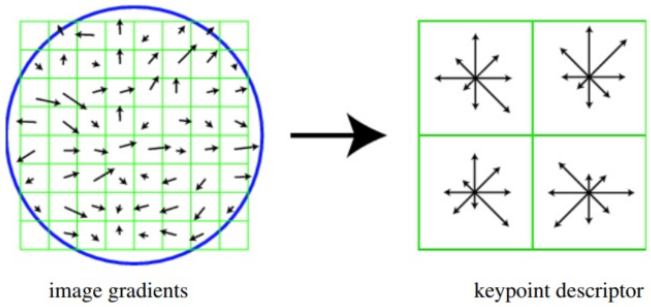


Figure 2.16: SIFT descriptor. The left part of the figure illustrates the image gradients orientations and magnitudes. The right part of the figure illustrates the weighted gradient orientation histogram. In the figure 8×8 window and 2×2 sub-region are shown as an example. [30]

features a similar to SIFT descriptor, but less computationally demanding. Each keypoint is assigned an orientation, based on summed up wavelet responses in a circular area around the keypoint. For applications that rotation invariance is not important, there is the version of Upright-SURF (U-SURF), which is even faster. More information for the SURF algorithm is available in [32]. SURF is more efficient than SIFT in terms of computational time, however the latter one is more robust. SURF is invariant to scale, rotation and illumination as well.

- **Features from Accelerated Segment Test (FAST):** FAST [33] is a corner detector, which has the advantage of being computationally efficient and faster than other detectors like Harris. To specify if a region is an edge, a corner or a uniform area, 16 pixels are considered around a center pixel. The intensity values of these 16 pixels are compared to the center pixel, and based on the percentage of similarities among them, the type of region is defined. FAST is considered suitable for real-time applications, however is very sensitive to noise, while a threshold have to be taken into account.

- **Binary Robust Independent Elementary Features (BRIEF):** BRIEF is an efficient feature descriptor presented recently in [34]. The descriptor is derived by comparing the intensities of randomly selected pairs of pixels around a center pixel. With this information binary strings are formed, which describe the surrounding region of the keypoint. BRIEF is considered to be the first approach of binary descriptors, in contrast to others, like SIFT that uses vectors of floating point numbers. For keypoint matching, the Hamming distance is utilized, which results in a higher computing performance. BRIEF is just a descriptor, so it has to be combined with a detector, like Harris, FAST or SIFT. Finally, BRIEF is invariant to changes in illumination, however is not invariant against rotation and scale.

- **Oriented FAST and Rotated BRIEF (ORB):** ORB [35] is based on the FAST and BRIEF algorithms, and is considered to be an efficient alternative to other feature descriptors like SURF and SIFT. The ORB is scale invariant, because it detects

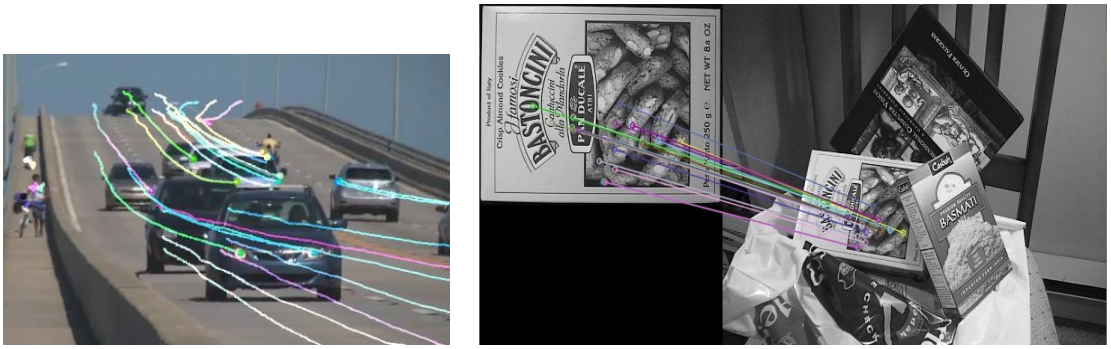
corners in a scale pyramid, by using an improved version of FAST. Then the best of the detected corners are chosen, by utilizing the Harris corner score. To obtain rotation invariance, the rotation is computed through an intensity centroid. More details on the mathematics of this method can be found in [36]. In the next step, a modified version of BRIEF descriptor is computed for every keypoint. ORB is gradually widely adapted, especially in embedded systems for robotics, due to its lower computational requirements. It is also invariant to illumination and viewpoint changes.

- **Binary Robust Invariant Scalable Keypoints (BRISK):** BRISK [37] is a detector and descriptor, which improves the BRIEF approach. Although there are other robust algorithms, such as SIFT or SURF, which are invariant to scale, rotation, viewpoint and many transformations, their computationally expensive descriptors are not the optimal choice for mobile applications. To cope with this challenge, various binary descriptors have been proposed and BRISK is one of them. BRISK utilizes an improved version of FAST to extract corners in a pyramid of different scales, thus making it scale invariant. BRISK descriptor differs from the BRIEF in the sense that intensity comparisons are conducted, using sampling patterns, corresponding to points spread around the keypoint in concentric circles. These pairs of points are divided in two groups of short-distance and long-distance pairs. Orientation invariance is achieved by using the aforementioned pairs to estimate and assign the keypoint orientation. This process leads to 512 pairs that form the 512 element binary descriptor. Similar to BRIEF, for descriptor matching the Hamming distance is used, resulting in a computationally better performance. [38]

2.4.7 Feature Matching and Feature Tracking

The next step after detecting features in two images, is to match them. The purpose of this step is to locate the same feature in both images, after it has undergone in changes in rotation, translation or scale. The term feature matching is used when features are matched across multiple images, while great changes might have also taken place (Figure 2.17b). Feature tracking is the procedure, during which the same feature is matched/tracked in consecutive frames. For feature tracking successive frames are considered, in which small changes in motion and appearance are observed. Furthermore, the search for features is done in a nearby location to the previous one, in contrast to feature matching, during which the whole image is scanned. The most popular tracking method is the Kanade-Lucas-Tomasi (KLT) tracker [28], which is illustrated with an example in Figure 2.17a. Another term that commonly appears is Data Association (DA). DA term is used for feature matching, during Simultaneous localization and mapping (SLAM) methods, in which features are searched in all the previous frames to determine if a place captured by these frames have been visited in the past.

To match features a similarity measure/distance has to be defined. Two features are associated when the distance between them is the minimum. The simplest way to match features is to do a brute-force comparison, by comparing all the feature descriptors in one image to the descriptors of the other, considering a similarity distance. Some common distance formulas are the Euclidean and the Hamming distance. If the descriptor is based



(a) (b)

Figure 2.17: (a) KLT tracker. The colored lines denote the track of various features across consecutive frames [39]. (b) Illustration of feature matching [40].

on the appearance of the area surrounding the feature, then the most common similarity measures are the Sum of Square Differences (SSD), Sum of Absolute Differences (SAD) and Normalized Cross Correlation (NCC). In SSD the squared difference between pixel intensities of the two images is considered, while in the SAD the absolute differences. The SSD is more robust compared to the SAD, but also more computationally demanding. On the other hand, the NCC sums the product of the intensities of corresponding pixels in patches, belonging to the two images, and then normalizes with a value that depends on those intensities. A high similarity between these two compared patches is denoted by a high NCC value. The NCC is one of the most popular and accurate methods for calculating the similarity of two images, outperforming the SSD and SAD, however it is computationally more expensive [22]. Finally, to raise the probability for correct matching a mutual consistency check can be used. Initially, the descriptors of the first image are examined and matched to descriptors of the second image. However, some descriptors in the second image may be matched with more than one from the first image. To eliminate these cases, the procedure is repeated, but this time comparing the descriptors of the second image to the ones of the first. Only the descriptors that were matched in both aforementioned processes are considered in the end.

2.4.8 Kanade-Lucas-Tomasi (KLT) Tracker

The KLT tracker is one of the most popular feature tracking algorithms initially introduced in [41]. Features are extracted with the Shi-Tomasi corner detector, described in Section 2.4.6, and then they are tracked in successive frames. The KLT does not calculate sophisticated descriptors for every feature, but it utilizes the intensity values of a patch (square window) around the feature to describe the feature's neighborhood. This offers the advantage of fast computations, but it is sensitive to illumination, rotation and viewpoint changes. The sensitivity to illumination variations can be mitigated by normalizing the patch, by scaling the intensity values in a specific range and by subtracting the mean. The patch is tracked in successive frames by utilizing a Newton - Gauss method to estimate a displacement that leads to the minimization of the sum of squared

differences. An example illustrating the KLT tracker is presented in Figure 2.17a.

2.4.9 Outlier rejection (RANSAC)

A significant part of a VO pipeline is the outlier removal, i.e. rejecting wrong feature associations. Ideally features would match only with their unique correspondences. However, in practice this does not happen for multiple reasons. A feature can be mismatched, because a frame might contain reoccurring similar patterns, such as a fence or a building with windows. In addition, outliers might occur due to sensor noise, occlusions, motion blur, and changes in illumination or in viewpoint. The accuracy of a VO implementation is sensitive to outliers, and for this reason they have to be eliminated.

The most commonly used and effective method is RANSAC. The main concept of RANSAC is that some data points are randomly chosen from a set, and a mathematical model hypothesis is formulated. Subsequently, this hypothesis is applied to the rest of the data points and then it is checked if the hypothesis stands also for them. The model hypothesis that is chosen is the one that is true for most of the data points. In our case the model is the estimated transformation (rotation, translation) and the data points are the feature correspondences. The algorithm is outlined in the following steps:

1. Randomly select the minimum amount of features that are required to derive the transformation.
2. With the chosen features estimate the transformation.
3. Apply the derived transformation to the rest of the features.
4. Calculate the distance d between the transformed points and their correspondences.
5. Considering a specific threshold t, determine how many of these features are inliers, i.e. the matches for which $d < t$ is true.
6. Repeat the previous steps for a predefined number of iterations n.
7. The transformation, with which the maximum number of inliers is observed, is the solution.
8. Estimate again the transformation, using only the inliers.

It is worth mentioning that RANSAC is not a deterministic algorithm, resulting in a different solution every time it is executed. As the number of iterations grows, the result tends to be stable [22]. A RANSAC example is illustrated in Figure 2.18.

2.4.10 Direct and Indirect Methods

Before moving on the description of the next VO approach, which is the appearance-based or dense methods, a brief clarification of two other commonly used terms has to be mentioned. The terms are Direct and Indirect methods, not to be confused with Dense and Sparse methods. Behind all the formulations there is a probabilistic model that considers as input noisy measurements and estimates the camera motion and the

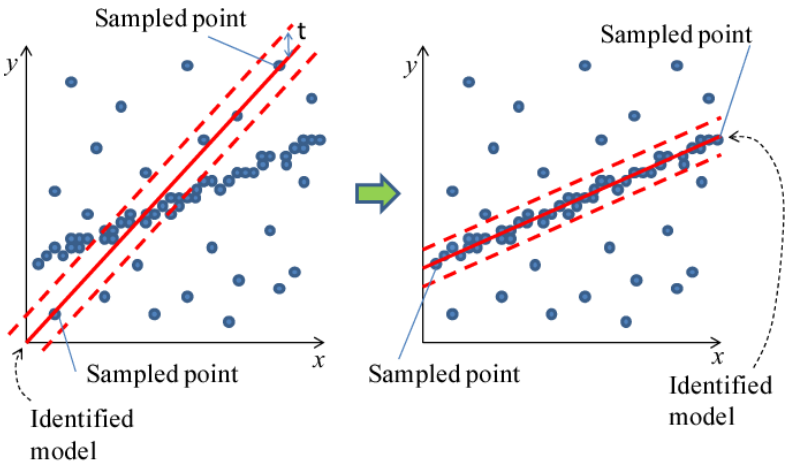


Figure 2.18: A RANSAC example. The points that are located between the dashed lines are the inliers, while those that are positioned outside the dashed lines are the outliers. [42]

3D world model. Direct methods aim to minimize a photometric error between corresponding pixels in successive frames, utilizing the sensor photometric measurements. The optimization lies on aligning camera poses, by considering the re-projected scene and how much the pixel intensities moved in the image. On the other hand, indirect methods minimize a geometric error, i.e. a re-projection error of corresponding 3D points and their observed projections in successive frames, as the estimated point positions are geometric quantities. However, the sparse term refers to the case that only certain points from the image are used for motion estimation and reconstruction, whereas the dense term refers to the case that all of the pixels from an image are utilized. It is commonly observed that indirect methods are combined with sparse approaches, by extracting and matching local features in images.

2.4.11 Appearance-based Methods

The appearance-based methods are also known as dense methods. In contrast to the feature-based methods, the appearance of an image is observed for changes, by extracting information from the pixel intensities. Since the intensity from all the pixels is utilized, much more information is extracted from the scene, setting the appearance-based methods more suitable for texture-less environments. While a feature-based method would probably fail if an image was representing a wall or sand, due the lack of features, a dense method would be still able to estimate the motion of an agent. However, appearance-based methods have a narrower baseline for matching, when compared to the featured-based methods. This results to a negative impact on reconstruction accuracy, which demands a broad baseline of features to limit the depth uncertainty. On top of that, they can only track and estimate motion in subsequent frames that do not include rapid changes. The appearance-based techniques also require accurate modeling, because they are sensitive to the artifacts caused by auto-gain, rolling shutter and

auto-exposure, while their computationally demanding nature has a negative affect on the mapping procedure. On the other hand the feature-based methods are capable on matching features on a wide baseline and can capture greater motion variations, as they are invariant to scale, illumination and viewpoint changes.

The appearance-based methods are based on Optical Flow (OF). The OF technique considers the intensities of adjacent pixels to estimate the displacement of intensity patterns from frame to frame. Using OF the relative motion between objects and a viewer can be estimated, while the value of OF for every pixel denotes how much the intensity of the pixel has moved in successive frames. The *Intensity Coherence* assumption is used to calculate the OF. According to this assumption, the pixels intensities of a point projected in two sequential frames are constant or approximately constant [43]. This results in the OF constraint equation:

$$\frac{\partial I}{\partial x}u_x + \frac{\partial I}{\partial y}v_y + \frac{\partial I}{\partial t} = 0 \tag{2.22}$$

where u_x and v_y are the optical flow components in x and y, $\frac{\partial I}{\partial x}$ and $\frac{\partial I}{\partial y}$ are the image gradients along x and y, and $\frac{\partial I}{\partial t}$ is the gradient along time (difference over frames). The OF problem can be solved by using a number of various proposed algorithms.

The OF methods are further classified into dense and sparse OF algorithms. The dense algorithms calculate the displacement for all the pixels in a frame, with the Horn-Schunck [44] algorithm being the most popular one. On the other hand, the sparse OF methods calculate the displacement only for a specific amount of pixels, such as the Lucas-Kanade [45] technique.

According to [24] the dense methods estimate motion by minimizing a photometric error cost function without knowledge of pixel correspondences, contrarily to the sparse methods that minimize a geometric error. The cost function for appearance-based methods is given by the following equation:

$$T_{k,k-1} = \underset{T}{\operatorname{argmin}} \sum_i \|I_k(u'_i) - I_{k-1}(u_i)\|^2 \tag{2.23}$$

where $T_{k,k-1}$ is the transformation, I_k, I_{k-1} are two subsequent frames and u_i, u'_i are the pixel locations in the corresponding frames (Figure 2.19).

Below the most popular appearance-based methods are described.

- **Dense Tracking And Mapping (DTAM):** DTAM is a dense real-time algorithm for tracking and mapping introduced in [18]. In this method a 3D model of the environment is constructed and utilized for camera tracking. The map is reconstructed by utilizing a bundle of keyframes as the camera moves around a static scene, while the 3D coordinates of all the pixels are estimated by considering space continuity. The camera tracking is achieved by aligning the whole captured images with the 3D dense model, i.e. the images are compared with images produced from the reconstructed map. According to the authors the DTAM algorithm exhibits notable performance under rapid motion variations. Additionally, it is quite robust against occlusions, multi-scale operations and motion blur. However, DTAM is computationally demanding, resulting to the need for parallelization on Graphics Processing Unit (GPU)s for

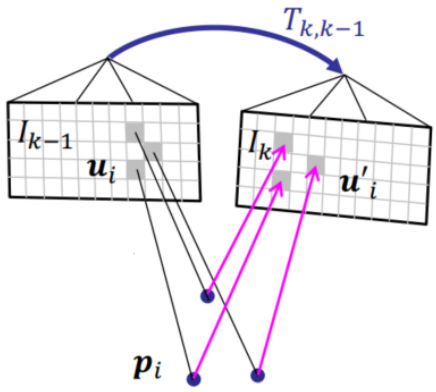


Figure 2.19: Motion estimation using appearance-based methods. [24]

real-time performance. The authors mention that DTAM is highly parallelisable, so implementation on GPUs is feasible. An example of DTAM is illustrated in Figure 2.20e.

- **Large-Scale Direct Monocular SLAM (LSD-SLAM):** LSD-SLAM is another popular monocular dense method described in [46]. The LSD-SLAM is an extension of the semi-dense VO algorithm [47], which reconstructs the environment partially by considering the intensity gradient of pixels. This means that only specific areas that are rich of texture are mapped, because in textureless areas the depth can not be estimated accurately. In the initial phase of mapping, random values of depth are assigned to each pixel and then with photometric consistency these values are optimized. The depth map is approximated by taking into consideration the disparity of each pixel between a captured image and an image that has been captured and stored already. Afterwards, the current approximated values are merged with the already stored inverse depth map with Gaussian merging (like a Kalman filter). In this way the depth map is estimated for every step. The LSD-SLAM extends the semi-dense VO by adding loop-closure detection and pose-graph optimization, in which each node represents an image with an inverse depth map, while the nodes are connected by edges representing the relative scale-aware alignment and information about the covariance. It is worth mentioning that the LSD-SLAM exhibits real-time performance on CPU. An example is depicted in Figure 2.20d.

- **Semi-Direct Visual Odometry (SVO):** SVO is a semi-direct method, for monocular systems, presented in [48]. SVO can be considered as the sparse version of the DTAM and LSD-SLAM, while it operates on pixel intensities. Motion is calculated by considering the same 3D scene projected in successive frames and by minimizing the photometric error of pixels that correspond to this scene and represent the same location. This means that a sparse image alignment approach is used to estimate motion. SVO tracks and triangulates pixels, that are characterized by intense image gradients, by utilizing direct methods. A probabilistic depth estimation for every feature pixel is also used, while feature pixels are extracted only for keyframes. For a higher robustness the algorithm considers small patches around the feature pixels.

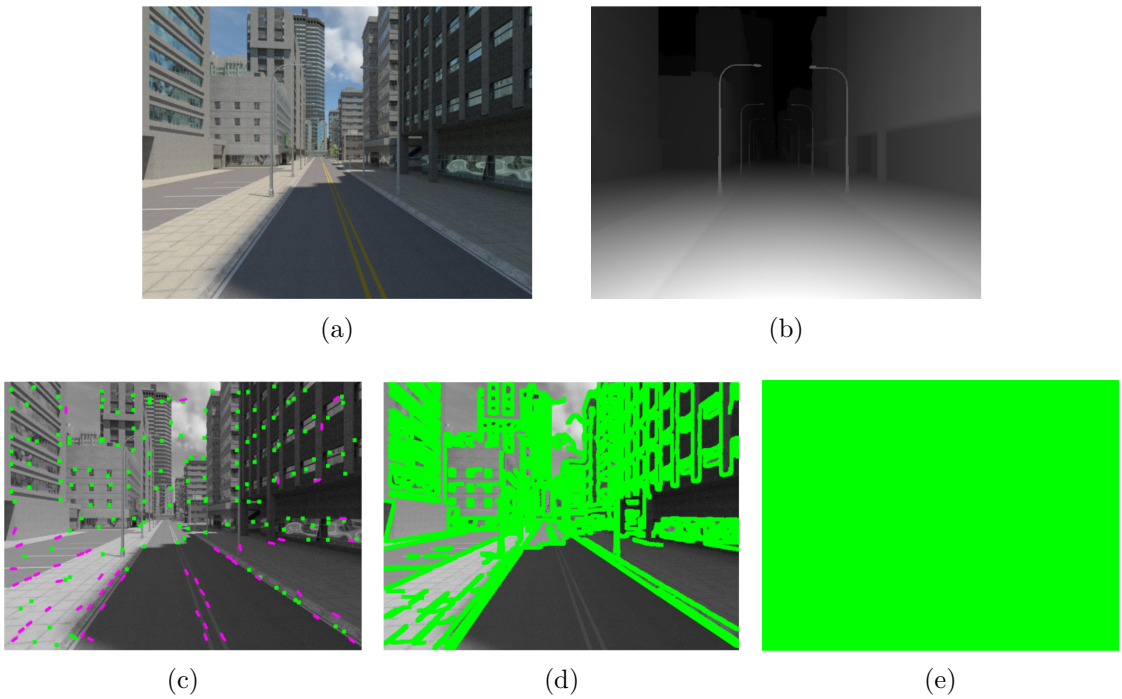


Figure 2.20: (a) Synthetic scene, (b) Depth of the scene, (c) SVO, (d) LSD-SLAM, (e) DTAM. Green pixels represent the pixels that are used by the algorithms for image alignment. Sparse methods (c) use selected pixels at corners or along intensity gradient edges, semi-dense methods (d) use just the pixels with notable intensity gradient, and dense methods (e) use all the pixels in an image. [50]

The authors claim that SVO exhibits notable performance, by achieving 55 frames per second on embedded devices and approximately 300 frames per second on consumer laptops. An example of SVO is illustrated in Figure 2.20c.

- **Direct Sparse Odometry (DSO):** DSO is a recent direct method that also features the advantages of sparse approaches and is presented in [49]. DSO does not use descriptors or detectors and can utilize from the whole image any pixel that is located in regions with intensity gradient, considering also edges or nearly smooth areas like walls. In DSO the acquired frame is split in blocks and then points with high intensity gradient are chosen for reconstruction. In this way, points are distributed across the whole image. Additionally, for a more accurate motion estimation, both photometric and geometric camera calibration are utilized. It is worth mentioning, that DSO takes into account only local geometric consistency, thus it is considered a VO method and not a V-SLAM. Also, it should be noted that DSO is capable of continuously tracking a point only for short distances, as it is not fully invariant to illumination and viewpoint changes.

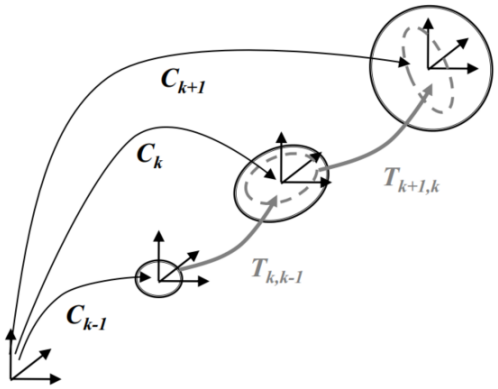


Figure 2.21: The uncertainty in pose C_k depends on the uncertainty of C_{k-1} (black ellipse) and the uncertainty introduced by the transformation $T_{k,k-1}$ (gray dashed ellipse). [51]

2.4.12 Bundle Adjustment

To recover a trajectory in VO subsequent transformations are concatenated. Every transformation $T_{k,k-1}$ introduces an uncertainty and the uncertainty of the agent pose C_k depends on the uncertainties introduced from all the previous transformations. This is depicted in Figure 2.21. To reduce this uncertainty and refine the trajectory there are various methods and a popular one is BA.

BA is an optimization method that is utilized to refine geometric parameters and it is used in the computer vision domain. BA is used to optimize the camera poses and 3D features coordinates that are observed and tracked over multiple frames. This is achieved by minimizing the following cost function:

$$\arg\min_{X^i, C_k} \sum_{i,k} \|p_k^i - g(X^i, C_k)\|^2 \tag{2.24}$$

where X^i denotes the ith 3D feature, p_k^i is its corresponding 2D point in the image at the kth frame and $g(X^i, C_k)$ is its image re-projection according to the camera pose C_k [22]. Practically, a number of 3D features with stable 3D coordinates are observed from different viewpoints (Figure 2.12a), introducing some constraints on where on the 2D images they should be re-projected. In that sense, BA is applied to estimate a mathematical model that predicts the location of 3D points in a set of images, and to refine the traversed trajectory.

If all the past frames are considered, the method is called Global Bundle Adjustment (GBA), however if only a window of a fixed number of n frames is taken into acount, then the method is called Local Bundle Adjustment (LBA). GBA performs a more accurate optimization, because it takes into account all the frames, but it is computationally demanding. In LBA the number of parameters to be optimized is smaller, so it is computationally faster and more suitable for real-time applications.

2.5 Related Work

One of the most recent and popular monocular SLAM approaches is the ORB-SLAM [52]. ORB-SLAM utilizes the ORB features for every task: mapping, loop closing, relocalization and tracking. It is suitable for operating in real-time in large environments and it features a robust and automatic initialization. Additionally, camera relocalization is achieved in real-time with invariance to illumination and viewpoint, it features real-time wide baseline loop closing by optimizing a pose-graph and registers keyframes generously, while is culling the redundant keyframes later. An open-source SLAM system for monocular setup is the ORB-SLAM2 [53] and is based on the monocular ORB-SLAM. ORB-SLAM2 supports also stereo and RGB-D cameras, while it offers a lightweight and robust localization mode for environments that have been already visited. The authors evaluate the performance of ORB-SLAM and ORB-SLAM2 on the KITTI dataset, which is recorded with a car, in contrast to this thesis that evaluates the algorithm in an underwater environment with an underwater vehicle.

In [54] the ORB-SLAM algorithm is evaluated in an underwater context, with various water conditions, like dynamic changes in visibility, lighting and water. The evaluation was conducted with the use of a monocular setup and a captured data-set near Perth of Australia. The data-set includes different places with man-made structures (e.g. pipes, boats) and natural environments (e.g. seabed, reefs). The experimental evaluation was performed in quite shallow waters (1 - 5.5m) and shows that ORB-SLAM performs well under certain conditions. The authors examined the ORB-SLAM algorithm that uses only visual input from a downward looking camera, however this thesis examines the ORB-SLAM algorithm augmented with an IMU and with a different camera orientation. Additionally, the authors give a rough estimation of the performance of the algorithm by presenting the locations were the algorithm was successfully tracking the motion or had lost the track of the vehicle. However, in the datasets that this thesis evaluates the visual-only ORB-SLAM, the performance of the algorithm is assessed in more detail with references to the Absolute Position Error (APE) in relevance to the groundtruth trajectory, while the impact of the algorithm's configuration is also examined.

An open-source[1] bio-inspired solution for underwater monocular SLAM, called DolphinSLAM, is presented in [55]. The proposed algorithm extends the RatSLAM algorithm for 2D ground robots, and utilizes a down-looking camera, an IMU, a DVL and a sonar. For feature detection the Hessian detector is used, while for feature description the SURF descriptor. The authors evaluate the DolphinSLAM, using a dataset with a simulated underwater environment, created with the Underwater Simulator (UWSim), and with a real data-set, which takes place at a marine pier. The authors utilize for motion estimation also a DVL and a sonar, in contrast to this thesis that evaluates a motion system with less sensors, only a camera and an IMU. Additionally, the datasets that are examined feature a downward looking camera, while the feature detector/descriptor utilized is more computationally demanding in contrast to the ones that are evaluated in this thesis.

In a recent study [56] they propose a monocular VO method, called UW-VO (UnderWater-Visual Odometry), which is based on Harris corners, extracted with the

[1] DolphinSLAM source code: `http://goo.gl/zuAsi8`

Shi-Tomasi detector, and tracked with the Lucas-Kanade optical flow technique. The authors claim that the method is robust to short occlusions, by featuring a re-tracking mechanism to locate lost features, and functions well when turbidity is present. In contrast to this thesis that evaluates algorithms that estimate also the scale with IMU fusion, in this paper the motion estimation is up to a scale by using visual-only input.

A method for AUV localization, utilizing visual measurements of structures and artificial landmarks in underwater environments, is presented in [57]. The method is based on particle filtering and on an appearance-based approach. The evaluation was conducted in a tank with an AUV featuring a monocular setup, an IMU, a DVL, a depth sensor and an Attitude and Heading Reference System (AHRS). The results show that the visual system contributes into limiting the drift from the Inertial Navigation System (INS), but in practical applications this method faces some limitations, such as narrowed visibility, due to the huge size of the underwater structures. While in this thesis only a camera and an IMU are considered, the authors of this paper utilize additionally a DVL and an AHRS.

In [58] a framework for underwater ship hull inspection is presented. In the study a monocular V-SLAM algorithm with real-time response is proposed, utilizing the open-source algorithm Incremental Smoothing And Mapping (iSAM) and using for feature detection and description a combination of SIFT and SURF algorithms. In contrast to this thesis the implementation in this paper considers measurements also from a pressure sensor and a DVL.

The authors in [59] propose a SLAM algorithm for underwater structures mapping. The algorithm extends the Visual-Inertial state estimation method OKVIS, and it combines data from a stereo visual setup, an IMU, and a sonar. In [60] the authors augment their aforementioned work by adding depth measurements to execute a robust initialization for depth estimation. In addition, they add in the pipeline a step for enhancing the image quality for better feature extraction and they implement a loop closing method with the use of Bag of Words (BoW). In contrast to this thesis that utilizes only a camera and an IMU for motion estimation, this papers makes use of a stereo camera setup and additionally of a sonar and a pressure sensor.

A low-cost embedded stereo system for underwater missions is presented in [61], along with its hardware and software specifications. A distributed embedded system consisting of three Raspberry Pi 3 is deployed. For feature detection the authors used the Harris-based Shi-Tomasi method [28], while for feature tracking the Pyramidal Lucas-Kanade method. When compared to SIFT, SURF and BRISK, SIFT and SURF were slightly more accurate, while BRISK performed with less accuracy. While this paper evaluates a motion estimation approach on an embedded system for underwater environments, a stereo camera setup is utilized, contrarily to our case that a monocular setup is considered.

Another embedded approach for underwater cases is proposed in [62], in which a monocular camera setup is utilized along with measurements from an IMU and a depth sensor. For feature matching the Shi-Tomasi and Lucas-Kanade pyramidal optical flow methods are used. The motion estimation algorithm is implemented on a Raspberry Pi 2 along with Robot Operating System (ROS). This thesis differs in the utilized embedded system as a processing device and in the camera orientation. The approach in this paper

contains a downward looking camera and a Raspberry Pi, whereas this thesis examines the performance of the lately released NVIDIA Jetson Xavier, utilizing also its embedded GPU.

In [63] another setup with a similar type of camera (FLIR Blackfly) to ours can be found. The authors use the stereo camera setup for a dense 3D map reconstruction and for tracking. In contrast to our case in which a USB 3 camera is utilized, the authors use a stereo setup and a GigE Vision camera, for which the control and communication are done over the Ethernet.

The authors in [64] evaluate which of the available detectors and descriptors are suitable for implementing the computer vision component of a localization algorithm for underwater environment, which utilizes the Extended Kalman Filter (EKF) principals. The following descriptors, detectors and matching methods were examined: SURF features matched with Approximate Nearest Neighbours (ANN), FAST features matched with Zero-mean Normalized Cross-Correlation (ZNCC), SURF features using ZNCC, Shi-Tomasi features with ZNCC, and Center Surround Extremas (CenSurE) features using ZNCC. The experiments were conducted with a down-looking camera and an IMU. The results suggest that for real-time applications the most suitable choices are SURF features with ANN and Shi-Tomasi features with ZNCC, while FAST and CenSurE proved to be inaccurate in cases where motion blur was present. This thesis examines a range of other algorithms for motion estimation in underwater environments that are not included in this paper.

In [38] the author evaluates an open-source algorithm, that uses FAST and KLT for motion estimation, and the ORB-SLAM2 algorithm. Contrarily, this thesis enhances the same open-source algorithm with a wide number of other feature detectors/descriptors, it uses descriptors for motion estimation instead of KLT, and also introduces a technique for more efficient features distribution. Furthermore, in this thesis ORB-SLAM2 is evaluated on a number of different underwater datasets and its performance is compare to the Visual-Inertial ORB-SLAM, assessing the impact of augmenting the algorithm with an IMU. Finally, one of the five underwater datasets of this thesis is the same with the aforementioned work, however the author of this thesis examines it for a different purpose, to assess the impact of image processing filters on this dataset.

An experimental evaluation of some promising open source algorithms for terrestrial, aerial and underwater vehicles, is presented in [65]. The following methods were examined: MonoSLAM, libVISO, Parallel Tracking and Matching (PTAM), ORB-SLAM, SVO, LSD-SLAM and RatSLAM. The underwater data-set used for evaluation, was captured at the coast of Barbados, containing coral reefs and shipwrecks. The ORB-SLAM and PTAM algorithms had the most promising performance regarding the underwater environment, while MonoSLAM, SVO and LSD-SLAM completely failed. A worth mentioning point reported is that the initialization step might greatly affect the performance, because in monocular methods the depth cannot be estimated from a single frame. This thesis examines a range of other algorithms for motion estimation in underwater environments that are not included in this paper. The only common algorithm is the ORB-SLAM, which in our case was evaluated on different underwater datasets and a thorough investigation was conducted for the impact of its configuration parameters.

In [66] some popular and publicly-available monocular VIO pipelines are evaluated

for aerial vehicles, using the EuRoC Micro Aerial Vehicle data-sets [67]. The pipelines: MSCKF[2], OKVIS, ROVIO, VINS-Mono, SVO[3]+MSF[4] and SVO+GTSAM[5], are implemented and tested on the hardware platforms: Intel NUC, UP Board, ODROID and on a laptop. The VINS-Mono exhibits the best accuracy and robustness on all the hardware platforms, especially when loop closure is achieved, but is more demanding on computing resources. A good alternative is ROVIO, with lower accuracy and more efficient usage of resources. While this paper investigates similar algorithms with the ones in this thesis, the evaluation is done on aerial datasets, whereas in our case the performance of the algorithms is assessed on underwater cases. Additionally, in this thesis the performance of the embedded device NVIDIA Xavier Jetson is evaluated, which is not included in this paper. Furthermore, the performance of a different VIO hardware is examined.

In [68] a monocular visual odometry algorithm for underwater vehicles is proposed, in which the scale factor is estimated online, by utilizing a pressure sensor and an IMU. This method is suitable for station-keeping and for precise control, as it drifts only in relevance to the travelled distance and not with time. The method was tested with a down-looking camera, and was proved to be also robust in position estimation, against minor movements by fish or shrimps. This paper utilizes for scale estimation also a pressure sensor and features a downward looking camera. This thesis uses only the IMU for scale estimation, while the camera has a different orientation, facing forward with a slight inclination downwards.

2.6 Conclusion

ROVs are widely used for various applications that take place in subsea. In many of these applications the precise localization and positioning of the underwater vehicle is quite important. While the ROVs can carry a number of different sensors, many of them can not be exploited for efficient motion estimation, due to the costs, complexity, weight and inaccuracy. Taking into account that camera sensors are relevantly cheap, small, light-weight, have low power requirements and most of the robots carry one, VO would be a very promising solution for underwater motion estimation. Not only that, VO has been proved to be more accurate and with less drift compared to other motion estimation methods. The cameras can be an ideal solution for navigation in mobile vehicles where the weight, the size and the power consumption are limiting factors. The most significant asset of VO is that it can be exploited for navigation in GPS denied environments, like isolated indoor and aerial areas, or in subsea. VO techniques can be deployed with various camera setups, like a stereo setup, a monocular, or an omnidirectional camera. Each of them comes with advantages and disadvantages, which have to be considered for the case of every application. Although the monocular setup is the most simple and compact solution, the scale ambiguity issue has to be considered. With just one camera the motion is estimated up to a scale and to compute the absolute scale an extra sensor has to be used, like an IMU. In case an IMU is fused with a camera then

[2] https://github.com/daniilidis-group/msckf_mono
[3] http://rpg.ifi.uzh.ch/svo2.html
[4] https://github.com/ethz-asl/ethzasl_msf
[5] https://bitbucket.org/gtborg/gtsam/

VO and V-SLAM fall into the categories of VIO and VI-SLAM respectively. While VO estimates the motion locally, V-SLAM augments VO with the abilities of loop closure and global map optimization, and at the same time it maintains a global map. Even in the case that building a map is not desired, V-SLAM should still be considered for motion estimation, because it results in more accurate and consistent trajectories. VO and V-SLAM can be implemented with two different approaches, by using feature-based or appearance-based methods. Feature-based methods make use of salient features that are extracted from the images, and are more robust to illumination, scale, rotation, viewpoint changes and to occlusions. On the other hand, the appearance-based methods utilize the intensities of the pixels and exploit more visual information, while they can estimate the motion even in textureless areas like the sand. Despite VO being a promising solution to motion estimation, it comes along with many difficulties for the underwater cases. The subsea environment is harsh and unpredictable, introducing the effects of attenuation, scattering and distortion. The floating particles and the turbidity of the water make it difficult to extract and track features robustly, while a lot of subsea areas lack of features. Due to the nature of the underwater environment it seems that the feature-based methods are more suitable for motion estimation than appearance-based, as they are more invariant to dynamic conditions. Finally, this thesis, compared to other works, evaluates the performance of various VI-SLAM algorithms deployed on a VIO system, featuring recent hardware and an alternative camera orientation, aiming the structured underwater cases.

3 Datasets

In this chapter the utilized data-sets are presented.

3.1 UWSim Dataset

The authors in [69] present a simulated underwater dataset, which is also available online[1]. Producing underwater data for Visual Odometry (VO) is an expensive and demanding procedure. Besides the expensive equipment that is required, there is also the need for a specialized crew, while the working conditions are also challenging. To overcome such difficulties the researchers of this paper utilized the Underwater Simulator (UWSim), an underwater simulator, which is capable of reproducing real underwater conditions from recorded logs.

This dataset is comprised by various trajectories, with some of them featuring different levels of turbidity. The seabed is constructed by mosaics illustrating real corals. In addition, the UWSim dataset simulates the Girona500 Autonomous Underwater Vehicle (AUV) with a number of sensors like Doppler Velocity Log (DVL) and Inertial Measurement Unit (IMU). In our case the IMU data could be useful for a Visual Inertial Odometry (VIO) assessment, however the raw gyroscope and accelerometer data are not provided. Instead only the orientation is provided in every time step in quaternions. It is worth mentioning that this dataset is not conveniently documented, with information about the utilized hardware missing, explicit groundtruth data missing, while there is no explicit information about the coordinate frames neither for the way that they are related.

The UWSim dataset comes in ROS bag format. Five different sequences were utilized in this thesis, the "Second proposed", illustrated in Figure 3.1, which is the biggest trajectory, while the other four are the same trajectory with various levels of turbidity, presented in Figure 3.2. The frames were captured with a downward facing camera in 10 fps with a resolution at 320×240. As it is a simulated dataset there is no distortion. The calibration parameters of the utilized camera are presented in Table 3.1.

[1] `https://goo.gl/GtMQkv`

Table 3.1: UWSim dataset calibration parameters

Focal Length (pixels):	f_x	257.3408	f_y	257.3408
Principal Point (pixels):	c_x	160	c_y	120

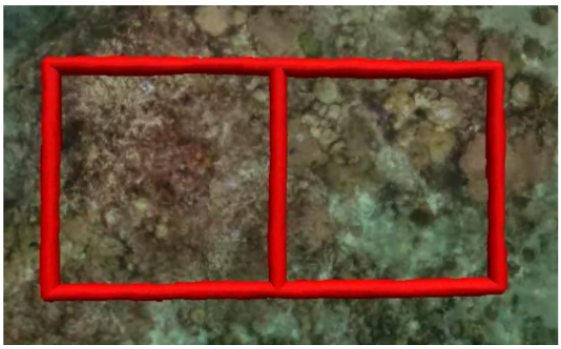
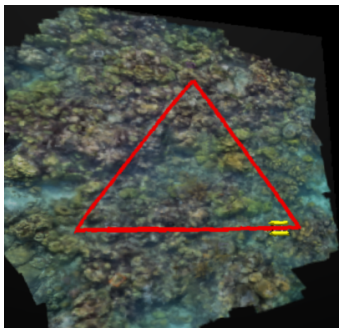
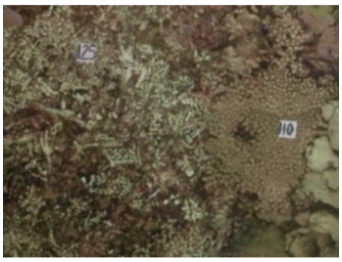


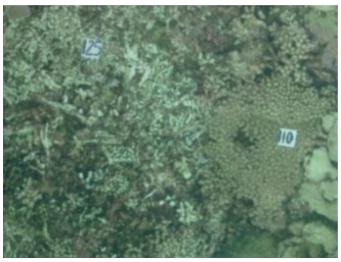
Figure 3.1: "Second proposed" trajectory of the UWSim dataset.



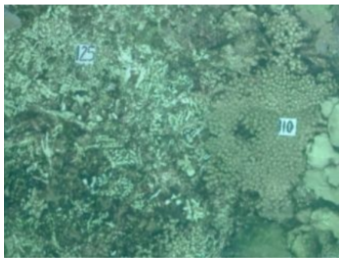
(a)



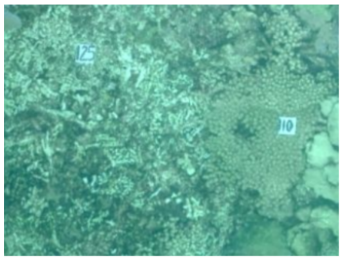
(b)



(c)



(d)



(e)

Figure 3.2: The third UWSim trajectory and its four different levels of turbidity. (a) "Third proposed" trajectory of the UWSim dataset. The names of these UWSim sequences are (b) "third_1" (c) "third_2", (d) "third_3", (e) "third_4".

3.2 Aqualoc Dataset

Aqualoc is a recent and real underwater dataset presented in [70] and published online[2]. It is comprised of seven different sequences acquired in a harbor, and it provides IMU data as well. All the sequences start and finish in the same spot, while each of them provide a trajectory of different length and various underwater conditions. All of the sequences are characterized by plenty of features, however the noise is still present and

[2] `http://www.lirmm.fr/aqualoc/`

Table 3.2: Aqualoc dataset calibration parameters

Focal Length (pixels):	f_x	413.325953665660	f_y	413.701987394836
Principal Point (pixels):	c_x	305.95074832849	c_y	259.44399489463
Distortion coefficients:	k_1	-0.061255682971369	k_2	-0.0037967433951352
	p_1	0.0273266347712045	p_2	-0.0302964031428870

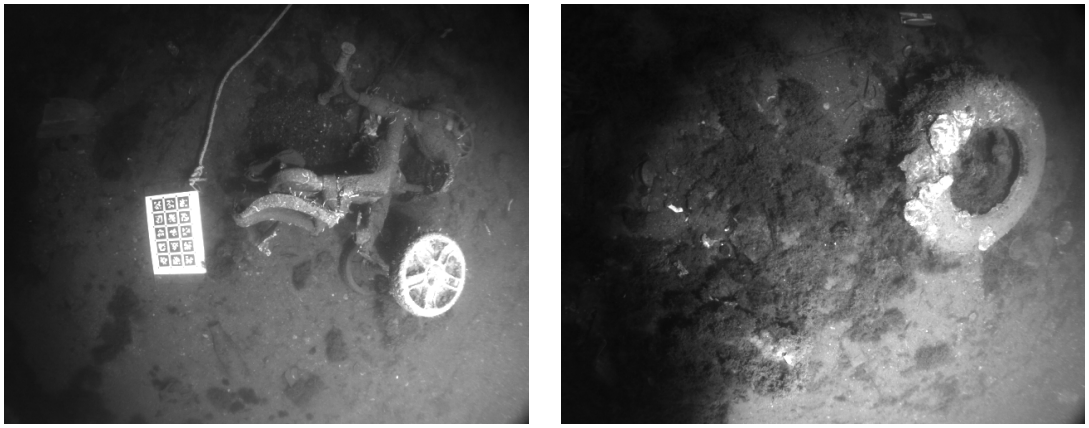
in some of them very intense. In some of the footage the back-scattering effect and floating particles are very obvious, while others feature low visibility parts. During the acquisition the Remotely Operated underwater Vehicle (ROV) oscillates intensively in many parts and it executes rapid rotations, making the dataset a challenging test.

The acquisition was done with a downward facing camera at 20 Hz and with a resolution of 640×512 pixels. Additionally to the camera data, IMU data are recorded at 200 Hz, with a MEMS - MPU-9250. Despite only the two aforementioned sources of information are utilized in this thesis it has to mentioned that the dataset provides compass and pressure measurements as well. The dataset is timestamped and it comes in two formats, in raw data with png images and csv files, and in a ROS bag. Although the frames and the IMU measurements are properly timestamped, the groundtruth data are not, but they correspond to specific camera frames, so they can be related to the timestamps of the frames. The dataset is properly documented, providing all the required information, including the hand-eye calibration between the camera and the IMU, presented in the matrix 3.1. The pinhole camera model is used with the equidistant distortion model. The camera calibration parameters that were utilized are mentioned in Table 3.2. Two representative frames of the dataset are shown at Figure 3.3.

$$\begin{bmatrix} -0.99978035 & 0.0169654 & 0.01230552 & -0.01719238 \\ 0.01210101 & -0.01210461 & 0.99985351 & 0.14944769 \\ 0.01711187 & 0.9997828 & 0.01189665 & -0.01915984 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{3.1}$$

3.3 Underwater Caves Dataset

The authors in [71] present a real underwater dataset taking place in an underwater cave complex. The dataset comes along with a number of various sensors, among them a camera and two IMUs. The acquired data are provided in ROS bag format and in raw files (png and txt) as well. The camera footage in some areas is very challenging and has bad visibility and extensive noise, as it can be seen in Figure 3.4. The images were captured with a downward looking camera with a resolution of 384×288 at 4 Hz. The camera distortion model is the Radial-Tangential, also known as "plumb bob" and the camera calibration parameters that were used are presented in Table 3.3.



(a) (b)

Figure 3.3: Representative frames of the Aqualoc dataset.

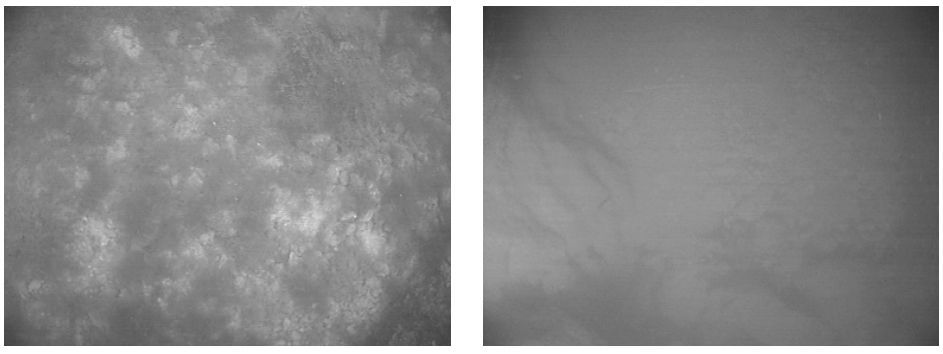
Table 3.3: Underwater caves dataset calibration parameters

Focal Length (pixels):	f_x	405.6384738851233	f_y	405.588335378204
Principal Point (pixels):	c_x	189.9054317917407	c_y	139.9149578253755
Distortion coefficients:	k_1	-0.3670656233416921	k_2	0.203001968694465
	p_1	0.003336917744124004	p_2	-0.000487426354679637

3.4 Archaeological Underwater Dataset

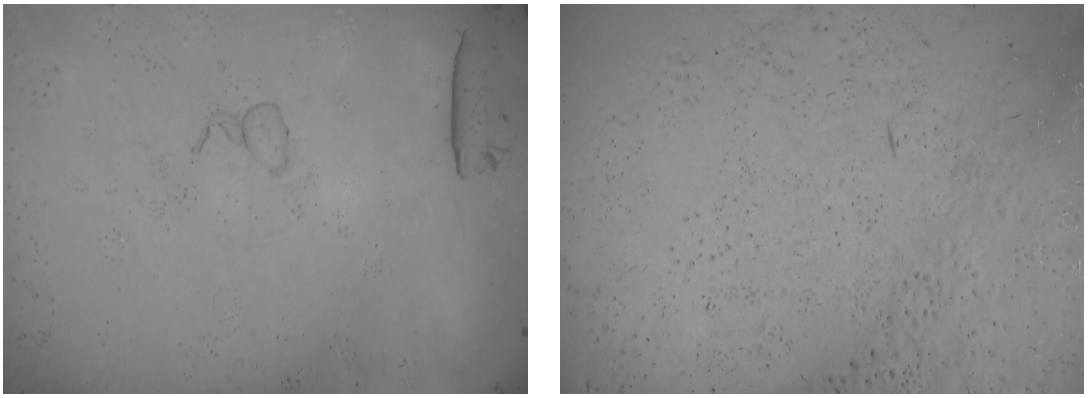
In [56] a real underwater dataset is presented, which is consisted of five sequences recorded at 500 m deep in the Mediterranean Sea (Corsica), and is available also online[3]. The dataset was captured at an underwater archaeological area, where parts of it feature buried objects in the sand providing salient features, while others are comprised only by the seabed sand. The footage offers very good quality images in most of the sequences

[3] `https://seafile.lirmm.fr/d/aa84057dc29a4af8ae4a/`



(a) (b)

Figure 3.4: Representative frames of the Archaeological dataset.



(a) (b)

Figure 3.5: Representative frames of the Archaeological dataset.

Table 3.4: Archaeological dataset calibration parameters

Focal Length (pixels):	f_x	243.63	f_y	327.26
Principal Point (pixels):	c_x	404.45	c_y	203.98
Distortion coefficients:	k_1	-0.080067796952468	k_2	0.0083194202396929
	p_1	0.0069641692677948	p_2	-0.002674189344796

and the motions are quite smooth. There is a variety in the level of turbidity and in some cases fish are present during the recording. The footage was captured with a downward looking camera at 16 fps with a resolution at 640×480, and is packed in a ROS bag file. Additionally, the dataset comes along with the groundtruth data. Also in this dataset the groundtruth are not properly timestamped, but each of the provided groundtruth pose corresponds to one frame, thus they can be related with the corresponding timestamps. Two representative images of the sequences are illustrated at Figure 3.5, while the camera calibration parameters are presented at Table 3.4.

3.5 Fugro Datasets

A notable number of datasets were recorded in Fugro premises with the deployed VIO system. Due to space limitation here only two are presented, but they are quite representative and illustrate the conditions under which the other datasets were captured as well. Initially, a dataset was recorded on air inside an office. This dataset includes a moderate number of features, as a white wall is part of the recorded scene. To simulate more challenging conditions, the exposure was set to a low value, which resulted into darker images, which is depicted in Figure 3.6a. On top of that, during the data acquisition a person passes in front of the camera, causing noise to the motion estimation algorithm.

The second dataset was captured underwater in an indoor tank. To simulate an underwater environment like the one that this thesis examines, there was the need to set up some structures. Some piles and starfish lie on the bottom of the pool, while a pillar occupies a notable area in the vertical axis, simulating common underwater man-made

Table 3.5: Fugro Underwater Dataset Calibration Parameters

Focal Length (pixels):	f_x	1003.36622688252	f_y	1001.99354884491
Principal Point (pixels):	c_x	374.08669313874	c_y	327.90535821388
Distortion coefficients:	k_1	0.13581294847844	k_2	0.559611440193901
	p_1	0.00145237954991137	p_2	-0.00421067421140184

Table 3.6: Fugro Air Dataset Calibration Parameters

Focal Length (pixels):	f_x	751.67376410316	f_y	751.35793206535
Principal Point (pixels):	c_x	382.43974949583	c_y	323.76831384354
Distortion coefficients:	k_1	-0.122540840350693	k_2	0.165539808334854
	p_1	-0.00015876742148404	p_2	0.00010787476357992

installations. Some piles were covered with yellow tape to create more features, and they were placed in a vertical position. In the real field, huge pillars occupy a significant amount of space in the vertical axis, while the seabed is rich of features and usually pipes lie on it. In this artificial environment it was a bit challenging to simulate the sea's turbidity, but there are less features in contrast to the real field, and the features are not evenly distributed in the acquired images, as there are regions that are occupied mostly from the black tank wall.

In both of the presented images in Figure 3.6 there are some blurred regions. These regions are occupied by proprietary Fugro patterns, which feature different sizes of black dots. They are blurred, for confidentiality reasons. The patterns were placed in the scene to create some additional features, but most importantly they are utilized to estimate the groundtruth trajectory of the robot. Both of the datasets were captured with an RGB machine vision camera at 1224×1024. To reduce the original huge resolution of the camera (4K), binning in camera's hardware was utilized, while the resolution of the captured images was reduced even more with OpenCV afterwards. The final utilized resolution is the 766×640, because it was more convenient for reasons described in the results chapter. The camera was placed in a forward looking orientation with a slight downward pitch. The camera frames were captured at 9 FPS. Furthermore, both of the datasets feature IMU measurements, which were acquired at 100 Hz. The camera frames and the IMU data are timestamped and packed in a ROS bag file. The hand-eye calibration between the camera and the IMU is presented in matrix 3.2, while the intrinsic calibration on air and in water are mentioned in the Tables 3.6 and 3.5 correspondingly.

$$\begin{bmatrix} -0.01261917 & -0.01290516 & 0.99983709 & 0.05242 \\ 0.99929868 & -0.03541738 & 0.01215523 & 0.0009 \\ 0.03525475 & 0.99928928 & 0.01334304 & 0.03845 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{3.2}$$

(a) (b)

Figure 3.6: Representative frames of (a) the dataset acquired on air at an office, (b) the dataset acquired in water in a pool.

3.6 Conclusion

There is a significant quantity of datasets for the ground and air domains. Many of these datasets are well documented and have been evaluated by a notable number of researchers, verifying their quality. On the other hand, for the underwater domain the amount of datasets and their quality are limited, while they are evaluated by a small amount of researchers. In addition, in contrast to the air and ground datasets, there are few to none examples, results and discussions on the underwater datasets online (excluding papers). Finally, most of the underwater datasets with camera and IMU measurements, feature a downward looking camera, pointing to the seabed. In the context of this research no underwater dataset was found online, featuring a forward looking camera and taking place in a subsea area with man-made installations. The aforementioned issues make it difficult to evaluate VIO and Visual Inertial SLAM (VI-SLAM) algorithms on diverse underwater cases.

Experimental Setup & Equipment

4

In this chapter the experimental setup and the rationale for choosing the components of the utilized Visual Inertial Odometry (VIO) system are described and justified. Moreover, additional equipment that was used is presented.

4.1 Visual Odometry (VO) Experimental Setup Selection

4.1.1 Monocular vs Stereo Setup

VO approaches are classified into two major categories, monocular and stereo setup. Stereo setup comes with the advantage of providing in every frame the absolute scale and depth of the scene, by triangulation. This is the main reason that many VO implementations are based on stereo setup, because having a known absolute scale for every frame saves a lot of extra effort for computing it with other means. However, the suitability of each setup depends on the desired goals and on the environment that the robot operates. Despite stereo setup being the most popular and straight forward solution, solving many problems, for the examined case in this thesis the monocular setup has been selected for a notable amount of reasons that are explained below.

The case that is studied in this thesis regards a Remotely Operated underwater Vehicle (ROV) aimed to operate in underwater environments, that are occupied by man-made structures. Such kind of scenes are very common in Oil & Gas industry, in which structures are widely deployed subsea. In contrast to the onshore domains, underwater missions are risky and challenging, due to the harsh environment of water and high pressures. The underwater equipment is notably expensive for a reason. Every piece of hardware that is utilized must be sealed to avoid getting destroyed by water leakage and must be armored in housings of special design and specific materials, that are capable of withstanding high values of pressure. It is obvious that the construction of a ROV needs more sophisticated and detailed design, compared to the ground and air robots. In the case of monocular setup the sealing becomes easier, as it regards only one camera instead of two, while the amount of material and effort for housing design and the costs are reduced in half. In addition, decreasing sealing and housing leads to smaller sizes and more compact ROVs. While mass reduction is desired, even more important is that a monocular setup results in less electronics, less connections and less mounted objects on a ROV. Already it is evident that the risk of failure, regarding the aforementioned domains, is limited, with even more justifications to follow. A stereo setup requires accurate synchronization between the cameras and power supply to support both of them. Moreover with two cameras there is demand for double the bandwidth, processing power, memory, and data that have to be handled as input in real-time. Another worth mentioning point is that more effort is required to calibrate a stereo system and to maintain

the baseline between the two cameras. From another point of view, the commercial, monocular setup is again the winner, as a monocular VO implementation is portable to most of the ROV, due to the reason that most of them feature and are able to handle at least one camera.

Taking into account that the underwater missions are very critical, expensive and need accurate planning and organization, the demand for redundancy and robustness becomes obvious. Lets assume that after investigation, the monocular case is not suitable and a stereo setup has to be deployed. Having examined and developed a monocular approach, in case of failure of the stereo setup, the system can immediately alter to the monocular approach to accomplish the mission, instead of aborting it, resulting in negative consequences in costs and time. The final reason that a monocular approach was selected to be studied in this thesis, is that that the considered ROVs have an advantage compared to the drones or other ground robots. More precisely, many of the air and ground robots utilize the stereo setup, because they do not feature powerful embedded systems that can handle more complex and effective algorithms, and they use medium to low precision Inertial Measurement Unit (IMU)s. On the other hand, in the examined case a powerful embedded system from NVIDIA is utilized, along with a high accuracy IMU. Thus, it was attractive to examine what will be the result of a monocular VIO algorithm executed on this hardware, and if eventually is possible to achieve adequate accuracy and performance with it, by exploiting at the same time all the aforementioned advantages of the monocular approach.

4.1.2 Camera Field of View

With the choice of monocular setup the next question would be why not an omnidirectional (360°) camera? It is true that omnidirectional cameras have some advantages. A 360° camera might offer better accuracy in rotation and translation estimation, because it can keep in track features for a longer period of time, as it can retain them in its Field of View (FOV) while they are in front of it and at the back of it as well. Moreover, omnidirectional cameras are more robust near homogeneous areas with small amount of features. The reason is that in case the robot approaches a feature-less plane, it can still capture features from the rest FOV, which might include the area behind it as well. However, omnidirectional cameras come also with some drawbacks. The calibration of a 360° camera is more challenging and difficult, compared to wide FOV lenses, especially when they are aiming the underwater environment. In addition, the sealing and housing is more challenging and costly when it has to be implemented for an omnidirectional camera. Another point to consider is that not all the VO implementations support a 360° camera model, while in case of using one, more complexity is added on the VO algorithm. A final and important concern is how much the advantages of such a camera can be exploited underwater. In the examined case of this thesis, the ROV will operate near a man-made structures environment, in which probably most of the structures will be in front of it, while in subsea the amount of features in the surroundings is significantly less, compared to onshore domains.

Nevertheless, it is still possible that an omnidirectional camera would be of worth to try it underwater for VO. In this case the type of the omnidirectional camera and

Figure 4.1: Light reflection causing loss in FOV. The small circles denote features that can be still extracted from the rest of the image. [72]

its drawbacks have to be considered as well, hence a rough assessment and proposal is provided below. The **Rotating Cameras** do not acquire a panoramic view in a single shot, so they introduce motion blur and distortion, setting them unsuitable for real-time applications. On top of that, they have moving parts, which they increase the power consumption and the risk of mechanical failure. The **Catadioptric Cameras** do not have a uniform resolution variation across the acquired images, due to the use of mirrors. Furthermore, as a single camera is used to point to the special mirror, the resulting resolution of the 360° image will be degraded. The most promising choice seems to be the case of **Multiple Cameras**. This setup is quite expensive, not very compact and stitching of images might be computationally expensive. However, if housing - sealing can be resolved and the demand for computational power can be faced with optimizations, the Multiple Cameras scenario might be a good option.

While an omnidirectional camera was not selected for this project, a wide angle lens will be used. A wide angle lens offers some of the advantages of the omnidirectional cameras, while keeping the complexity in a lower level. Features in significantly different directions are still visible from the camera and can be tracked for a longer period of time. Additionally, the extracted features are better distributed, and not gathered in the same area, providing the ability to estimate even small rotations and translations more accurately. Finally, it is possible that the ROV will operate in the same area along with others. In such a case the lights of one ROV might reflect on others' cameras, causing loss in FOV. Figure 4.1 illustrates a case were a source of light (sun in this case) causes loss of sight. If we consider this light source to be the light of a ROV, as it is observed from this figure, other features can be still captured in the rest areas of the acquired image.

4.1.3 Machine Vision vs Consumer Cameras

For this project a machine vision camera from FLIR was selected. Here the reasons why it was chosen over traditional consumer cameras are presented. Machine vision cameras

are agile and offer a range of options, which are mentioned below:

- High resolution and high frame rates.
- Custom resolution and frame rates. Consumer cameras offer standard resolutions and specific values of Frames Per Second (FPS).
- Robust design to withstand industrial and extreme environmental conditions, such as high pressure, high temperature and water.
- Reliability. Machine vision cameras usually come with longer warranty and have longer life span.
- Hardware and firmware control. Machine vision cameras offer the chance to trigger them via software or hardware, and provide options such as setting binning and exposure time. Additionally, they come with pins that can be used to interface them directly with other hardware.
- Precise data tracking and timing. Timestamping and precise acquisition time is very important in applications such as VIO.
- Capable of operating at very low or very high amounts of light.
- Featuring high speed connections and interfaces, such as GigE Vision over Ethernet and USB 3.
- Detailed technical support and documentation.
- SDK for building proprietary software and adapt them accordingly for every application. [73]

4.2 Experimental Equipment

The machine vision camera that was utilized for the experiments is from *FLIR* (formerly Point Grey), and it is an RGB camera. More precisely the model is the *Blackfly BFS-U3-51S5C*, featuring a USB 3.0 interface, a global shutter Sony sensor, and a number of General-Purpose Input/Output (GPIO) pins. The camera is depicted in Figure 4.2 and in Table 4.1 a brief summary of important specifications is presented. More detailed information about this model can be found online[1].

For inertial navigation measurements an IMU from *SBG Systems* was utilized, with an output rate of 200 Hz. More precisely the model that was used is the *Ellipse2-A* and is illustrated in Figure 4.3. The IMU provides 3 axis gyroscopes, accelerometers, magnetometers, and temperature sensors. The IMU parameters, regarding the noise and random walk after the calibration, are presented in Table 4.2. Further information about this IMU can be found online[2].

[1] `http://softwareservices.ptgrey.com/BFS-U3-51S5/latest/Model/spec.html`
[2] `https://www.sbg-systems.com/wp-content/uploads/Ellipse_Series_Leaflet.pdf`

Figure 4.2: Blackfly S (BFS-U3-51S5C-C) machine vision camera from FLIR.

Table 4.1: FLIR Blackfly S machine vision camera specifications.

Resolution	2448 × 2048	**Sensor**	Sony IMX250, CMOS, 2/3”
Frame Rate	75 FPS	**Readout Method**	Global Shutter
Megapixels	5.0	**Interface**	USB 3.0
Power Consumption	3 W max	**Power Requirements**	8 - 24 V via GPIO or 5 V via USB 3.0 interface

For synchronization and timestamping of camera frames and IMU data a micro-controller was employed. The micro-controller utilized is the *CY8C5868AXI-LP032* from *Cypress Semiconductor*. Some important specifications are presented in Table 4.3, while further details can be found online[3].

Two embedded systems from *NVIDIA* were utilized for different purposes. *NVIDIA* has released some popular embedded systems aiming at robotics. These embedded systems are compact, but powerful as well, while they feature a GPU and support CUDA for accelerated programming. The first *NVIDIA* system that was utilized is the Jetson TX2, with some of its important specifications mentioned in Table 4.4. The Jetson TX2 was employed for data acquisition, interfaced with a carrier board, a micro-controller, the machine vision camera, and the IMU. The Jetson TX2 was utilized as a module along with a carrier board and a custom PCB, to interface the peripherals. The carrier board used is the *Connect Tech Elroy Carrier for NVIDIA Jetson TX2/TX2i/TX1*. To power the *NVIDIA* module along with the peripherals, 24 V DC was provided, and it was operated at 15 Watts. The second *NVIDIA* platform that was utilized is the Jetson

[3] `https://www.cypress.com/part/cy8c5868axi-lp032`

Table 4.2: SBG IMU Calibration Parameters

Parameter	Values	Symbol	Units
Gyroscope White Noise:	0.000044	σ_g	$\frac{rad}{s}\frac{1}{\sqrt{Hz}}$
Accelerometer White Noise:	0.00056	σ_a	$\frac{m}{s^2}\frac{1}{\sqrt{Hz}}$
Gyroscope Random Walk:	0.000000001	σ_{bg}	$\frac{rad}{s^2}\frac{1}{\sqrt{Hz}}$
Accelerometer Random Walk:	0.00000498	σ_{ba}	$\frac{m}{s^3}\frac{1}{\sqrt{Hz}}$

Figure 4.3: IMU Ellipse2-A from SBG Systems.

Table 4.3: Micro-controller specifications.

CPU Core	Flash (KB)	SRAM (KB)
ARM Cortex-M3	256	64

Xavier. The Jetson Xavier is the most recent and powerful embedded system of *NVIDIA* and it was used in the form of developer kit. The Xavier is the main embedded system that this thesis aimed to evaluate its performance, as it is the more recent from *NVIDIA*. While the Xavier was used to implement the VIO algorithm and conduct experiments offline, it was not employed for data acquisition at the moment of writing this thesis, because it was recently acquired and it was necessary to be interfaced with a carrier board, which was not available. Some important specifications about the Jetson Xavier are presented in Table 4.5.

The Jetson TX2 along with its carrier board was interfaced with the machine vision camera and the IMU in a prototype setup illustrated in Figure 4.4. For the air experiments the Jetson TX2 was outside the depicted housing. After exhaustive testing and evaluation, the whole system, including the camera, the IMU and the Jetson TX2 with the carrier board, was placed in a sealed housing and mounted on the ROV. The ROV that was utilized to deploy the VIO system and to conduct experiments is the BlueROV2 from *BlueRobotics* and is depicted in Figure 5.28b.

4.3 Additional Equipment

The performance of the algorithms along with the utilized datasets were first examined on two laptops. During the period of time that experiments were conducted, the author

Table 4.4: NVIDIA Jetson TX2 specifications

CPU	ARM Cortex-A57 (quad-core) @ 2GHz NVIDIA Denver2 (dual-core) @ 2GHz
GPU	256-core Pascal @ 1300MHz
Memory	8GB 128-bit LPDDR4 @ 1866Mhz | 59.7 GB/s
Storage	32GB eMMC 5.1
Power	7.5W / 15W
OS	Linux Ubuntu 16.04

Table 4.5: NVIDIA Jetson Xavier specifications

CPU	8-core NVIDIA Carmel 64-bit ARMv8.2 @ 2265MHz
GPU	512-core NVIDIA Volta @ 1377MHz with 64 TensorCores
Memory	16GB 256-bit LPDDR4x @ 2133MHz | 137GB/s
Storage	32GB eMMC 5.1
Power	10W / 15W / 30W profiles, 9.0V-20VDC input
OS	Linux Ubuntu 18.04

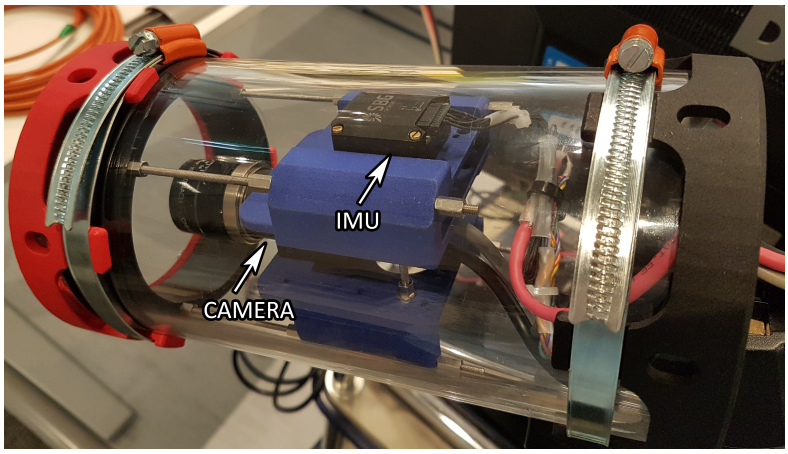


Figure 4.4: Prototype including the machine vision camera and the IMU.

of this thesis had to replace the first utilized laptop. This is the reason why the initial experiments were conducted with a different laptop. This has no affect on the quality or the methodology of this thesis, as the initial results, regarding the hardware performance, are just indicative. On top of that the first laptop was used to execute experiments on different datasets. Thus for every examined dataset, that is a different case, consistency is maintained. From now on and for the rest of this thesis when the equipment "laptop 1" is mentioned, it will refer to the specifications mentioned in the Table 4.6, and when the "laptop 2" is mentioned it will refer to the specifications it Table 4.7. Here it has to be mentioned that everything was conducted using Linux in a virtual environment, leading to a significantly degraded performance and low utilization of the actual laptop hardware.

Table 4.6: Dell laptop 1 specifications

CPU	Intel i7-4800MQ @ 2.70GHz, 4 cores, 8 logical processors
RAM	8GB DDR3 SDRAM, 800.0 MHz
Drive	Samsung SSD 840 EVO
GPU #1	AMD Radeon HD 8790M, 2GB
GPU #2	Intel HD Graphics 4600, 1GB
OS	Virtual Machine - Linux Ubuntu 18.04, 4GB RAM

Table 4.7: Dell laptop 2 specifications

CPU	Intel i7-8850H @ 2.6 GHz 6 cores, 12 logical processors
RAM	32 GB DDR4 SDRAM, 1333.3 MHz
Drive	PC401 NVMe SK hynix
GPU #1	NVIDIA Quadro P2000, 4GB
GPU #2	Intel UHD Graphics 630, 1GB
OS	Virtual Machine - Linux Ubuntu 18.04, 14GB RAM

4.4 Conclusion

VO systems are classified according to their camera setup to monocular or stereo. Each of the cases follow different approaches and methodologies. For the examined case of structured underwater environments the monocular setup was selected as the more suitable. The monocular approach was chosen because among others offers mass reduction, less electronics, less connections, less power consumption, less mounted objects on the ROV, requires easier and cheaper housing, and the approach is more portable to various ROVs. The camera features a wide angle lens, providing more observable features in the FOV for a longest period of time. This has as a result a better distribution of features, which leads to more accurate rotations estimation. Additionally, in case of partial occlusions the wide FOV can still provide an adequate number of features to track. A machine vision camera was selected over a consumer one, because among others offers high and custom resolutions and FPS, robust design to withstand extreme conditions, reliability, technical support and extensive documentation, low level hardware and firmware control, precise data tracking and timing and it is programmable. The monocular VO approach was augmented with an IMU that provides 3 axis accelerations and velocities. For the camera and IMU synchronization a micro-controller was also employed. Finally, one of the most recent and powerful embedded systems of NVIDIA, the Xavier Jetson, with a high performance Central Processing Unit (CPU) and Graphics Processing Unit (GPU) is utilized to complete the proposed VIO system.

VIO System Implementation & Results

5

In this chapter the complete Visual Inertial Odometry (VIO) system implementation is discussed and the results from the experiments are analyzed. More specifically, the performance of five selected algorithms is evaluated on various underwater datasets and on the in-house datasets along with the deployed VIO system. A detailed description of the hardware architecture and functionality is presented. Moreover, the most suitable of the five examined algorithms is deployed on the VIO hardware and its performance on motion estimation is evaluated. Finally, the performance of the algorithm's accelerated version is examined on the embedded system, while results about the embedded device capabilities are presented.

Some of the graphs in the results have been produced by using and adapting parts of the evo package for evaluation of Visual Odometry (VO) and Visual Simultaneous Localization And Mapping (V-SLAM) [74]. At this point it is worth mentioning that one of the challenges while evaluating VO methods is the variety of reference systems that are involved in each implementation and dataset. Since every algorithm and dataset has its own reference system it is impossible in some cases to directly compare the estimated trajectory with the groundtruth. The estimated trajectory might have the same shape with the groundtruth, however the trajectories are not aligned. To overcome this issue, the evo package uses some common approaches for scale and rotation/translation alignment. For the scale the Sim(3) alignment method is used, while for the rotation/translation the SE(3). It is possible to use each of the alignments separately or in a combination. The scale alignment is usually significant for the vision-only monocular VO approaches, in which the absolute scale is unknown and the trajectory is estimated up to a relative scale.

At this point it has to be noted that initially it was attempted to implement a VO algorithm in C++ and OpenCV, based on some open-source approaches. The algorithm was implemented in a way, that various feature detectors and descriptors can be used, while optimizations for a better performance were implemented. With the deployed algorithm some initial results and conclusions were derived for the performance of various feature descriptors/detectors on a ground dataset. However, the algorithm did not produce usable results for the underwater cases. For this reason, other more advanced VIO / Visual Inertial SLAM (VI-SLAM) algorithms are examined. The algorithm and the results of the experiments are presented in the Appendix A.

After researching the advances on VO it was decided to follow an approach in which a camera and an Inertial Measurement Unit (IMU) are fused, so that the absolute scale could be computed as well. Although working with just a camera was feasible and many proposed algorithms and studies utilize only visual inputs, adding an IMU would lead to a more complex, however complete real-time motion estimation system. Five of the most state-of-the-art VO / VIO / V-SLAM / VI-SLAM algorithms were selected

to conduct experiments and to investigate their performance for a monocular setup. These algorithms were chosen, because they are among the most popular in the research community and their performance on the aerial and ground domain has shown promising results. The selected open-source implementations are:

- ORB-SLAM2
- Visual-Inertial ORB-SLAM
- OKVIS (Open Keyframe-based Visual-Inertial SLAM)
- ROVIO (Robust Visual Inertial Odometry)
- VINS-Fusion

The algorithms can be deployed in two different forms. One form uses raw files (png, txt) as inputs, while the other one utilizes the Robot Operating System (ROS) and its associated files. All the implementations were utilized in their ROS version. Here it has to be mentioned that in contrast to the rest of the examined algorithms, ORB-SLAM2 does not make use of inertial data and estimates the traversed trajectory up to a relative scale only by images. However, ORB-SLAM2 is evaluated, as it is one of the most popular V-SLAM algorithms, while in this thesis we assess and compare its performance against the Visual-Inertial ORB-SLAM that combines also inertial data. In this way we can evaluate the impact of augmenting the algorithm with an IMU and inertial measurements. Moreover, it has to be pointed out that as the trajectory is calculated up to a scale, the estimated trajectory is compared to the groundtruth by aligning its scale with the groundtruth one, for all the datasets. Eventually, the two mainly utilized algorithms are ORB-SLAM2 and Visual-Inertial ORB-SLAM. The rest of the three implementations were deployed and examined, but it was not possible to produce usable results for the examined underwater cases, for reasons that are explained in the next sections. For this reason the ORB-SLAM2 and Visual-Inertial ORB-SLAM are described below in more detail.

5.1 Motion Estimation Algorithms

ORB-SLAM2

ORB-SLAM2 is one of the most popular feature based V-SLAM approaches. The algorithm incorporates three parallel threads: tracking, mapping and loop closing. The ORB features are utilized for all the threads, because they can be quickly extracted and they are robust against viewpoint, scale, illumination and rotation changes.

The algorithm begins with map initialization, during which the relative pose between two frames is computed and an initial set of map points are triangulated. As the authors propose, for map initialization two geometrical models are computed in parallel, a homography for planar scenes and a fundamental matrix for non-planar ones. To compute the homography 4 points are used, wheres for the fundamental matrix 8 points. Based on a heuristic and a relative score, the best of the the two models is selected to recover the

relative pose. The proposed method detects low-parallax cases and avoids initializing a corrupted map. With the selected model various motion hypotheses are estimated and if one is significantly better from the others, exhibiting high parallax and low reprojection error, a full Bundle Adjustment (BA) is done, otherwise the initialization process starts over.

The tracking thread is responsible to localize the camera in every frame and to decide whether a new keyframe will be registered. Since a lot of incoming frames are processed by the algorithm, many of them have overlapping information and for this reason they are not inserted into the map. This algorithm registers a frame whenever it provides a significant amount of new information, and the frame is called keyframe. The algorithm uses a strategy called "survival of the fittest", which means that new keyframes are frequently inserted, according to some criteria, to make the algorithm robust against challenging scenarios. This means that initially many frames are used in case there is notable noise in the scene or rapid motions, so that the motion estimation can be achieved nevertheless and the track is not lost. However, later these frames might be discarded if they are proved to be redundant. The tracking thread is the most critical one in terms of timing, and it has to operate in real-time to keep track of the motion. If the camera track is lost then the place recognition module takes over and tries to re-localize the camera within the maintained map. ORB features are extracted and correspondences between frames are computed. By the time features are matched between the frames, matches between the current frame and the local map are searched through re-projection and the camera pose is optimized by utilizing them.

The mapping thread is responsible for creating and maintaining a global map. By the time a keyframe is registered, it is integrated into the covisibility graph and the spanning tree is updated by linking the new keyframe with another keyframe, having the most points in common. Moreover, a Bag of Words (BoW) representation for the new keyframe is formed. New map points are registered by triangulating unmatched ORB features from the new keyframe with points in connected keyframes that belong in the covisibility graph. The new points are maintained in the map if they pass some tests, like being observable in more than 25% of the frames that are supposed to be visible and to be at least by three keyframes observable. After keyframes and keypoints registration, in order to avoid extra cost and to retain a compact map that will not grow uncontrollable, the local mapping thread detects the redundant keyframes and the non trackable points and removes them.

The last parallel thread is the loop closing, which is responsible for detecting places that have already been visited before. For every new keyframe the algorithm compares the BoW vector representation of the new keyframe and of its neighbors from the covisibility graph. If the similarity between these vectors is high enough the loop is closed, the two sides of the loop are aligned and duplicated points are merged. Additionally, a similarity transformation is computed, which informs for the accumulated drift in the loop. After the loop closure a pose graph optimization is conducted to maintain global consistency, and the accumulated drift along the trajectory is corrected like a "domino effect". More details can be found in [52], while an implementation of the ORB-SLAM2

is available online[1].

The implementation of ORB-SLAM2 offers various parameters for configuration. The parameter "number of features" indicates the upper limit of the amount of features that can be detected and tracked. The "number of levels" represents the amount of the Gaussian Pyramid levels that are considered to extract features. The parameter "scale factor" determines how much the size of an image is reduced between the Gaussian Pyramid levels. Finally, the parameters "FAST thresholds" indicate the minimum response required to extract a feature. The biggest value determines the initial threshold and if no features are detected then the lower threshold is used.

Visual-Inertial ORB-SLAM

Visual-Inertial ORB-SLAM is a tightly-coupled nonlinear optimization method that is based on ORB-SLAM and augments it with IMU measurements. This method incorporates a sparse front-end based on ORB features, a pose-graph optimization in back-end, while it is able to close loops and re-use its map to reduce the drift in localization in mapped areas. While Visual-Inertial ORB-SLAM is based on ORB-SLAM there are some changes on the three parallel threads.

The tracking thread is responsible for tracking the sensor velocity, pose and the biases from the IMU, at frame rate. By the time the camera pose is predicted, map points from the local map are projected and correspondences with the keypoints from the current frame are matched. Subsequently, the current frame is optimized by minimizing the IMU error term and the reprojection error of all the correspondences.

The mapping thread executes local BA every time a new keyframe is inserted, but it optimizes only the last N (local window) keyframes and all the points seen by them. The cost function includes the IMU and the re-projection error terms. This is why the visual-inertial version is more complex than the visual-only, as it features 9 more states (biases and velocity) that need to be optimized for every keyframe. To achieve real-time performance an appropriate local window size must be chosen. While the mapping thread is in charge of discarding redundant keyframes, in the visual-inertial version it is done with a restriction. The motion of successive keyframes is constrained by the IMU data. However the bigger the temporal range between successive keyframes, the weaker is the information provided by the IMU. Hence, redundant keyframes are removed only if this does not cause two consecutive keyframes to differ more than a specific amount of time.

The loop closing thread performs pose-graph optimization only on 6 Degrees of Freedom (DoF), instead of 7, because the scale now is observable. While this pose-graph optimization ignores velocity and IMU biases, afterwards a full BA is executed in a parallel thread, optimizing all states, including biases and velocities.

Finally, Visual-Inertial ORB-SLAM introduces also a novel initialization method to calculate an initial estimation of gravity direction, scale, velocity, and accelerometer and gyroscope biases, given that some initial keyframes are processed by the SLAM algorithm. For a successful initialization, the SLAM algorithm has to be executed for a few seconds, assuming that the sensor moves in such a way that all the variables

[1] `https://github.com/raulmur/ORB_SLAM2`

(motions) are observable. The only restriction is that any two successive keyframes have to be close in time, so that the IMU noise integration is reduced.

More details can be found at [75], while an implementation of the algorithm is available online[2].

OKVIS

OKVIS [76] is a VI-SLAM approach that uses non-linear optimization for keyframe poses on a sliding window. Weighted inertial error terms are utilized along with weighted reprojection errors regarding visual landmarks, to form the cost function. For the vision part of the algorithm, features are extracted with multi-scale Harris corner detector and Binary Robust Invariant Scalable Keypoints (BRISK) descriptors are calculated for them. The keyframes that do not belong in the ongoing sliding window are not considered for the states being estimated. OKVIS is available online[3].

ROVIO

ROVIO [77] is a VIO implementation that is based on an Extended Kalman Filter (EKF). It utilizes FAST corner features, which are described by extracted multi-level patches located around them. Tracking of these patches is done by considering the IMU-predicted motion, while the photometric errors are utilized in the update step. ROVIO is available online[4].

VINS-Fusion

VINS-Fusion [78] is a VI-SLAM method that utilizes a sliding window for non-linear optimization. It detects Harris corners features and tracks them with a sparse KLT optical flow algorithm. For sensor fusion initialization it utilizes a loosely-coupled procedure, so that the estimator is bootstraped from arbitrary initial states. Furthermore, for relocalization a tightly-coupled process is used, while for the optimization the IMU data are pre-integrated prior to their use. VINS-Fusion is available online[5].

5.2 EuRoC Dataset

The EuRoC dataset is one of the most popular and high quality datasets available online. Since it features a drone with 6 DoF and a forward looking camera, it is the closest possible case to the one examined in this thesis, as the ROV also features a forward looking camera and 6 DoF. It is important to highlight the fact that the EuRoC was also chosen for its high quality of data and its clarity in documentation. More precisely, the data are organized in suitable formats and measurement units for VO applications, the utilized hardware specifications, parameters and configuration are available online,

[2] https://github.com/jingpang/LearnVIORB
[3] https://github.com/ethz-asl/okvis_ros
[4] https://github.com/ethz-asl/rovio
[5] https://github.com/HKUST-Aerial-Robotics/VINS-Fusion

and the groundtruth data are published as well. There are various examples of implementations and datasets online for VO, regarding the air and ground domain, however for subsea there is none. Configuring properly these implementations and the datasets needs much attention, as many factors might affect the resulting performance, making it hard to determine the root of a bad outcome and specify if the organization and the accuracy of the dataset is the reason for this or the configuration and set-up of the implementation. This fact becomes even more important in the case of acquiring and using proprietary footage and equipment, and until the phase where everything works as it is expected to, a reference base is mandatory for experimenting and configuring the whole system properly.

The EuRoC dataset fulfills this purpose during the experiments of this thesis. The five selected algorithms where deployed, configured and tested initially with the EuRoC dataset. By the time it was verified that they were established properly, it was possible to assess how the parameters of the dataset and the algorithms might affect the performance of the motion estimation process. The EuRoC dataset not only consisted a reference base for experimenting on the software aspect, but also an inspiration for configuring and setting-up later the in-house hardware architecture for VIO, aiming the subsea domain.

As by now the purpose of utilizing this dataset has become clear, only some indicative results are briefly presented in the appendix B, regarding mainly the performance of these VIO algorithms. This offers a rough first estimation on their performance.

5.3 UWSim Results

The first underwater dataset that was examined is a simulated one. The dataset comes in a ROS bag format and the images are in a ROS compressed format inside the bag file. Since the algorithms require the images to be in a non-compressed format, 3 nodes in ROS had to be executed in parallel. As it is illustrated in Figure 5.1 a node is playing the ROS bag and is publishing the compressed images, while another node is subscribed to the compressed images message, captures them, decompresses them and publishes them again. The last node is the one running the V-SLAM algorithm and it considers as input the decompressed images. This dataset does not provide explicitly groundtruth data, so a script was implemented, in Python utilizing the ROS Application Programming Interface (API), to extract the groundtruth data from the ROS bag file. The ROS bag includes a message type called “tf”, which describes the transformations between different coordinate frames during an experiment. Since the UWSim ROS bag includes a number of transformations between the utilized sensors and the parts of the ROV, only the transformation that refers to the ROV in relevance to the world was extracted for all the provided time instances. The information that was extracted and used as groundtruth trajectory was the transformation between the so named “world” and “girona500” coordinate frames. It was possible to exploit only the camera frames from this dataset, so the experiments were conducted only with ORB-SLAM2. The IMU data could not be used because the raw gyroscope and accelerometer data were not provided. The algorithm configuration details are presented in Table 5.1. The experiments were conducted on the “Laptop 1” setup.

In Figure 5.2 the estimated trajectory of the sequence “Second proposed” is illus-

Table 5.1: UWSim ORB-SLAM2 Parameters

Num of Features	Scale Factor	Pyramid Levels	FAST Thresholds
800	1.2	8	7 & 22

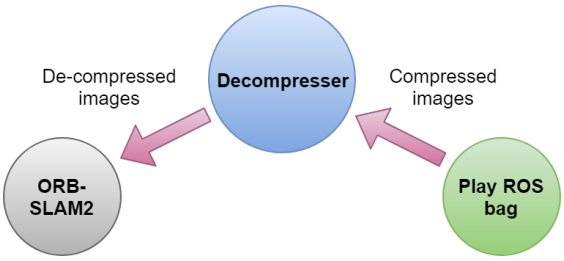


Figure 5.1: ROS nodes communication for UWSim dataset.

trated, and it is about 227 m long. In this graph only the poses of the keyframes are plotted, because attempting to use a line graph resulted in faulty visual results. From the resulting keyframe poses it is evident that the overall trajectory is estimated with adequate accuracy, with a mean Absolute Position Error (APE) being around 18 cm as the Table 5.2 presents. This trajectory provides many chances for loop closure, which ORB-SLAM2 achieved, decreasing significantly the drift. This shows that for this underwater case the place recognition process functioned properly with the extracted features. Additionally, the constructed map of the 3D cloud points of the seabed is presented in Figure C.1.

The other sequence that was examined from this dataset is the so called "Third proposed", which is quite more simple forming a triangular trajectory of approximately 57 m. In this case the same trajectory was examined with four different levels of turbidity

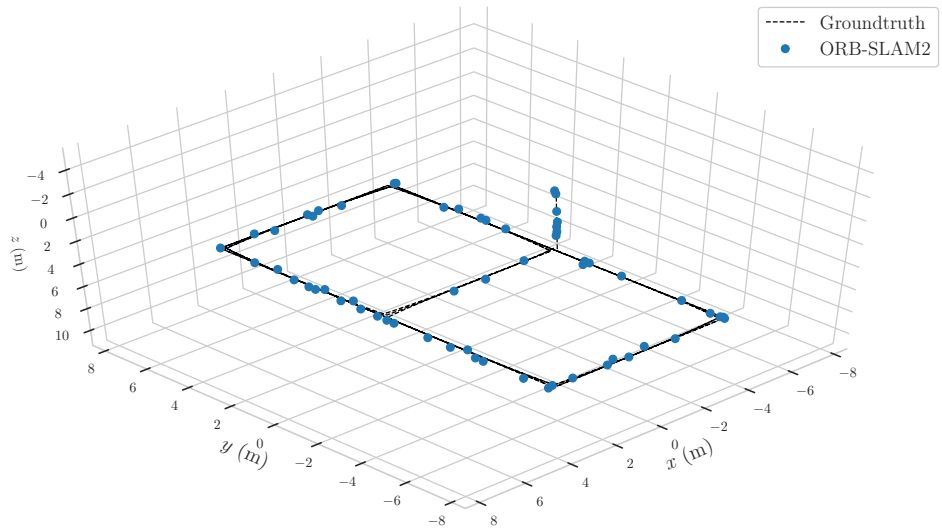


Figure 5.2: UWSim dataset, "Second proposed" sequence trajectory with ORB-SLAM2.

Table 5.2: UWSim "Second proposed" sequence Statistics

	Min	**Mean**	**Max**	**Std**
APE (m)	0.056566	0.182742	0.368493	0.072051
FPS	-	14.704217	-	4.125316

and its affect on the motion estimation was assessed. Figure 5.3 illustrates the APE of the estimated trajectory in relevance to the groundtruth. It is observed that ORB-SLAM2 performs with high accuracy and only during rotations the APE slightly increases. The performance of the algorithm during rotations is also obvious in Figure 5.4, in which roll and pitch slightly fluctuate around the groundtruth, with a maximum error of 4°. On the other hand, yaw, which is the significant one in this case, follows the groundtruth data quite precisely. A worth mentioning point is that, despite the estimated yaw follows the groundtruth pattern, its values are different from the groundtruth's. This is because the groundtruth data use a different reference system than the one that ORB-SLAM2 does. The algorithms that are used in this thesis to align the groundtruth and the estimated trajectories, take into consideration only the translation part. This is an example on the confusion that might be caused due to different utilized reference systems in VO. One interesting note is that during the visualization of this sequence and motion estimation, the loop closure effect and the reduction of drift became obvious. By the time the Remotely Operated underwater Vehicle (ROV) returned to its initial position, drift had already been accumulated. When the algorithm recognized that the initial position is a place already visited before it closed the loop and corrected the drift from the whole traversed trajectory by aligning the two sides of the loop.

The triangular trajectory has been also examined with four different levels of turbidity. The level of turbidity can be seen in the datasets Section 3.1. We would expect to see a drop in accuracy in the sequences with more turbidity, however the accuracy of ORB-SLAM2 is approximately the same for all of them. In fact in Figure 5.5 it is observed that the clear sequence (third 1) has even higher APE than the others. This is due to the non-deterministic nature of the algorithm, leading to different values of accuracy every time the experiment is conducted, but with a small deviation. One can say that for all levels of turbidity ORB-SLAM2 exhibits interestingly the same accuracy, as the mean value is very similar for all the four cases, with a deviation around 1 cm.

Despite the great results, the aforementioned behaviour of ORB-SLAM2, in reality, stands only for the first three levels of turbidity. The turbidity in the fourth level degrades the image quality and the amount of salient features significantly, forcing the algorithm to search for a longer amount of time to extract the required number of them. Furthermore, the algorithm possibly needs to process even more levels of the Gaussian pyramid, in order to extract the appropriate number of features. This causes significant delay to the tracking thread, which can not process fast enough the information of the current frame. As a consequence the tracking is lost.

While these observations are quite insightful for the structure of the ORB-SLAM2, ROS provides us with an option to examine whether the algorithm would also fail even if more processing power was available. This is achieved by playing the ROS bag in a slower speed, thus publishing the recorded data as they had been captured in a slower

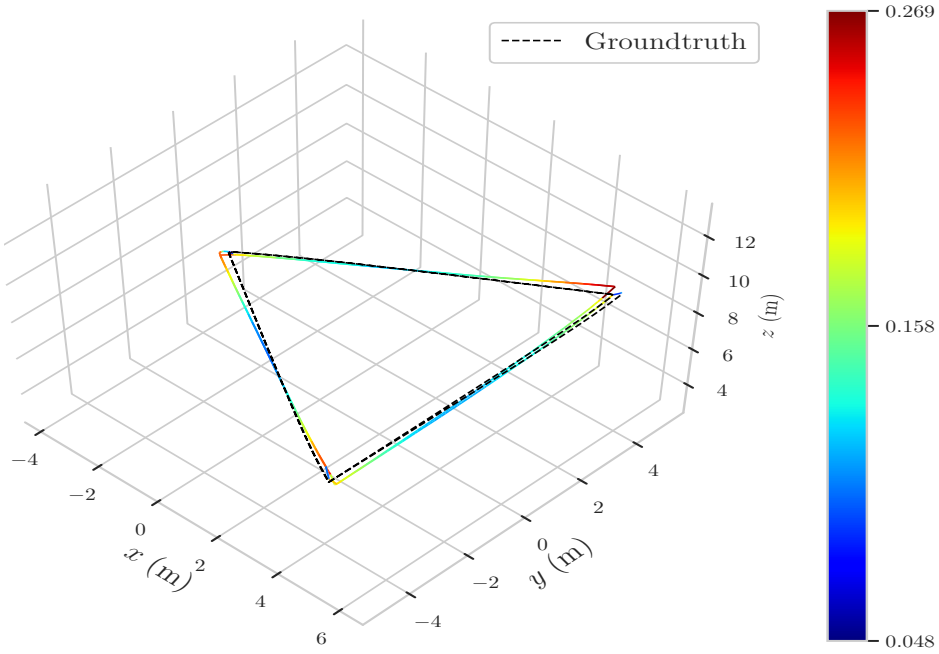


Figure 5.3: ORB-SLAM2 absolute position error in relevance to groundtruth for the UWSim dataset, "Third proposed" sequence, minimum level of turbidity. The colored bar and the colors mapped on the estimated trajectory indicate the level of the APE in meters (m).

rate. Whenever the bag was played at normal speed, ORB-SLAM2 was keep on failing on the fourth level of turbidity, no matter the different configurations that were tested. The ROS bag was executed at 20% of its original speed and the algorithm was able to successfully process the whole trajectory, resulting in the APE statistics presented at Figure 5.5. From this it is evident that the algorithm did not fail due to inability to extract features in this challenging environment, but due to the processing power.These results suggest that there is a chance that with Graphics Processing Unit (GPU) acceleration and optimizations the algorithm would be capable of succeeding at such challenging experiments.

Finally, in the Table 5.3 the performance of ORB-SLAM2 in Frames Per Second (FPS) for the four levels of turbidity is presented. It is obvious that with the utilized equipment the algorithm is capable of processing approximately two times the number of frames that are required (10 FPS) per second, in the tracking thread for the first three levels of turbidity. Also in this case applies that the algorithm is non-deterministic in terms of performance, however the results indicate that the level of turbidity might affect the efficiency of the tracking thread as well. This table shows that in the case with the highest clarity the algorithm is able to process more frames per second. This is reasonable, because if the quality of the extracted features and of their descriptors is low, the algorithm might need more time to determine which features are of good quality to extract and track.

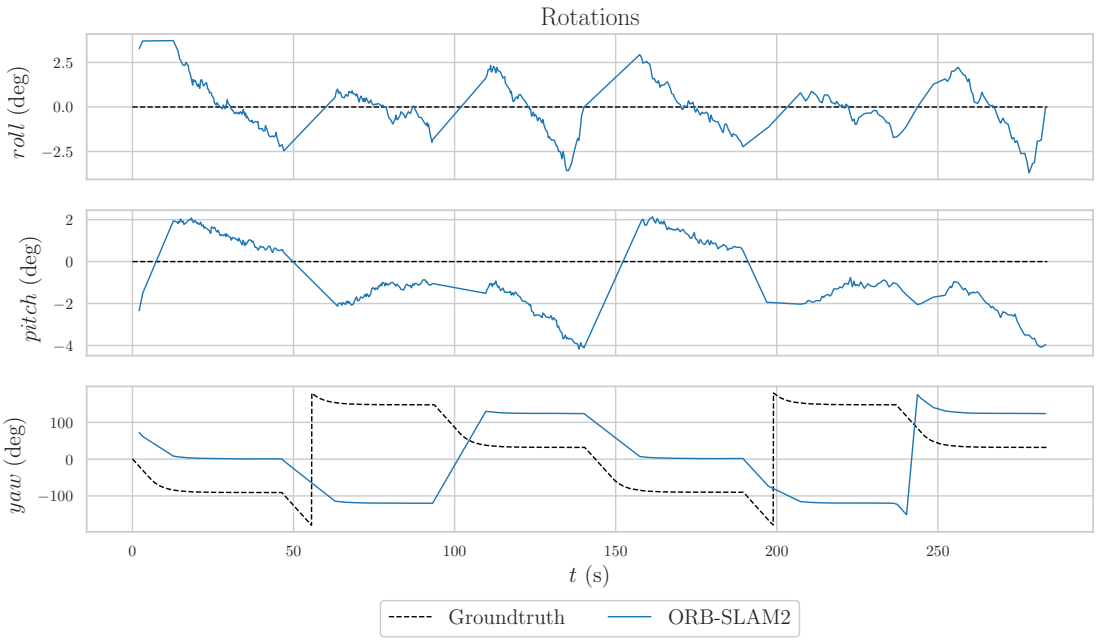


Figure 5.4: Rotations during UWSim dataset, "Third proposed" sequence, minimum level of turbidity with ORB-SLAM2.

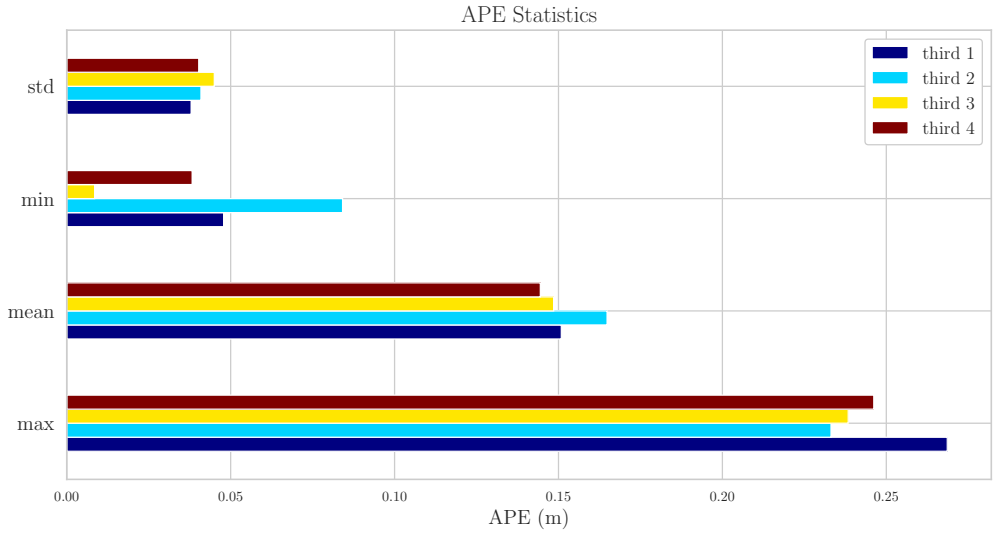


Figure 5.5: APE statistics for the four levels of turbidity of sequence "Third" of UWSim dataset, using ORB-SLAM2. The number 1 - 4 denote the level of turbidity with 4 being the worst case of clarity.

Table 5.3: UWSim "Third proposed" sequence FPS (four turbidity levels)

	Third 1	**Third 2**	**Third 3**	**Third 4**
Mean	21.792012	18.4607	19.819472	-
Std	6.294274	5.985929	6.217383	-

5.4 Aqualoc Dataset Results

The Aqualoc dataset provides seven different sequences and all of them have been examined with ORB-SLAM2 and Visual-Inertial ORB-SLAM. However, only the results for two of them are presented here, as the conclusions are approximately the same for the rest as well. Initially the "sequence 05" was selected and executed with ORB-SLAM2, because it is quite challenging and covers many of the existing conditions of the whole Aqualoc dataset. For the second part of the experiments the "sequence 06" was selected and tested with the ORB-SLAM2 and the rest of the VIO algorithms that are examined in this thesis.

The estimated trajectory of the "sequence 05" is illustrated in Figure 5.6 and it is approximately 28 meters long. This sequence features low vision parts and many floating particles, while the ROV exhibits some rapid motions and rotations. Up to this point of the presented results ORB-SLAM2 has shown satisfying and adequate performance. However, it is notable in this case how quickly its accuracy might degrade in very challenging environments with intense rotations. In Figure 5.6 the estimated trajectory resembles the groundtruth, but the algorithm failed to capture many parts of it. Furthermore, it is evident that the algorithm failed to achieve loop closure.

To interpret this outcome we have to consult the graphs in Figure 5.7, which represent the translation and rotations throughout the whole experiment. Taking a look at Figure 5.7a, the estimated translations exhibit some errors, but they do not diverge extensively from the groundtruth. The most significant reason of failure seems to be the rotations estimation, which are illustrated at Figure 5.7b. Pitch, roll and yaw estimations are notably deviating from the groundtruth data, probably causing the significant miss-estimation of the trajectory. It is well known that ORB-SLAM2 does not cope very well with quick and absolute rotations and here this might be the issue.

Despite the challenges during this sequence, ORB-SLAM2 still produces a rough estimation of the traversed path. Although floating particles pass in front of the camera introducing notable noise, ORB-SLAM2 still manages to keep track and estimate the motion. Some of the features detected on moving particles are considered as outliers and they are rejected, hence the trajectory estimation is not completely failing.

The "sequence 06" produces better results in accuracy, and for this reason it is utilized to examine the performance of the VIO algorithms. The "sequence 06" is approximately 19.5 meters long and it was utilized to compare all the implementations that are assessed in this thesis, as it features also IMU data. ORB-SLAM2 and Visual-Inertial ORB-SLAM were executed successfully on this dataset. Unfortunately it was not possible to properly execute the rest three VIO algorithms with the provided calibration IMU parameters and IMU measurements from this dataset. VINS-Fusion, OKVIS and ROVIO are sensor fusion implementations with a lot of configuration parameters for the IMU on top of

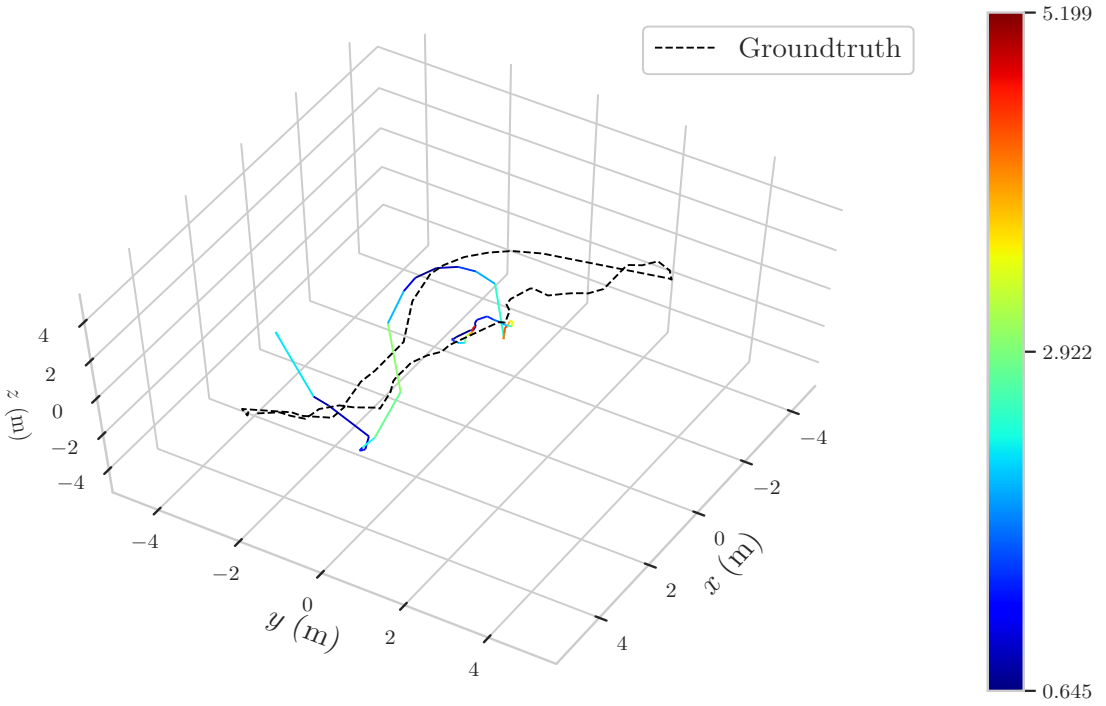
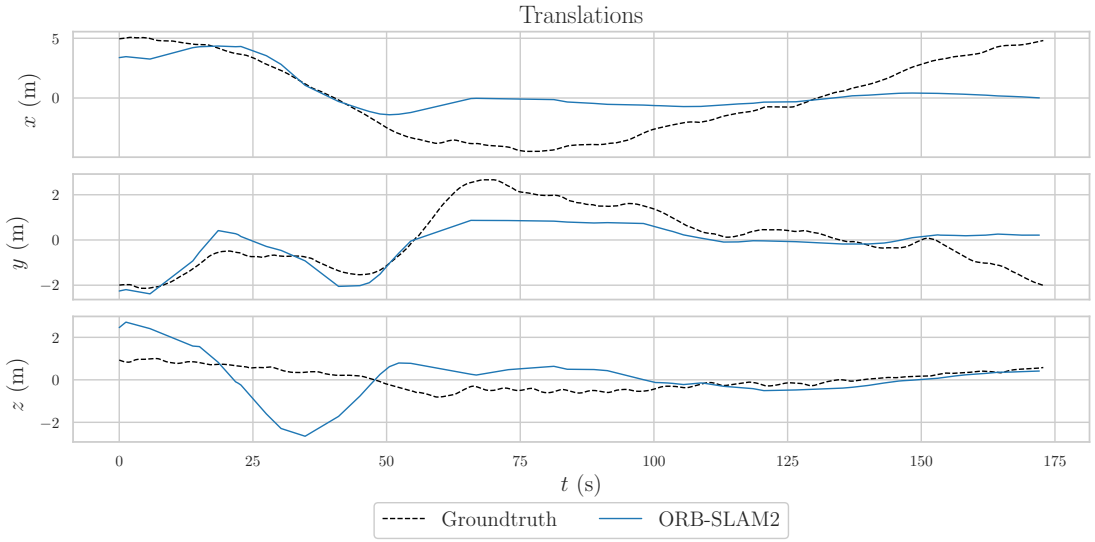
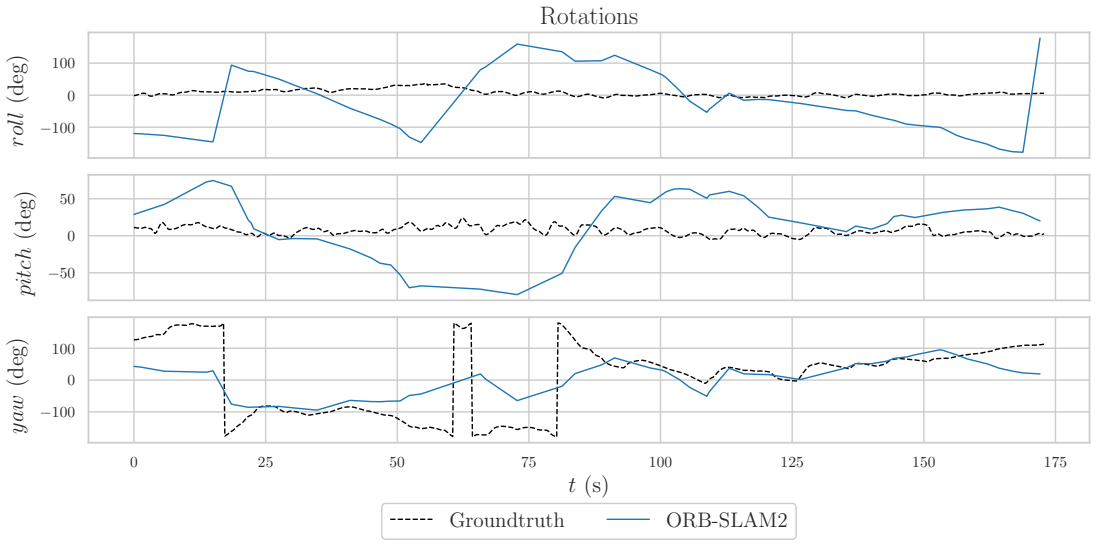


Figure 5.6: ORB-SLAM2 absolute position error in relevance to groundtruth for the Aqualoc dataset, sequence 05. The colored bar and the colors mapped on the estimated trajectory indicate the level of the APE in meters (m).

the VO part. Some of these parameters can be derived during hand-eye calibration between the camera and the IMU. This dataset was calibrated with the most popular and most used tool for calibration, the Kalibr, so we trust that the hand-eye calibration parameters have the proper format and quality. However, here is another point where the level of ambiguity is significant on the examined domain. Except from the hand-eye calibration, for the configuration of these algorithms also some other IMU parameters are required, like "noise density" and "random walk", for both the accelerometer and the gyroscope. The ambiguity is present in the sense that these two terms are used with various names, and even sometimes one might be referred by the name of the other. The Aqualoc dataset provides these values and they were set in the algorithms as they are. These three VIO algorithms did not function as expected, i.e. they quickly and significantly diverged, resulting to unreasonable motion estimations. While investigating the root of the problem, an important outcome was that these implementations are very sensitive in relevance to the accuracy of these IMU parameters. A small change in these values would result in a rapid divergence or would lead to a smooth motion estimation for a small amount of time, before they completely fail. On top of that the ROVIO features the biggest amount of parameters to be investigated and configured, which was out of the scope of this thesis. Setting up and building these implementations was very time demanding, and the fact that they did not function properly in the end was very unfortunate.



(a)



(b)

Figure 5.7: Estimated (a) translations and (b) rotations, compared with the groundtruth for the Aqualoc dataset, sequence 05.

5.4.1 ORB-SLAM2 and Visual-Inertial ORB-SLAM Performance

Despite the fact that it was possible to compare only two of the five implementations, some interesting results were observed. The ORB-SLAM2 and Visual-Inertial ORB-SLAM are based on the same visual motion estimation algorithm, while Visual-Inertial ORB-SLAM adds on top of that the fusion of the IMU, offering the opportunity to examine if the sensor fusion leads to an improved performance. The estimated trajectories

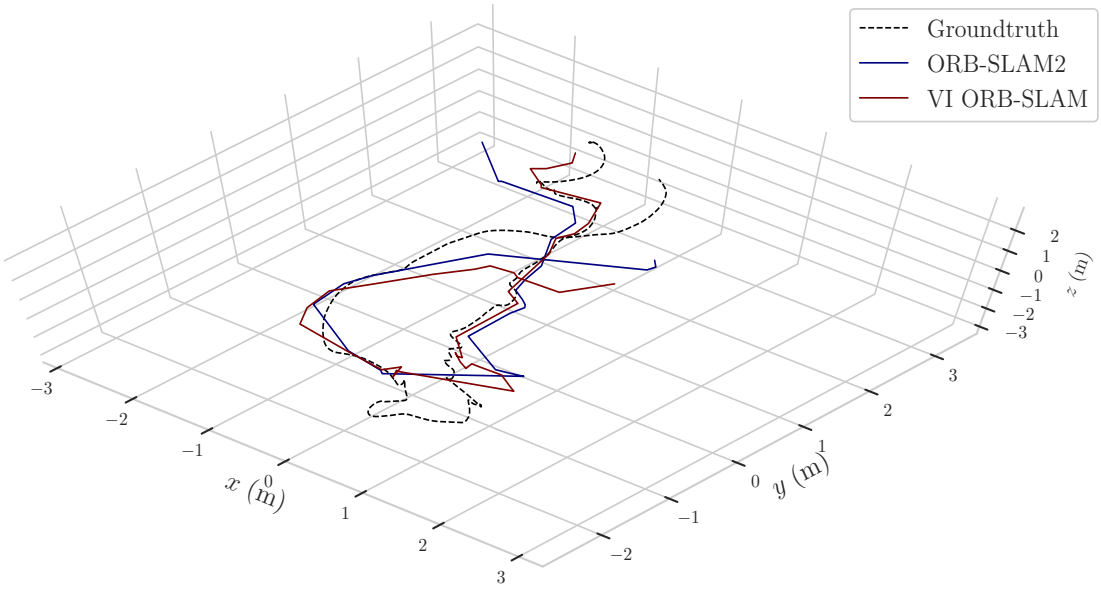


Figure 5.8: Trajectories comparison for the Aqualoc dataset, "sequence 06" using the ORB-SLAM2 and Visual-Inertial ORB-SLAM. The resulting trajectories have been undergone scale alignment.

are compared in Figure 5.8, after they have been undergone scale alignment as well, due to the utilization of ORB-SLAM2. Keeping this in mind, it is evident that the addition of the IMU contributes not only to the accurate scale estimation, but also to the increase of motion estimation accuracy. This becomes even more obvious by taking a look at the statistics in Figure 5.9, where the Visual-Inertial ORB-SLAM exhibits better performance. If we also take into account that the Visual-Inertial ORB-SLAM estimates the absolute scale as well, it is superior to the ORB-SLAM2 implementation and a complete approach.

After the observation that enhancing the ORB-SLAM2 with an IMU fusion leads to better results, the next question to be answered is whether the sensor fusion has a negative impact on the execution time of the algorithm. For these measurements the "Laptop 2" setup was utilized. Interestingly, the Table 5.4 shows that the performance in terms of speed is approximately the same for both the algorithms. This contradicts our expectations, considering the theory. The case of the Visual-Inertial ORB-SLAM is more complex, because also the IMU states are considered during the optimization per frame. For this reason in Visual-Inertial ORB-SLAM the number of keyframes that are used in BA can be limited to achieve a real-time performance. Although a slower performance was expected for the Visual-Inertial ORB-SLAM, probably the number of keyframes that was set for BA causes the algorithm to achieve similar performance to ORB-SLAM2. The results indicate that the approach of sensor fusion in Visual-Inertial ORB-SLAM is efficient and produces a complete approach for the motion estimation of

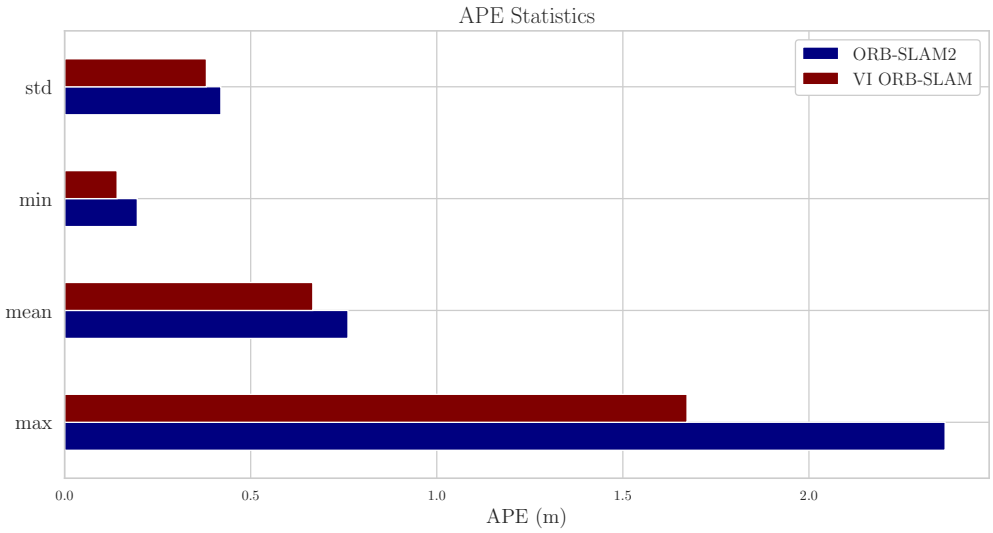


Figure 5.9: APE statistics, for the Aqualoc dataset, "sequence 06", for ORB-SLAM2 and Visual-Inertial ORB-SLAM.

Table 5.4: FPS for ORB-SLAM2 and VI ORB-SLAM
Aqualoc - "sequence 06"

	Min	**Mean**	**Max**	**Std**
ORB-SLAM2	5.41706	24.027201	44.4576	4.597301
VI ORB-SLAM	7.6064	24.000441	38.5107	3.970403

monocular setups, without the need for post-processing to calculate the absolute scale.

5.4.2 Mapping

Another interesting aspect of the Visual-Inertial ORB-SLAM is the construction and the maintenance of a global map. Although creating a map is very important for relocalization and for drift reduction with loop closing, maintaining a large one might be costly. To assess the impact of the map size on the memory, experiments were conducted to examine how fast a map can grow with relevance to the time and the number of features utilized. Before analyzing the results, we have to take into account that the environment plays a significant role in the evolution of map size. Even if the maximum number of features is set to a specific value, there might be the case that the environment can not provide so many features. In the examined dataset the environment provides a vast amount of features, setting obvious the impact of the number of features on the map size in Figure 5.10. It should be also considered that the ROV is mostly moving along with the time, visiting new places and capturing frames with new information. The map size is described by the equation (5.1),

$$MapSize = sizeof(keypoint) * NKP + sizeof(keyframe) * NKF \tag{5.1}$$

with

$$sizeof(keypoint) = 696\ Bytes,\ sizeof(keyframe) = 3648\ Bytes \tag{5.2}$$

where NKP is the number of keypoints and NKF is the number of keyframes. It also has to be mentioned that the results are indicative, because the algorithm is non-deterministic, meaning that it would give slightly different behavior on the size of map for every execution. From the first look in Figure 5.10 it is evident that the line representing the 800 features exhibits different tendency compared to the others, in the sense that it never decreases. The explanation of this lies in the methodology that ORB-SLAM follows for culling the redundant keypoints and keyframes. Initially, new keypoints and keyframes are added generously into the map, however if later they do not contribute to the map a notable amount of new information, they are discarded. In the case of 800 features, because the number is relatively low, the stored information has no significant overlap from keyframe to keyframe, so most of the new keypoints and keyframes are maintained, resulting in a steadily increasing size of a map. On the other hand, the bigger the number of features, the more redundant information is provided by the keyframes and keypoints. This is obvious especially in the line representing the 3500 features, where the size of map fluctuates significantly, meaning that from time to time the redundant keyframes and keypoints are discarded. It is reasonable that the more features detected in every frame, the highest the possibility to have overlapping information regarding the environment. This indicates that is not always required to increase the number of detected features, as many of them provide overlapping information and they increase the CPU load.

Observing all the lines in the graph, and taking into account the culling of keyframes and keypoints, we could say roughly that the map size grows linearly with relevance to the time, as the ROV is continuously moving. While the ROV traversed approximately 19.5 meters in 2.10 min the map size reached the 7.5 MB maximum. Assuming that the robot would move with the same approximately velocity, and that the map size tendency would keep this linear growth, in 24 hours of travelling the size would reach at 5143 MB = 5.143 GB. Although this is an extreme scenario, as the ROVs operate for less amount of time and they travel around a specific region, visiting the same places, the map size is manageable for an embedded system.

5.4.3 ROV Design

A last comment on the Aqualoc dataset experiments, regards the significance of the ROV design. In Figure 5.11 the red circle marks an area, where the floating particles are more obvious in the acquired image. This phenomenon is present for most of the times during the footage acquisition. What can be derived from this is that placing the lights close to the camera might cause such kind of negative effects. Mounting the lights on that part of the ROV makes the bottom right part of the images not only useless, but also a significant source of noise. A huge amount of floating particles are constantly visible and features are detected in this area. Of course these features are outliers, because the particles' motion is vague and different from the ROV's, making more difficult and less accurate the motion estimation. These observations suggest that it is very important to strategically design the ROV architecture (camera and lights position) for better VO

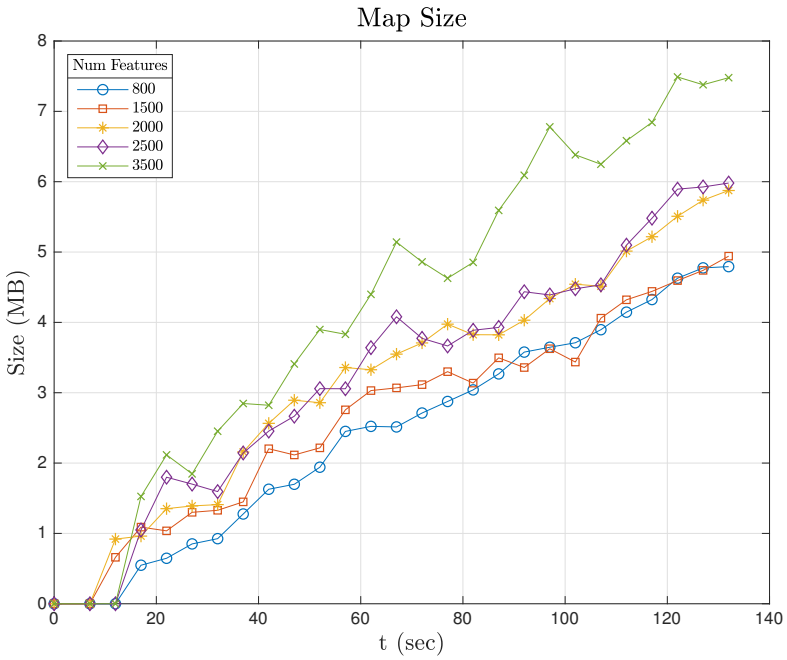


Figure 5.10: Evolution of map size in MB with relevance to the time.

Figure 5.11: The red circle marks the area where the phenomenon of floating particles is more intense, in the Aqualoc dataset.

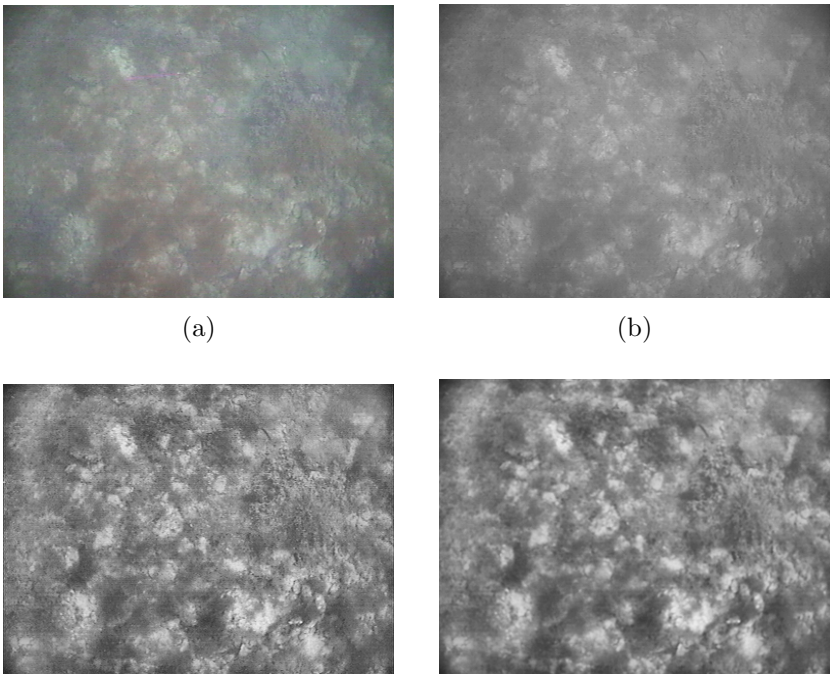
performance, while they highlight the various number of factors that might affect the efficiency of the VO algorithms.

5.5 Underwater Caves Dataset and Image Processing

This dataset is taking place in underwater cave complex and it is the most challenging of the underwater datasets that were examined during this thesis. Unfortunately, no matter the amount of ORB-SLAM2 configurations that were tested, the algorithm did not manage to process the whole sequence successfully. In fact, it looses track a little bit after the beginning of the footage. Various numbers of features were tested and even the FAST thresholds were set in very low values, but still the algorithm kept on loosing track, especially in a specific area with only sand and not at all salient features. Furthermore, it was attempted to lower the ROS bag playback speed, as it was successfully done in UWSim dataset, but it did not make any difference. In addition, the IMU data could not be utilized, because the dataset does not provide the hand-eye calibration between the camera and the IMU. Hence, it was not possible to examine the performance of the VIO algorithms. However considering the notable failure of ORB-SLAM2 and the challenging conditions, most probably the other algorithms would fail as well.

Although the ORB-SLAM2 failed to perform adequately in this dataset, this case offered the chance to try some optimizations, as far as the image quality is concerned, and examine whether they would make any difference in the performance. After researching and determining some promising image processing filters from OpenCV in C++, they were embedded into the algorithm in such a way, so that they are executed in real-time during the VO pipeline. Observing the challenging and noisy quality of the footage, the first approach was to remove the noise with a Gaussian blur or a Non-local Means Denoising filter. While the result was much better for the human eye, it did not make any difference for the algorithm. The next attempt was to enhance the contrast of the image, so that salient features would pop out. This was attempted, not just with a simple contrast enhancement filter, but also with Contrast Limited Adaptive Histogram Equalization (CLAHE). The results were impressive for the human eye, but still for the algorithm nothing changed towards a better performance. The final attempt was to apply the CLAHE and a Gaussian blur filter as well, however leading again to failure.

Figure 5.12 shows the original acquired image, its grayscale version, the result of CLAHE, and the outcome of combining the CLAHE and a Gaussial blur filter. The challenging image quality can be observed in the original picture. In Figure 5.12c the result of contrast enhancement is a more sharp image, but the noise is amplified as well. Figure 5.12d illustrates the impact of the Gaussian blur on top of contrast enhancement, making evident that along with the noise elimination, the sharpness of the features is also degraded. An interesting outcome is that the image processing not only did not improve the performance of the algorithm, but notably it made it even worse. When the filters were applied, the ORB-SLAM2 failed in parts of the dataset that was able to successfully process in their bare original form. As disappointing is this result, it is also reasonable. By applying these filters in every frame independently, the characteristics of the features are radically changed. From frame to frame the lighting conditions and the neighbour pixels of a feature are changing, so the impact of the same image processing filter will lead to a different result from the previous frame, essentially introducing even more “noise” for features. In this way the filters alter the characteristics of the features from frame to frame, making it more difficult to track them. While the image processing attempts



(a) (b)

(c) (d)

Figure 5.12: Image processing filters on the underwater caves dataset. (a) Original image, (b) grayscale image, (c) contrast enhancement, (d) contrast enhancement and gaussian blur.

were tested initially on the underwater caves dataset, later were examined also for the other datasets, resulting in the same outcome, worse algorithm performance. Maybe it could be possible to successfully exploit image processing filters, but this would require to apply them in a more sophisticated way, by relating their impact on features from frame to frame, and not being applied independently, significantly altering the features' characteristics. Eventually, perhaps this would lead to the increase of processing power needs, so the trade-off should be carefully assessed. Finally, despite the failure of ORB-SLAM2 in this dataset, it is worth mentioning that such cases, with nearly none features present, will probably not be faced in the examined cases of this thesis, as it aims to structured underwater environments.

5.6 Archaeological Dataset Results

This is the final underwater dataset that was found available online and examined in this thesis. Taking into consideration that the UWSim dataset was a simulated one, the Aqualoc dataset was very challenging with intense changes in motion and the Underwater Caves dataset did not produce any usable results, the Archaeological dataset was selected for some more thorough experiments. The "sequence 01" is approximately 100 meters long, and it was selected as it is quite smooth and produced adequate accuracy in motion estimation. Hence, by varying the ORB-SLAM2 parameters we can examine

Table 5.5: Default ORB-SLAM2 Parameters
Archaeological Dataset

Num of Features	Scale Factor	Pyramid Levels	Fast Thresholds
1500	1.2	8	7 & 20

their affect on execution time and accuracy. Although this dataset does not provide IMU measurements and the VIO algorithms can not be examined, the results will be useful for the Visual-Inertial ORB-SLAM as well, as it uses a similar visual motion estimation algorithm as the ORB-SLAM2.

In this experiment four important parameters of ORB-SLAM2 and Visual-Inertial ORB-SLAM are assessed. These parameters and their "default" values are shown in Table 5.5. For every examined parameter, its value will be varying while the rest of the parameters will be constant with their "default" values. The experiments were conducted using the setup "Laptop 2". Before evaluating the impact of the parameters, the trajectory of "sequence 01" is illustrated in Figure 5.13 and its 3D map in Figure C.2, which are the results of ORB-SLAM2 with the "default" values, to give an insight of the traversed path that is examined. It can be observed that the accuracy is neither great nor bad, with a mean APE at 1.65 meters and the maximum error taking place during a rotation, as it was expected.

Figure 5.14a illustrates the impact of the number of utilized features on ORB-SLAM2 performance. As it was expected, by increasing the number of features the execution speed of the algorithm decreases, as it has to process a greater amount of points. It has to be mentioned that for the case of 500 features, the algorithm struggled to initialize, as it requires a specific number of features to be extracted, matched and to be of a certain quality. While the impact on the FPS meets our expectations, the resulting APE does not. It was expected that as the number of features grows, the APE would decrease, at least slightly, but instead is quite stable around 1.6 meters. The small deviation around this point are due to the non-deterministic nature of the algorithm. The explanation for the behavior of APE is straight forward. While the algorithm is capable of detecting and extracting more features, it does not mean that they are also useful for estimating the motion. If a certain amount of detected features is well distributed across the whole image and their quality is adequate to be matched across frames, then any extra features are redundant. This shows that there is an ideal value of number of features for every case, that leads to an efficient trade-off between speed and accuracy. For this case this value is around 1000 features, because with 500 features the algorithm struggled to initialize.

The impact that the number of pyramid levels has on the algorithm's performance is depicted in Figure 5.14b. By increasing the number of levels, ORB-SLAM2 searches for features in more pyramid levels, resulting to the need for more time to finish the process of every frame. Same as before, the APE seems stable throughout the experiment. For the examined case this is reasonable. The different pyramid levels are useful when features are present in various sizes and motion takes place in various levels, i.e. if motion can be estimated by observing small and large salient areas as well. To be more precise, maybe a huge uniform area of an image is moving, however does not contain any small

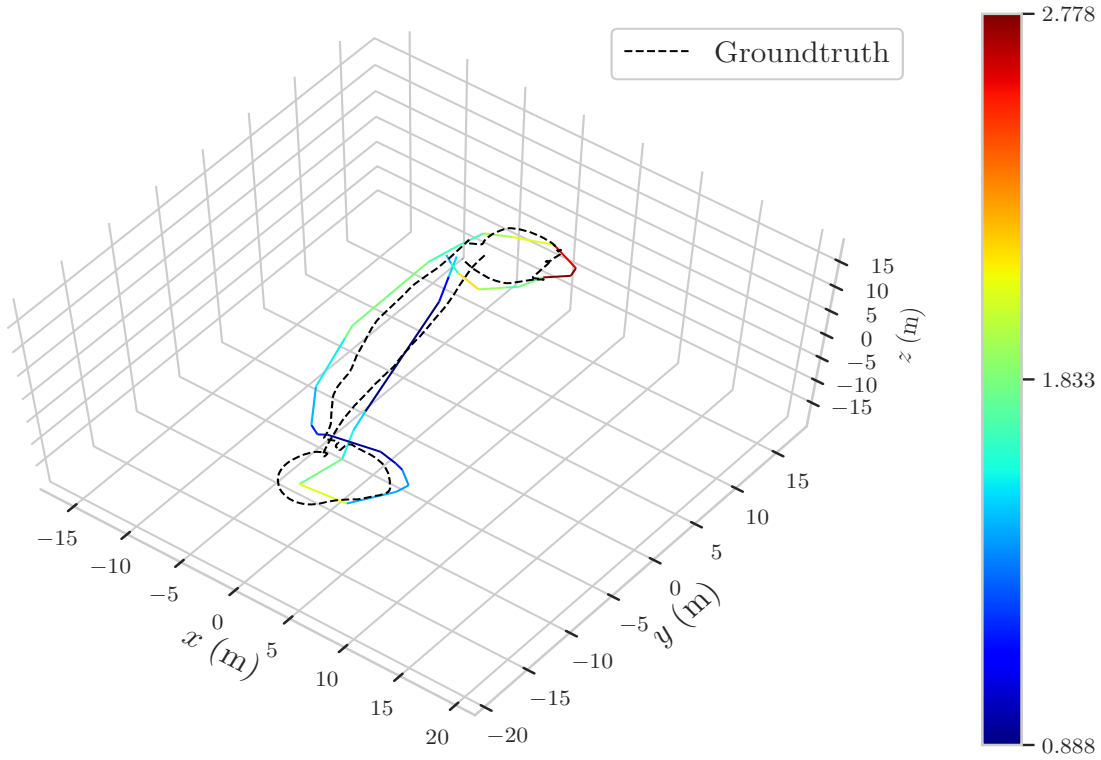
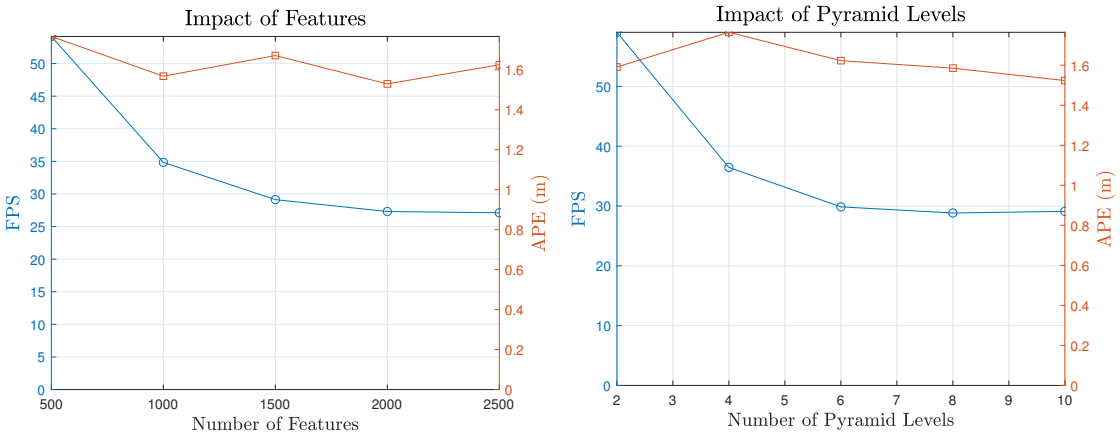


Figure 5.13: ORB-SLAM2 absolute position error in relevance to groundtruth for the Archaeological dataset, "sequence 01". The colored bar and the colors mapped on the estimated trajectory indicate the level of the APE in meters (m).

features that can be extracted. If the image is blurred and its size is reduced, the size of the area is reduced as well, and it can be captured as feature, thus the motion is also detected. For the case of the archaeological dataset, many of the available features are small, like holes on the seabed, and motion can be estimated even with just two pyramid levels, without reducing anymore the size of the image. However, it is evident that as the number of levels is increasing above 6, the APE is improved, most probably because there are also some big archaeological objects in the scene, which can be tracked as well.

Figure 5.15a presents the impact of the scale factor, i.e. how much the image is resized for every pyramid level. From level to level the size of the image is gradually reduced by a scale factor, to captured the motion in larger scales. It is reasonable that as the scale factor increases the execution time will be improved, because the image will shrink in size faster and calculations will have to be executed on a smaller amount of pixels. As far as the APE is concerned, it is shown that around the value 1.7 lies the optimal case for accuracy. It was also mentioned for the previous graph that there are some large archaeological objects in the scene, which can be also tracked if the image size is significantly reduced. The accuracy is decreased above the value 1.7 most probably because low quality features are tracked and lead to erroneous motion estimation.

The final group of experiments regards the values of FAST thresholds, and the results



(a) (b)

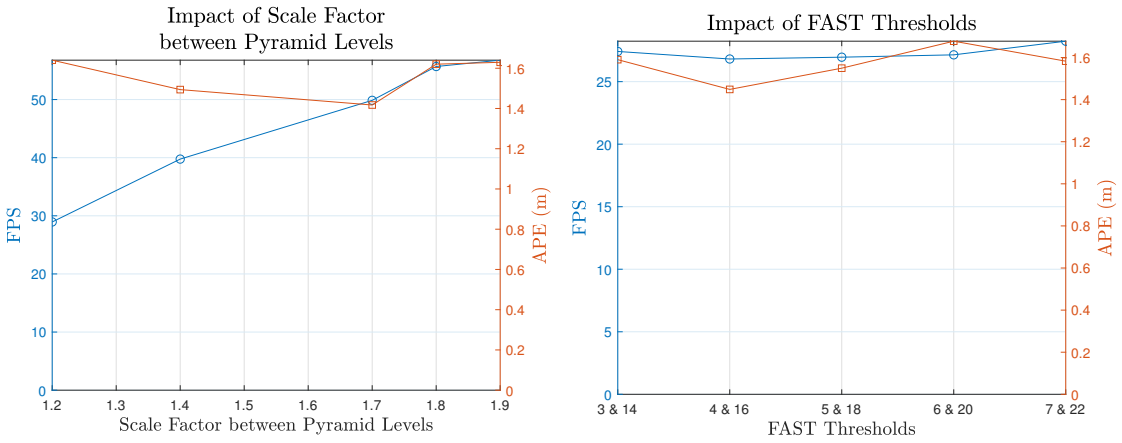
Figure 5.14: (a) Impact of number of features, and (b) Impact of the number of scale pyramid levels, on the FPS and APE, using ORB-SLAM2 on the Archaeological dataset, "sequence 01".

are presented in Figure 5.15b. The graph indicates that the FAST thresholds in this video sequence do not have any impact on the FPS. It would be reasonable to observe a decrease in FPS along with the reduction of these thresholds, because theoretically more features would be detected and extracted, resulting to a higher demand of processing power and time. It seems that, for this case, the detected features are of adequate quality and similar amount of features that are detected for low thresholds are also detected for higher ones. An interesting point to highlight for this graph is the optimal values regarding the APE. These values are 4 for the lower and 16 for the upper threshold. We would expect that for greater values the APE would remain stable or even decrease, as higher quality features are extracted. An explanation to this is that, due to the higher threshold, less features are extracted, which might lead to uneven distribution of features across the image. In that case the motion estimation would be less accurate, producing higher APE.

It is obvious that determining the ideal parameters for a VO algorithm is not trivial at all. The results are usually not straight forward and have to be thoroughly examined to interpret them. There is a huge amount of factors that can affect the performance of a VO implementation, and they have to be exhaustively investigated for robust results, for every case. For every type of environment and for various kinds of motions these optimal parameters change, that is why a significant number of different environments and motions have to be investigated, especially if the goal is to have an efficient implementation that covers a lot of cases.

5.7 VIO System Implementation and Results

Up to this point the most suitable and recent approaches of VO have been examined on all the available underwater datasets, while the subsea challenges and the performance of



(a) (b)

Figure 5.15: (a) Impact of scale factor between pyramid levels, and (b) Impact of FAST threshold values, on the FPS and APE, using ORB-SLAM2 on the Archaeological dataset, "sequence 01".

these algorithms have been assessed. However, the ultimate goal of this thesis is to exploit all the knowledge and the results produced by the conducted study and experiments, to design and deploy a proprietary VIO system that will cover the needs of the underwater structured areas case. Besides the system implementation, the aim is to optimize on it the most suitable VIO/VI-SLAM algorithm, and to produce proprietary datasets in the field, to test the whole system, including the hardware and software aspects. The examined VIO system is comprised of a machine vision camera, an IMU, a micro-controller and an embedded platform from *NVIDIA*. The specifications of all the components have been presented in Section 4.2.

As it has been stated, the *NVIDIA* Jetson TX2 was used for data acquisition, because we had at that time the appropriate carrier board to interface it with all the peripherals. On the other hand, the *NVIDIA* Jetson Xavier was used to execute and evaluate the chosen VIO algorithm. The fact that the Jetson TX2 was utilized to record data, does not have any impact on the initial goal to evaluate the performance of the new Jetson Xavier for VIO implementations. One of the most challenging tasks and main issues of a VIO implementation is the sensor fusion part. The fusion of the camera and the IMU is a highly researched field, and still in these days is considered to be of a high interest topic. While the sensor fusion is inherently a very challenging task, in VIO implementations it gets even more important, as these algorithms are very sensitive to hardware synchronization and timing issues. As the VO is a process during which errors are accumulated, if there is even a slight misalignment between the camera frames and the IMU data, the negative impact on the results might be severe.

5.7.1 System Architecture & Synchronization

During motion estimation it is very important to relate the calculated motion from a specific camera frame to the calculated motion from the IMU data. If the two sensors

are not precisely synchronized, then the IMU data might be related to the next or the previous frame, instead of the desired one, and this already has a negative impact on the resulted accuracy. The timestamping of the camera frames and the IMU messages is very important and it has to be done in a detailed and well organized way. The strategy of timestamping must be planned by taking into account possible bottlenecks, such as processing power or bandwidth. For example, if we decide to timestamp a camera frame by the time it arrives at the embedded system, then the amount of processing load on the CPU has to be checked, and also the size of the camera frame. For high resolution images there will be a significant overhead while transferring the frame from the camera to the embedded system memory, resulting in a delayed and bad timestamping. In addition, the embedded system CPU is responsible to handle other ongoing tasks as well, so even if the camera frame transfer delay is small, there is still a high possibility that it will not be timestamped immediately by the CPU. It is becoming evident that precise control of incoming data and timestamping is mandatory. Triggering of camera and IMU can provide us with such kind of control, while it is obvious that bursts of incoming data at vague time instances is harder to handle. However, even with triggering the possibility of synchronization failure is still present. For example, if a trigger is sent at the same time from the embedded system to both, the camera and the IMU to request data for the specific instance, the IMU will send the acquired data faster, as the camera comes with the overhead of capturing the photons and forming the image on its integrated FPGA, while it is a fact that the exposure time might vary. One more thing to consider is that the IMU operates at high rates, like 200 Hz, while the camera functions at lower, like 30 FPS, making it more difficult to synchronize the data acquisition of these two different sensors.

Taking into account all the challenges that come along with sensor fusion, the architecture of the proposed VIO system, presented in Figure 5.16, is evaluated. To avoid possible delays in timestamping, due to the NVIDIA system CPU overload, a micro-controller is utilized to assign timestamps to the camera frames and the IMU messages. The micro-controller is integrated on the carrier board, and it is connected with the embedded system, the IMU and the camera. While the IMU is hardware triggered by the micro-controller, the camera is software triggered by the embedded system. To have a more thorough understanding of the system functionality two detailed graphs are presented in Figures 5.17 and 5.18, regarding the camera and the IMU correspondingly.

In Figure 5.17 it is observed that the camera acquisition starts by a software trigger sent from the embedded system to the camera over USB 3. By the time the camera is triggered, a high level signal is transmitted to the micro-controller from the camera General-Purpose Input/Output (GPIO), and it determines the beginning of image acquisition. The transmitted signal is constantly high during image acquisition. The moment that the image is completely captured, the transmitted signal turns to low, determining the end of frame acquisition. The micro-controller timestamps both, the beginning and the end of acquisition, however we use only the timestamp that corresponds to the end time. On the embedded system side, after software triggering the camera, some processing takes place for the current frame, and after a reasonable amount of time this processing is temporarily interrupted to send a request message to the camera. The request message should be send after the image has been captured, i.e. in a time instance

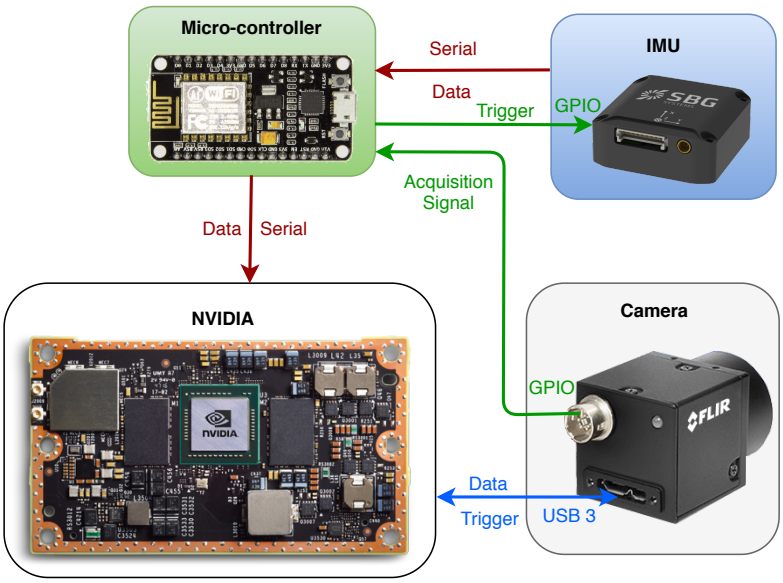


Figure 5.16: VIO system architecture.

greater than the exposure time of the camera. At the same time that the captured frame is sent over USB 3 to the embedded system, the micro-controller transmits a timestamp for this frame over a serial connection to the embedded system as well. In this way timestamping delays, due to CPU overload or image transmission overhead, are avoided. In the graph two times, denoted by t_1 and t_2, are present on the embedded system side. The t_1 and t_2 are the times that the micro-controller timestamp and the captured frame arrive at the Jetson TX2 correspondingly. These times have similar values and by comparing them we can associate an incoming timestamp to an incoming frame.

On the other side of the micro-controller lies the IMU, which is capable of providing data at a rate of 200 Hz. However, in offshore operations long cables are utilized, which are more sensitive to losses. To face the losses, a baud rate of 115 kb/s is used, which is not adequate to transmit IMU data produced at 200 Hz. The baud rate is set in both sides, the IMU and the micro-controller. Hence, the micro-controller hardware triggers the IMU through a GPIO at 100 Hz. After triggering, the IMU transfers the acquired data to the micro-controller, which by its turn transmits the timestamped data to the embedded system. Here it has to be mentioned that the camera and the IMU are not truly synchronized. If they were truly synchronized there would be a common timestamp for the two sensors in constant ranges of time. In our case the IMU data are related to the camera frames by considering IMU timestamps that are close to frames timestamps, with their difference being below a certain threshold.

5.7.2 Machine Vision Camera Interface

The machine vision camera that was utilized is from FLIR and offers a wide variety of configuration options along with event handling. As it is a machine vision camera some low level interfacing had to be done between the camera and the computing device

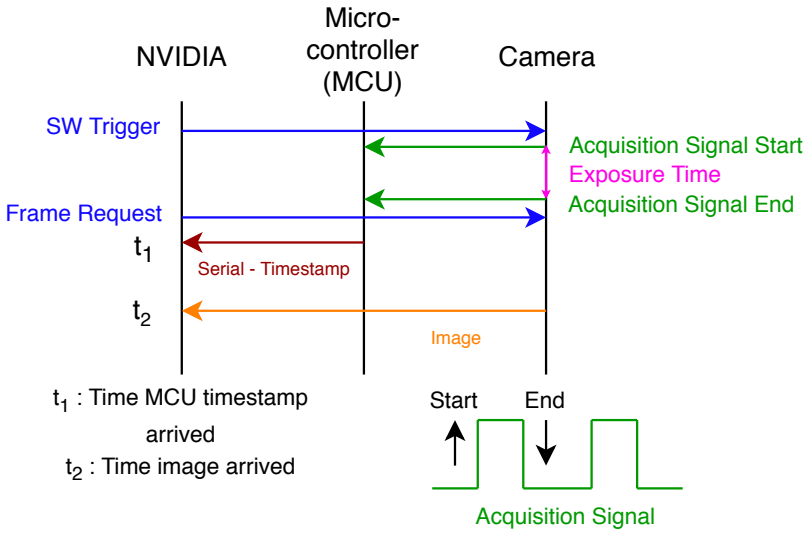


Figure 5.17: Illustration of communication and utilized signals for the camera. t_1 and t_2 are the times that MCU timestamp and frame arrived at the embedded system correspondingly.

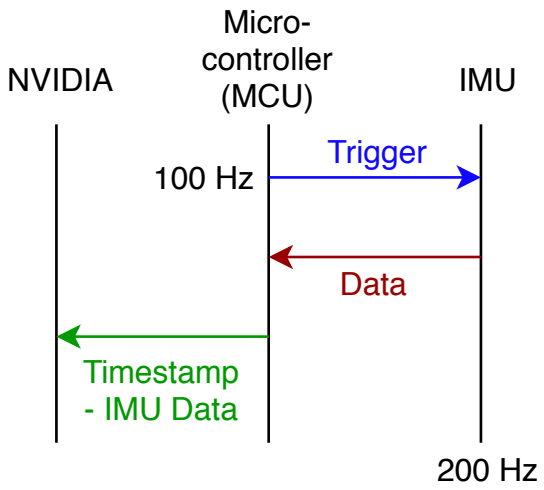


Figure 5.18: Illustration of communication and utilized signals for the IMU.

(laptop or embedded system). To configure and manipulate the camera, the SDK from FLIR was exploited. The configuration engine can be seen at Figure 5.19. The interface implementation was done in C++ with the C++ version of the provided API. Initially, the camera and the communication with the computing device are instantiated and the desired camera is selected by its ID. Afterwards, the acquisition mode is defined, whether it will be in burst mode or in single frame mode. Additionally, the triggering mode is set by choosing between hardware and software triggering. In our case the software triggering is used and the frames are requested one by one, and not in burst mode. The

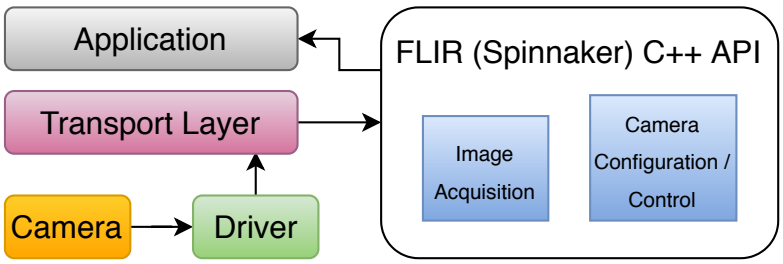


Figure 5.19: Machine vision camera configuration engine.

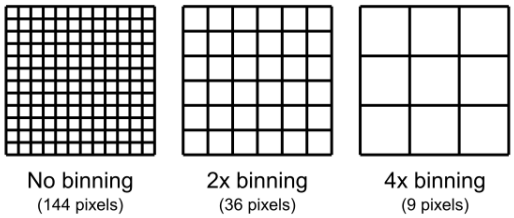


Figure 5.20: Binning examples.

option of software triggering is preferred in this case, because it provides more detailed control by sending the trigger in varying desired time instances. Further configurations that are set, are the resolution of the camera, the FPS and the binning. The binning is a very useful option, during which a high resolution can be reduced in hardware, by merging the information of neighbor pixels. To make things more clear an example is presented is Figure 5.20. Finally, the captured image is requested with a signal that is sent by the computing device. The image arrives in raw format and it has to be converted in an OpenCV editable format before it can be utilized. A brief flowchart of the interface is presented in Figure 5.21. To save some space the flow of initial and final frame is not depicted, where during the first triggering there is no current frame to process and of course the last frame is processed before terminating the whole process.

At this point an issue with the camera FPS has to be pointed out. While the machine vision camera is capable of acquiring data at 60 FPS, there is a bottleneck in the process of saving the captured images into the embedded system drive. Due to the relatively high resolution of images there is a significant overhead in converting the images to .png and storing them in real-time. For this reason the frames are stored in uncompressed format, in .tiff. However, the drawback with .tiffs is that they are larger files, so due to memory limitations it was still not possible to acquire data at high frame rates. For these reasons the camera was set to capture images at 9 fps. A possible solution would be to use a smaller resolution, which was attempted and resulted in 19 fps. But the higher resolution is necessary to acquire a high quality images of a utilized custom pattern to estimate the groundtruth trajectory, something that will be further explained in the next sections. By exploiting the Jetson Xavier, which is more powerful, in the future it will be possible to store faster data into the drive. Additionally, it should be highlighted that during real-time operations the overhead of storing images will be eliminated, as the

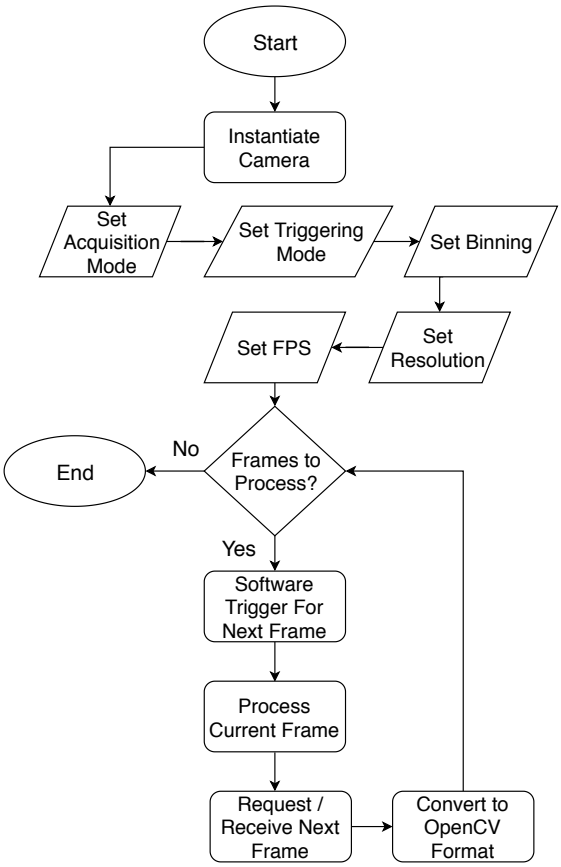


Figure 5.21: Flow chart of camera interface.

images will be directly used as input in the VIO algorithm. So with these in mind, it is evident that the whole system is capable of operating eventually at higher FPS. Despite the limitation in capturing rate, the 9 fps are still adequate to evaluate the proposed system and the examined VIO algorithms.

5.7.3 Improvements on Time-stamping System

To fuse the camera with an IMU and to interface them with the selected embedded system, there was the need for a sensor fusion architecture. At the time of this thesis, such a system was in the stages of prototyping in-house, however it was meant to be used for other computer vision applications. Hence, it was decided to evaluate this system and to examine if exploiting it for a VIO implementation would be feasible. During the testing of this system along with the rest of the VIO implementation some challenges were faced, regarding the hardware setup. While evaluating the robustness of the hardware setup by acquiring some visual inertial data, the loss of some camera frames timestamps was observed. The utilized communication protocol between the embedded device and the micro-controller feature some combinations of characters that indicate the beginning and the end of a message. These combinations rarely appeared also in the contents of

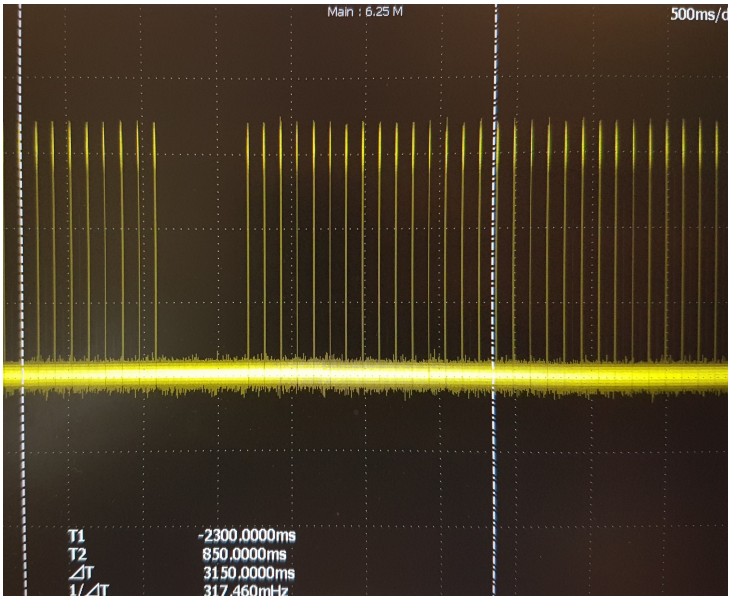


Figure 5.22: Missed pulses of the camera GPIO depicted on the oscilloscope.

the message's header (and not only in the beginning or the end), so faulty messages were parsed. As a result timestamps were not processed properly and they were lost. The next step was to check the hardware side. For this an oscilloscope was utilized to check for lost signals. As it is illustrated in Figure 5.22 some pulses sent from the camera GPIO to the micro-controller sometimes were lost. Furthermore, some images were received corrupted by the embedded system. After experimentation it was proved that this was due to the long cables that were used in the prototype and due to the intense movements conducted during data acquisition. Post-processing was required to clean the datasets from faulty data. For this to be achieved the system was modified, so that the micro-controller transmits along with the timestamps also the corresponding frame ID. These issues were solved during the final hardware setup, which was sealed in a waterproof housing, with the connections being steady and small in length.

5.7.4 System Calibration

Before data acquisition it was necessary to calibrate the IMU, the camera, and the miss-alignment between them. For the calibration of the IMU various approaches were assessed and finally the *kalibr_allan* was utilized, which is available on Github[6]. To calibrate the IMU, it was required to acquire data with the IMU completely stable for about 4 hours. The data were recorded in a ROS bag format, which was afterward used as input to the *kalibr_allan* tool. The *kalibr_allan* after running for about half an hour produced the Allan graphs for both, the accelerometer and the gyroscope. The estimated *Random Walk* values were taken from the Allan graphs, while the *White Noise* values were derived from the IMU datasheet. The exact values of these parameters are

[6] `https://github.com/rpng/kalibr_allan`

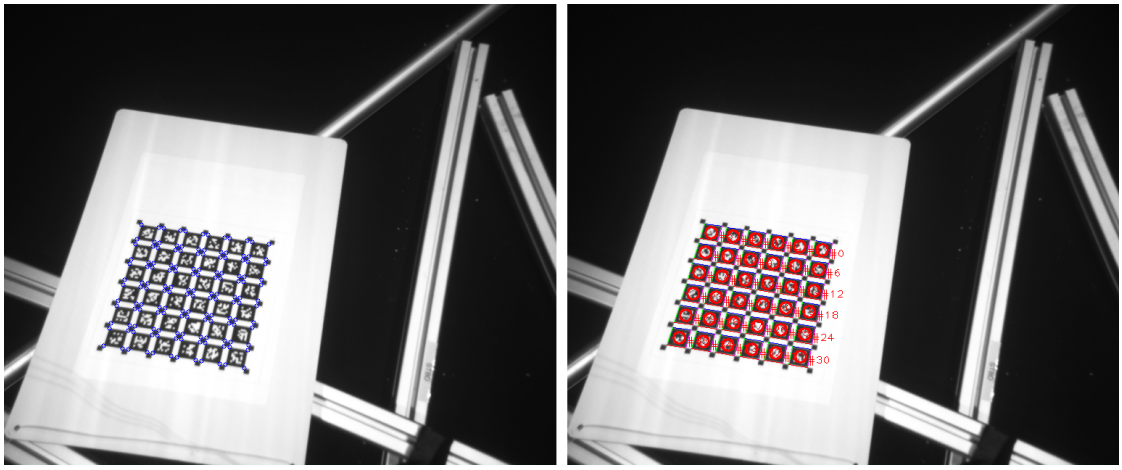


Figure 5.23: Representative frames during intrinsic camera calibration in water.

mentioned in the Section 4.2. It is worth mentioning that these parameters caused some confusion during the research. There is not a standard and common reference for these parameters. Sometimes the *White Noise* and the *Random Walk* are mentioned with the same exact term, or even worse one might be mentioned with the term of the other. Additionally, they can be seen in the literature and in the datasheets with various measurement units. These IMU parameters have to be very precise, because they are important and required not only for the VIO algorithms, but also for the hand-eye calibration between the camera and the IMU.

Similar to the IMU the camera had to be calibrated as well to determine its intrinsic matrix and the distortion coefficients. The camera calibration was done along with the hand-eye calibration, between the camera and the IMU, by utilizing the *kalibr* implementation, which is available on Github[7]. The estimated hand-eye calibration is the same for all the conducted experiments, regarding air and water, because the camera and the IMU are rigidly mounted on a 3D printed body. On the other hand, the intrinsic calibration of the camera had to be computed on air and in water as well, due to the additional distortion that the water causes on the acquired images. For this calibration the pattern illustrated in Figure 5.23 was placed steadily in a location, while the camera was moving in front of it in various angles and translations. The blue and red markers denote the recognition of parts of the pattern in real-time during the calibration. Here it has to be noted that the outcome produced by *kalibr* for the intrinsic camera calibration was compared to the one that was produced by a robust in-house method. The results showed that the *kalibr* algorithm had great performance for both, the air and water, as it produced a calibration very similar to the in-house one.

With the precision of calibration being very significant for a proper and accurate algorithm execution, the challenges and the difficulties to get the whole VIO system to perform accurately started becoming obvious. The *kalibr* documentation suggests that good calibration results can be obtained with a camera at 20 FPS and IMU measurements at 200 Hz. However, for purposes mentioned above, the examined VIO system features

[7] `https://github.com/ethz-asl/kalibr`

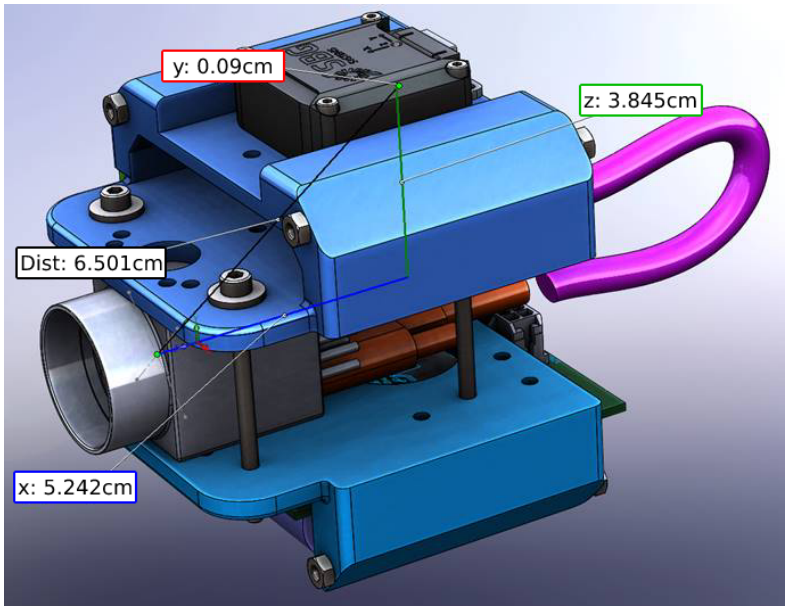


Figure 5.24: 3D CAD model of the body that rigidly accommodates the camera and the IMU. (Property of Fugro)

a camera at 9 FPS and IMU measurements at 100 Hz, essentially acquiring half of the suggested amount of data. Furthermore, the documentation of the IMU does not clarify what is the exact behavior of the sensor when the measurements are sub-sampled, i.e. it is ambiguous if at 100 Hz only the last acquired measurements are utilized or also the intermediate skipped values are pre-integrated to the utilized ones. So here follows the impact of the reduced frame rates that were used. The output of the hand-eye calibration is a 4×4 matrix, with the upper left 3×3 one being the rotation matrix and the remaining 3×1 vector being the translation one. The only way to evaluate in practice the accuracy of the resulted hand-eye calibration was to examine the translation vector, which provides information about the actual distance between the camera and the IMU in the 3 axis. The resulted translation vector provided different translation values from the actual ones. More precisely the distance between the camera and the IMU was calculated to be in x axis: 4.643229 cm, in y axis: 1.408808 cm, and in z axis: 4.851676 cm, while the actual translation values can be seen in the 3D CAD model in Figure 5.24. To examine how much these errors affect the VIO algorithm performance, an experiment was conducted with the initial hand-eye calibration provided by *kalibr*, and another one with the translation vector replaced by the actual values shown in the 3D CAD model. The results confirmed how important the precision is for calibration, as the experiment conducted with the actual translation values produced much more accurate trajectory estimation. Hence, while the rotation matrix was kept as it was provided by the calibration algorithm, the translation vector was replaced with the actual distances between the camera and the IMU, resulting in the hand-eye calibration defined in the matrix 3.2, which was used for all the conducted experiments.

5.7.5 Air Experiments

The implemented VIO system was initially tested on air, specifically in an office, with a moderate number of features, as part of the scene is a white wall. Additionally, it has to be pointed out that during the experiment a low exposure was set, so that the scene gets darker and more challenging. Moreover, a person passes in front of the camera, causing significant noise to the motion estimation algorithm, as it is a moving part of the scene. The conducted trajectory is approximately 7.65 meters long. In contrast to other examined datasets, the camera is wandering around the same scene, observing the same features, simulating in that way a station-keeping case. More details about the dataset can be found at Section 3.5.

Before proceeding to the results it is important to mention how the groundtruth trajectory was estimated. Throughout the whole experiment the camera contains in its field of view a custom pattern, which features specific geometric shapes with known distances between them. By utilizing a computer vision algorithm the pose of the camera against this pattern is estimated for every frame. In that way the groundtruth trajectory was calculated with high accuracy. It is significant to note that since the camera pose is estimated with relevance to the pattern, the resulting coordinates are different from the ones that would be estimated if the camera pose was calculated in relevance to the world frame. On the other hand the VIO algorithms estimate the traversed trajectory against their own arbitrary coordinate frame and with reference to the initial camera position. This practically means that the groundtruth trajectory and the estimated one will have similar shape, but different coordinates, since they are computed in different reference frames. Hence, to compare the estimated and the groundtruth trajectories, they are aligned with the SE(3) method.

Figure 5.25 illustrates the results of executing ORB-SLAM2 and Visual-Inertial ORB-SLAM along with the proposed VIO architecture. In this case the scale of the ORB-SLAM2 estimated trajectory was not aligned with the groundtruth to highlight the actual difference of using just a camera to estimate a trajectory in contrast to utilizing a camera along with an IMU. At the first glance the well known issue of using just one camera is pretty evident, as the estimated trajectory diverges a lot from the groundtruth as far as the scale is concerned. However, the most interesting outcome is the actual trajectory estimation and not its scale. The vision-only algorithm estimated faulty a significant part of the path that the camera traversed. Even with scale alignment the deviation from the groundtruth trajectory would be notable. On the other hand, the Visual-Inertial ORB-SLAM not only estimates very accurately the shape of the traversed path, but also calculates precisely the scale. Here we observe the significance of augmenting a monocular vision-only algorithm with an IMU. The IMU addition provides a complete motion estimation system, without the need of post-processing to estimate the scale. Moreover, the IMU measurements contribute significantly in the motion estimation accuracy and compensate for the drift introduced by the motion estimated only from vision.

In Table 5.6 the APE statistics are presented. Despite the non-ideal hand-eye calibration and the relatively low camera and IMU frame rates, the Visual-Inertial ORB-SLAM exhibits very good performance in this case, with a mean APE around 1.78 cm. The low camera and IMU frame rates do not have a significant negative impact on the trajectory

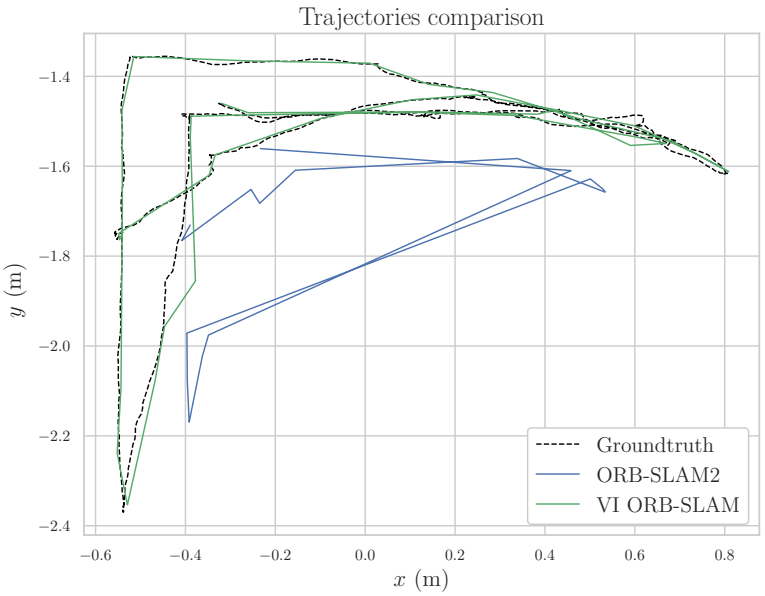


Figure 5.25: Trajectories comparison using ORB-SLAM2 and Visual-Inertial ORB-SLAM on the Fugro air dataset.

estimation, most probably because the motion was quite smooth, with no shocks, not excessive rotations and no motion blur. In Figure 5.26a it is obvious that the translations are estimated quite precisely in all the axis, with negligible deviations. In Figure 5.26b it is observed that although the estimated rotations do not capture in high detail the actual values, they generally follow adequately the trend of the groundtruth. The estimated Roll has also the same trend as the groundtruth, however since the alignment is done by considering the translations, the rotations values are not always aligned. So the estimated Roll is essentially similar to the groundtruth, however its values refer to a different reference frame. Furthermore, while in Figure 5.25 the trajectories are aligned and referenced to the groundtruth data reference frame, in Figure 5.26a they are referenced to the Visual-Inertial ORB-SLAM frame for illustration purposes, to facilitate the reader to visualize the traversed trajectory in a coordinate frame more common to the human perception.

The rest of the three VIO algorithms were deployed and tested on this dataset, however they did not perform successfully. VINS-Fusion, OKVIS and ROVIO are sensor fusion implementations with a lot of configuration parameters for the IMU on top of the VO part. Some of these parameters can be derived during hand-eye calibration between the camera and the IMU and others during the IMU calibration. The estimated parameters from the conducted calibrations were utilized. However, these three VIO algorithms did not function as expected, i.e. they quickly and significantly diverged, resulting to unreasonable motion estimations. While investigating the root of the problem, an important outcome was that these implementations are very sensitive in relevance to the accuracy of these IMU parameters. A small change in these values would result in a rapid divergence or would lead to a smooth motion estimation for a small amount of

Table 5.6: APE Statistics for Visual-Inertial ORB-SLAM for the Fugro Air Experiment

	Min	**Mean**	**Max**	**Std**
APE (m)	0.002769	0.017848	0.058837	0.014841

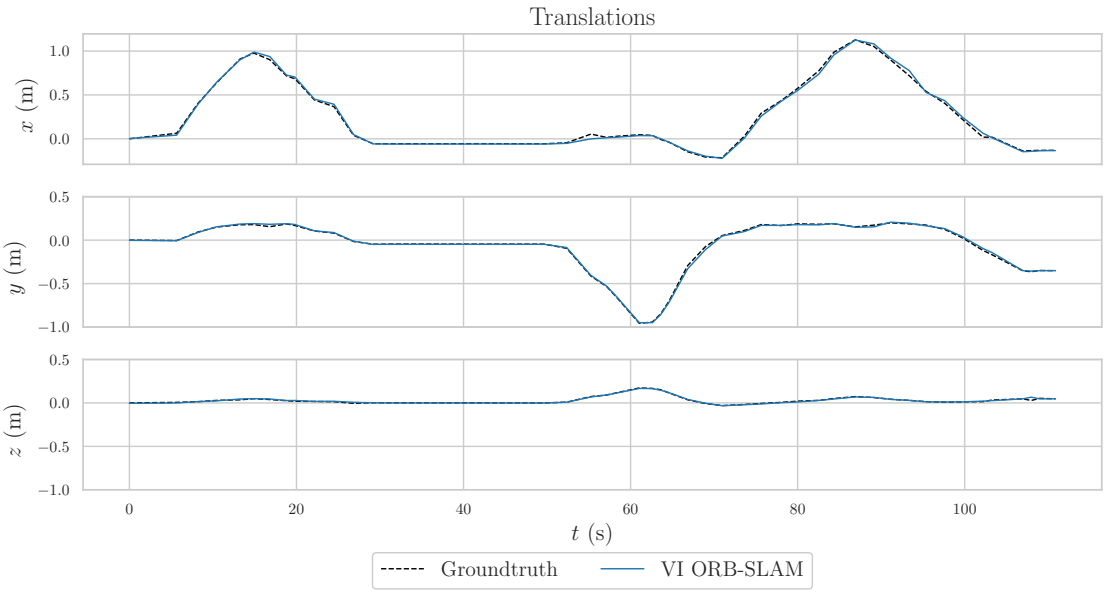
time, before they completely fail. It might be the case that for this experiment these algorithms failed due to the rough estimation of the hand-eye calibration or due to the relatively low camera and IMU frame rates. However, it has to be reminded that these algorithms failed also in the case of the Aqualoc dataset. Compared to the EuRoC drone dataset, with which these algorithms were executed successfully, the differences in our system are the accuracy of the hand-eye calibration, the frame rates and the quality of the synchronization (further analyzed in the next section).

5.7.6 Water Experiments

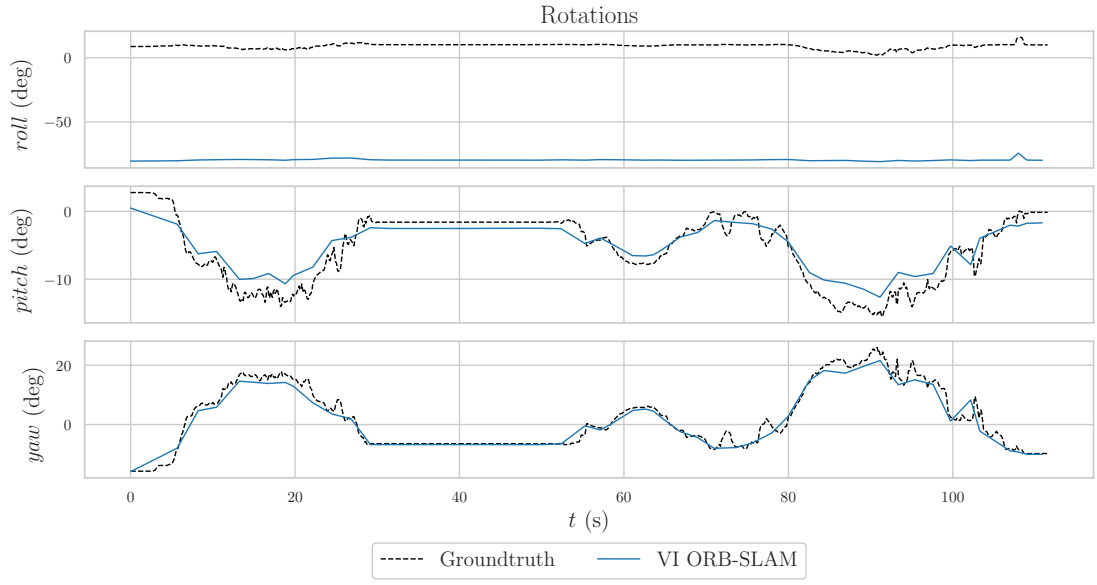
Since the examined VIO system performed well in the air experiments along with the Visual-Inertial ORB-SLAM, the next step was to test it underwater and determine the impact of the underwater environment on the system's performance. Conducting experiments in underwater environment is quite challenging, because it requires time, money, specialized staff and expensive equipment. On top of that, it is even harder to execute experiments near a structured underwater environment, like underwater Oil & Gas infrastructures, due to the risks and difficulties to approach such kind of areas. To evaluate the proposed VIO system a structured underwater environment was created in a tank. Some piles and 3D structures were strategically sank into the tank, so to simulate an underwater environment with visual features in the vertical axis, and not only on the seabed.

The whole VIO system hardware, including the camera, the IMU and the embedded platform, had to be isolated against water. Hence, the hardware was placed in a waterproof housing. The housing was made of metal for better heat abduction and to avoid ventilation deployment. Two worth mentioning issues came up at the step of sealing the hardware. The first one is related to the leakage test that was performed on the housing. To examine if the housing was prone to water leakage, pressure was removed from its inner part. The tricky part was that between the camera body and its mounted lens there was a nearly isolated cavity, including a rubber gasket. Since the inner space of the housing had a pressure of less than 1 atm and the cavity contained pressure of 1 atm, the phenomenon of gas discharge took place. The only way of discharge was to deform the gasket, which by its turn blocked the field of view of the camera, resulting in an image with a huge black area. The second issue was a white dot in the field of view, caused by a reflection which initially was considered to be produced by external lights placed over the tank. However, it is interesting that the source of reflection was eventually the hardware led lights that were reflecting on the front glass of the housing and back to the camera's lens. This issue was faced by placing a piece of foam between the camera lens and the hardware, that was placed in the back of the camera, preventing the light to reach the front part of the housing.

With the technical issues solved, several experiments were executed in the tank to



(a)



(b)

Figure 5.26: Estimated (a) translations and (b) rotations, compared with the groundtruth for the Fugro air dataset.

evaluate the performance of the proposed hardware and of the examined algorithm underwater. Since the experiments were conducted in a 4×4 m tank the freedom of movements and the diversity of routes that could be followed were limited. Furthermore, the ROV was controlled via a game console joystick, and the precise manipulation was challenging.

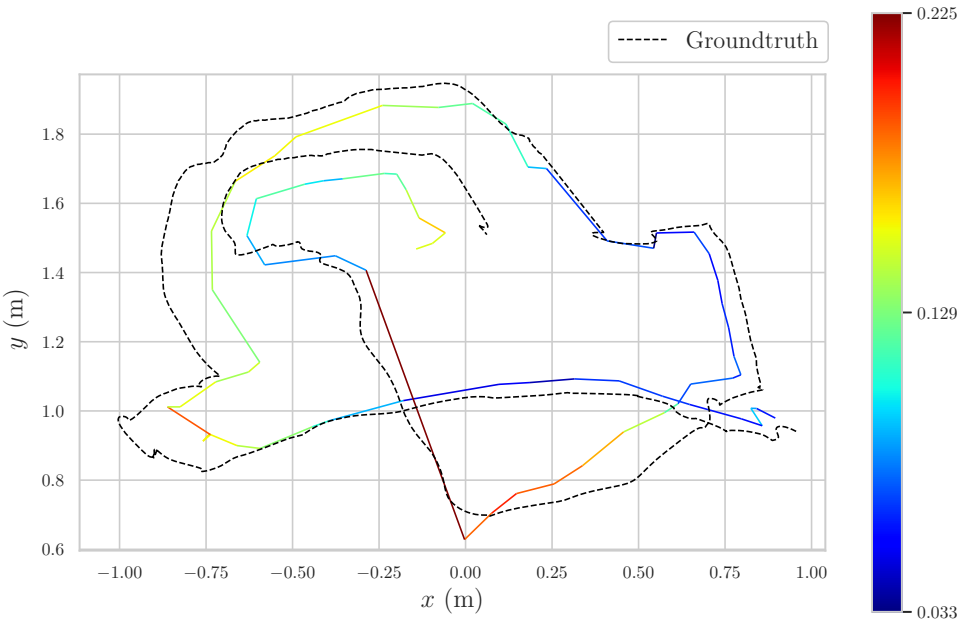
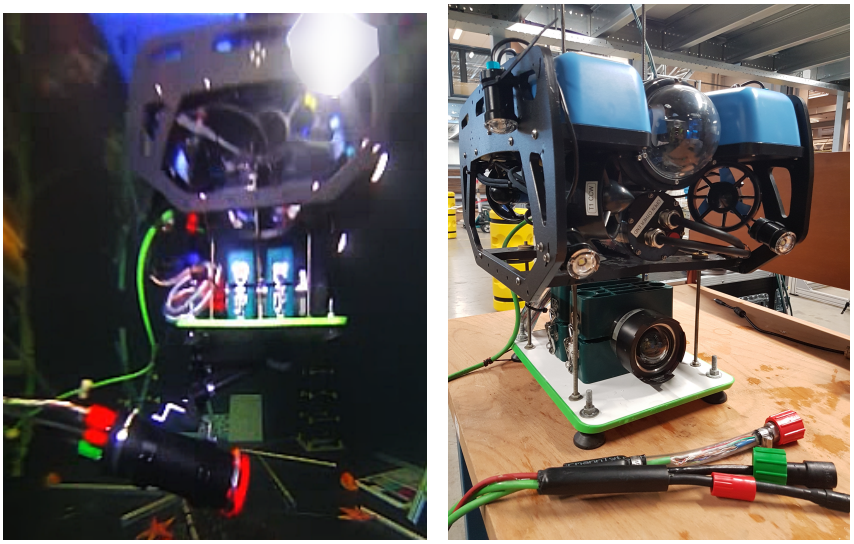


Figure 5.27: Estimated trajectory compared with the groundtruth for the Fugro underwater experiment, with APE information. The colored bar and the colors mapped on the estimated trajectory indicate the level of the APE in meters (m).

Hence, most of the traversed routes were similar, with alike motions, and one of the most representative trajectories is presented and analyzed in the results. Figure 5.27 illustrates the estimated traversed trajectory compared with the groundtruth. The robot traveled in total for approximately 10.5 m. The groundtruth trajectory was estimated with the same method that was described in the previous section, regarding the experiment in the air. Also in this case, the estimated trajectory and the groundtruth were referenced to different reference frames, so they had to be aligned. Additionally, it has to be noted that although in the air experiments the results of various executions of Visual-Inertial ORB-SLAM were approximately the same with very good trajectory estimations, in the underwater case the resulting trajectory sometimes differed significantly. The presented trajectory is one of the best cases and the non-deterministic behavior of the algorithm has to be taken into account.

In Figure 5.27 it is observed that while the estimated trajectory in general follows the shape of the groundtruth, parts of it deviate notably. More precisely, it seems that the scale drifts during the traversed path. This contrasts the air experiments, where the trajectory was estimated with high accuracy. To interpret this result it is useful to also have in mind the structure of the ROV during the underwater experiments. In Figure 5.28a a picture of the ROV in the tank is presented. The first part that is mounted under the ROV should be ignored. The VIO system is hanging in the bottom of the ROV with a mechanical arm. The VIO system was mounted in that way so that the camera can have a field of view that includes features in front of the ROV and under it.



(a) (b)

Figure 5.28: (a) The utilized ROV with the VIO system mounted under it. Picture taken underwater during the operation. (b) A more clear picture of the utilized ROV out of the water and without the VIO system mounted on it.

While the most common approaches for ROVs feature a downward looking camera, here it is examined and proposed a different camera orientation. Since the ROV is aimed to operate in underwater areas with man-made installations it is useful to capture features not only from the seabed, but also from the structures in front of the robot. This ROV structure resulted in undesirable effects during the experiment. Whenever the robot moved and stopped the bottom of the ROV, that includes the VIO system, oscillated and acted like a pendulum.

On top of the described phenomenon, it is important to mention an inherent limitation of monocular approaches. Monocular systems struggle to deal with pure rotations, especially in un-mapped areas. It is essential for the robot to move is such a way that the translation is more notable than the rotation, so that high parallax is achieved and new 3D points can be triangulated from different positions. For the monocular cases, under degenerate motion, like constant velocity motion with no acceleration excitation, the scale can not be observed. In such cases the system falls back to monocular vision-only approach, so the scale-drift issue by accumulated errors is difficult to be avoided. The pendulum phenomenon caused small movements with no significant translation and acceleration excitation. In addition, the difficulty to manipulate with accuracy the ROV and its delayed motion response due to water resulted into loosing the pattern, which was used to derive the groundtruth, from the field of view. The repeated attempts to keep the pattern in the field of view led unavoidably to pure rotations. This behavior was quite different from the one observed in the air experiment, during which the motion was smooth and rotations were accompanied with notable translation. In contrast to the prototype design that was used in the experiments, the ROVs that will be used in the field will feature the VIO system mounted closely to their main frame, so the motion

will be smoother with not so many angular movements. One more point that it has to be taken into account when the air and underwater results are compared, is that in the air experiment the features were better distributed throughout the scene, while in the tank the features were concentrated in specific areas of the scene and regularly in the center of the image, which leads to bad motion estimation.

Attempting to further interpret the results and to determine the source of bad accuracy in trajectory estimation, the examined VIO system is compared with the case of the initially examined drone dataset and its VIO design, which performs very well. The first difference between these two cases is the features distribution across the scene. For the drone case throughout the whole trajectory the features are well distributed and there is a balance in the way they are spread in the field of view. As it was already mentioned, in the underwater experiment this is not the case, in which the features are not evenly distributed and many times they are concentrated in the center of the image, making it difficult to estimate especially the rotations. The second significant point of difference is the utilized frame rates. The drone dataset is considered to be one of the best with high quality acquired data. While they utilize their camera at 20 fps and their IMU at 200 Hz, in our case the camera operates at 9 fps and the IMU at 100 Hz. Essentially, in our case we acquire 50% less information compare to the drone dataset. The third point has already been mentioned and it is the hand-eye calibration between the camera and the IMU. As it has been analyzed, the relatively low frame rates resulted in a non-ideal calibration, which was manually refined. Due to the challenging nature of the underwater experiment and the accumulated errors from other sources, the calibration might had a negative impact, further enhancing these errors. The last and very important point is the synchronization that takes place in the proposed VIO system. Precise sensor synchronization is as challenging as important. Figure 5.29 illustrates various synchronization conditions. The examined VIO hardware stands somewhere between the sub-optimal and worst case, because camera FPS sometimes deviates due to Central Processing Unit (CPU) task scheduling issues and due to frames/timestamps losses. One more point to be mentioned on this domain is that the drone dataset timestamps have accuracy of nanoseconds, whereas in our case the timestamps have accuracy of milliseconds, which might play a role in precise sensors synchronization. The interesting part of this analysis is that the aforementioned issues did not have a negative impact during our experiment in the air. Most probably this was due to the kind of motions that took place in the air experiment, where the movements were quite smooth. On the other hand, during the underwater experiment the ROV was oscillating frequently and went through some instant intense motion changes, which in combination with the low frame rates probably resulted in bad motion estimation, because the rapid movements could not be captured.

In Table 5.7 the APE statistics are presented for the underwater experiment. The mean APE is around 11.5 cm. Even though the value is notably worse compared to the air experiment, for possible reasons that were analyzed above, it is still a very good accuracy for many underwater operations that do not require high precision in trajectory estimation. In fact the achieved accuracy is equal and even better compared to traditional acoustic positioning methods that their accuracy might be far more worse than this. Moreover, if the achieved accuracy is evaluated by taking into account also the

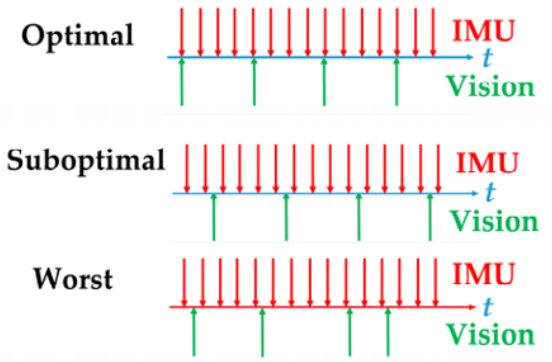


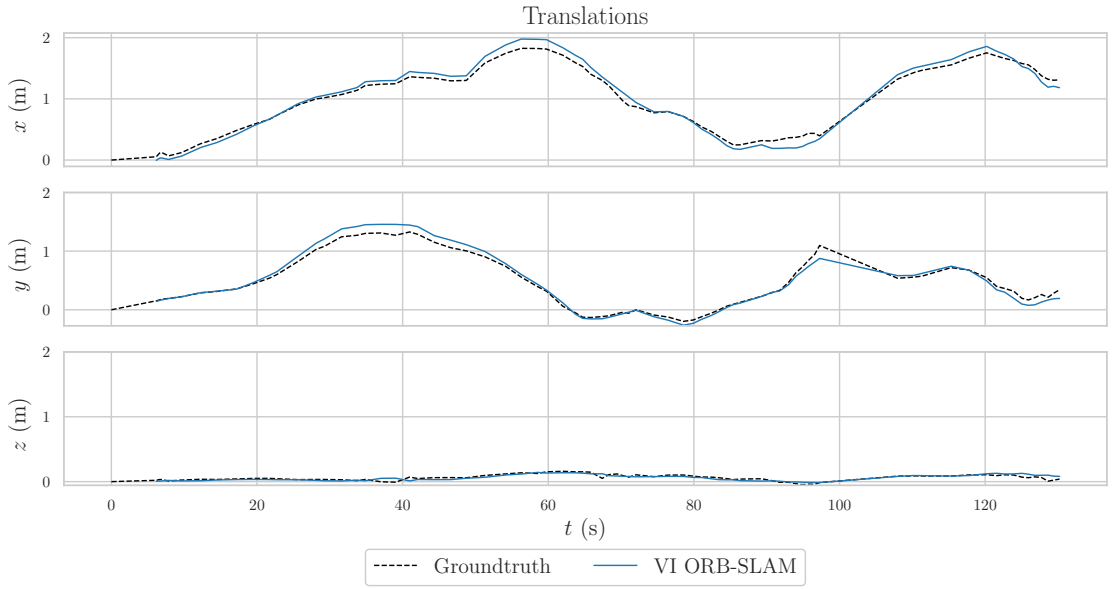
Figure 5.29: Various camera - IMU synchronization conditions. [79]

Table 5.7: APE Statistics for Underwater Tank Experiment

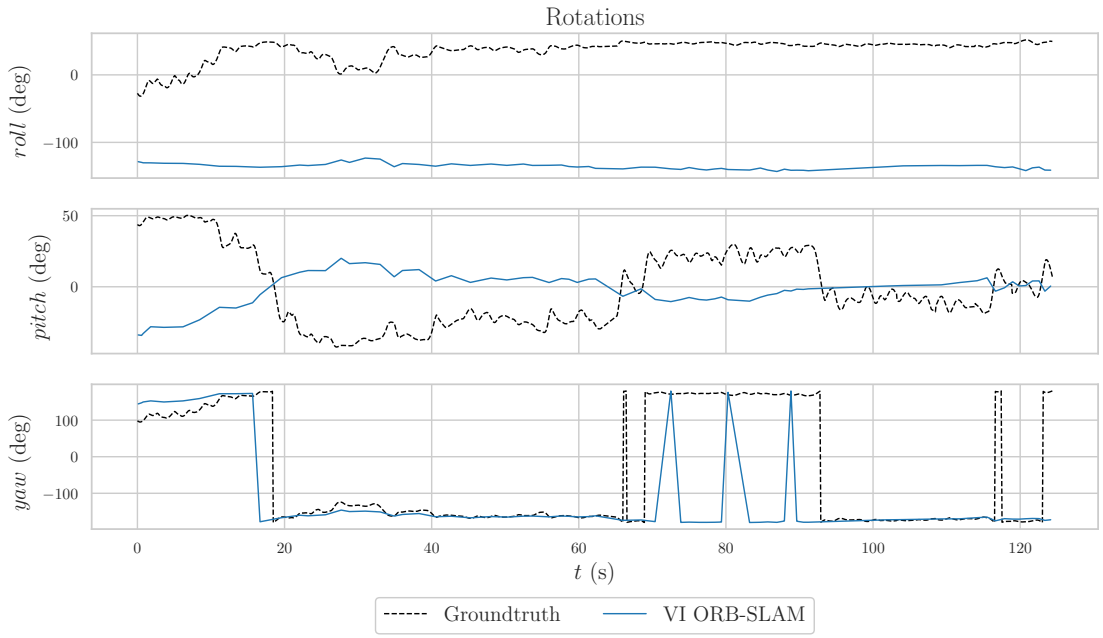
	Min	**Mean**	**Max**	**Std**
APE (m)	0.033373	0.114841	0.225312	0.045878

required costs for trajectory estimation, we would say that the trade-off between accuracy and cost is quite good. Other high precision methods that are utilized for underwater positioning make use of far more expensive equipment than the examined VIO solution, and they require a lot of time and staff to be deployed, resulting in additional costs. In Figure 5.30 the estimated translations and rotations are illustrated in more detail and they are compared with the groundtruth data. It is evident that the estimated translations are calculated quite accurately and they are similar to the groundtruth. On the other hand, the rotations deviate notably from the groundtruth data. While the different values of the estimated and the groundtruth data are reasonable, due to different reference systems, the actual rotations are estimated roughly. This shows that the accumulated errors that lead in a big APE are caused mainly from the miss-estimated rotations. The reasons for these miss-estimated rotations are already analyzed above. It should be noted that in the yaw graph there are some "jumps" between the values of 180 and -180 degrees. This is not a faulty estimation, but it is the result of the notation used to describe a complete circle, i.e. the right part of the circle is described with 0 to 180 degrees, while the left part with 0 to -180 degrees. The values 180 and -180 represent the same point in circle, so slight deviations around these values cause the observed result in the yaw graph.

Finally, two interesting outcomes should be mentioned, that came up by observing the visual output of the Visual-Inertial ORB-SLAM in real-time. The algorithm manages to capture accurately and in detail even the very small motion changes of the ROV. This means that it is a promising approach that could be used for station keeping purposes, by providing the output of the visual-inertial motion estimation as input to a controller responsible for the thrusters manipulation. In addition, the robustness of the algorithm against moving objects was examined. Since the lights of the ROVs attract fish, it is a common phenomenon to observe moving beings that invade in camera's field of view.



(a)



(b)

Figure 5.30: Estimated (a) translations and (b) rotations, compared with the groundtruth for the Fugro underwater dataset.

To evaluate the algorithm's robustness, the ROV was kept still and a pile, included in its field of view, was moved repeatedly. Although the algorithm was initially tracking features on that pile, it stopped tracking them by the time the pile started moving. The

Visual-Inertial ORB-SLAM was capable of identifying the pile's features as outliers by determining the rest of the scene, that included most of the features, to be static. Hence, Visual-Inertial ORB-SLAM exhibits notable performance against distracting moving objects and this makes it a promising approach for underwater environments.

Also in the underwater experiments the other three VIO algorithms were examined, however they failed for the same reasons described in the section with the air experiment.

5.8 Visual-Inertial ORB-SLAM Optimization and Performance Evaluation for Embedded Platform

The research on VIO algorithms resulted in the selection of the Visual-Inertial ORB-SLAM. In the previous section the complete VIO system, including the algorithm, was thoroughly discussed and the performance of the algorithm on the conducted underwater experiments was analyzed. The embedded system that was chosen for the implementation of the VIO system is the latest powerful device from NVIDIA, the Xavier Jetson. The Xavier features also a powerful GPU. Many computer vision algorithms are implemented in such a way that they can be optimized on a GPU. To exploit as much as possible the embedded device and examine how much the performance of the VIO system can be increased, an investigation was conducted to determine if parts of the algorithm can be accelerated on the GPU. An accelerated version of the ORB-SLAM2, with promising performance, was already implemented, so the accelerated computer vision parts of the algorithm were embedded in the Visual-Inertial ORB-SLAM algorithm. This resulted in an accelerated version of the Visual-Inertial ORB-SLAM that utilizes the GPU along with the CPU. The accelerated computer vision parts that were utilized can be found in the accelerated version of ORB-SLAM2, which is available online[8]. The last two presented experiments (in air and underwater) with the examined VIO system were conducted by utilizing the accelerated version of the Visual-Inertial ORB-SLAM. While in the previous section the results regarded the performance related to VIO, in this section the algorithm's performance on the embedded device is analyzed.

The Xavier Jetson can be operated in various custom power modes/profiles, which come with different configurations regarding the number of online CPU cores, and CPU, GPU, RAM frequencies. The standard profiles are presented in the Table 5.8. While analyzing the results it is important to keep in mind the combination of the varying values that each power mode is configured with. It should be clarified that the indication of Watts in the column "Mode" is just a budget of power that the profile is offering, i.e. it is the upper power limit that can be provided, while the actual power consumption is presented in the forthcoming graphs. The MAXN mode is not limited with an upper power threshold, however an investigation on this suggests that the power consumption might fluctuate around 50 Watts.

The most critical part of the Visual-Inertial ORB-SLAM to be executed in real-time is the tracking thread, which processes every new incoming frame. The bottlenecks in this thread are the well-known processing power demanding tasks relevant to features' detection and extraction. The latest versions of OpenCV provide a number of features

[8] `https://github.com/yunchih/ORB-SLAM2-GPU2016-final`

Table 5.8: NVIDIA Xavier Jetson Power Modes

Mode	Online CPUs	CPU Max Freq (MHz)	GPU Max Freq (MHz)	Mem Max Freq (MHz)
15W	4	1200	670	1333
30W-2Cores	2	2100	900	1600
30W-4Cores	4	1780	900	1600
30W-6Cores	6	1450	900	1600
30W-ALL	8	1200	900	1600
MAXN	8	2265.6	1377	2133

Table 5.9: Visual-Inertial ORB-SLAM Parameters

Num of Features	Scale Factor	Pyramid Levels	Fast Thresholds
3500	1.2	8	7 & 18

that contribute in the acceleration of the algorithm, such as the accelerated implementation of the algorithms for FAST features detection, for ORB feature extraction, and for the Gaussian Blur Filter. In addition, the CUDA GpuMat type is utilized for more efficient image manipulation.

To evaluate the impact of the algorithm acceleration and to compare the sequential with the accelerated version, a moderate configuration for the algorithm parameters was used, presented in Table 5.9. Before proceeding with the analysis, the meaning of the asterisks accompanying two of the power modes in the following graphs should be clarified. The * indicates that for the specific parameters configuration the sequential version could not keep track and the algorithm was failing. This was due to the inadequate computational resources. For a coherent illustration of data, the sequential measurements are still presented, however they were acquired by playing the ROS bag in a slower speed, specifically at 30% of the original. The playback speed had to be reduced that much for the sequential to be successfully executed. By playing the ROS bag in a slower speed a slower frame rate of incoming data to the algorithm is achieved, so the algorithm manages to handle them and to not fail. The ** indicates that for the specific parameters configuration both, the sequential and the accelerated versions, failed to keep track of the motion throughout the experiment. To acquire data and for the algorithm to successfully keep track of the motion throughout the whole experiment the ROS bag playback speed was reduced to 50% and 80% of the original speed for the sequential and accelerated version correspondingly. However, it has to be pointed out that the amount of features and the image resolution used are quite bigger than the common ones or the required ones for proper results.

Figure 5.31a illustrates in detail the average FPS achieved for every NVIDIA power profile, for both the sequential and the accelerated versions, while Figure 5.31b shows the actual speedup. The speedup is defined as the execution time of the sequential algorithm over the execution time of the accelerated one:

$$Speedup = \frac{t_{sequential}}{t_{accelerated}} \tag{5.3}$$

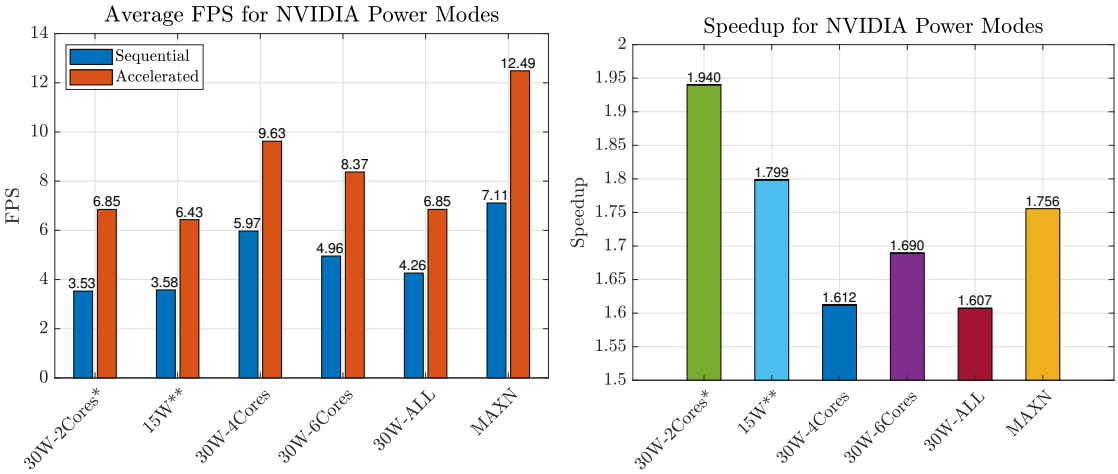
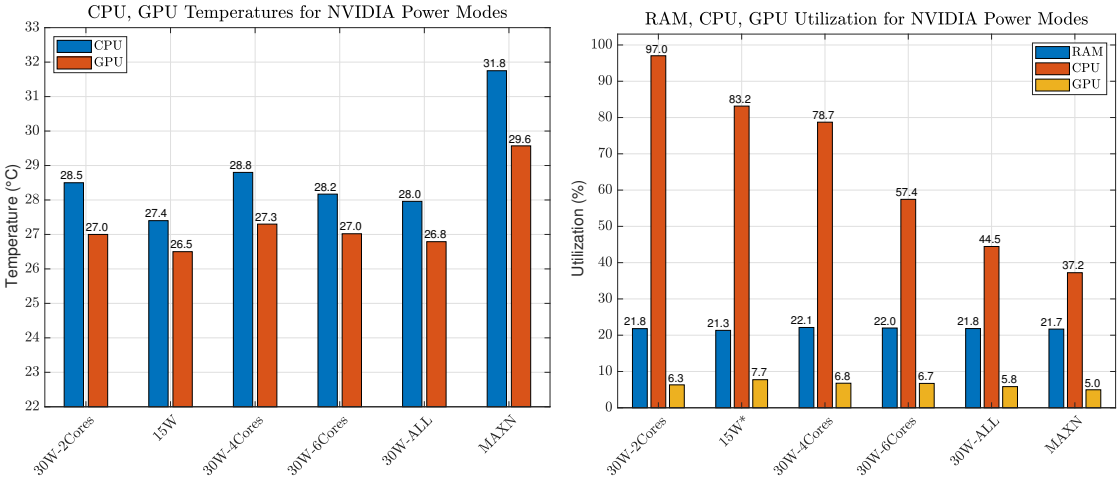


Figure 5.31: (a) Average FPS of the sequential and accelerated algorithm versions for the NVIDIA power modes. (b) Speedup for the NVIDIA power modes.

The speedup for each power mode regards the sequential and accelerated performance of the algorithm for the specific mode. While the contribution of the GPU to enhance the overall performance of the algorithm is quite obvious and notable comparing the FPS achieved by the sequential and the accelerated versions, the most interesting result regards the 30W-2Cores power mode. Although it is evident that for this mode the speedup is the biggest one, in fact it is even bigger and the contribution of the GPU vital. This speedup has been derived by considering the average FPS for the sequential version, which was acquired by playing the ROS bag in a slower than the actual speed. This means that the sequential version was not capable of performing in real-time, however the accelerated version was executed successfully. The mode 30W-2Cores introduces the biggest challenge for a successful algorithm execution, as the Visual-Inertial ORB-SLAM uses 3 parallel threads and this mode provides only 2 CPU cores. Most probably two of the threads are executed in the same CPU core, while the GPU contributes significantly into reducing the amount of processing and computations that the CPU is required to accomplish. Hence, the accelerated version makes it possible for the algorithm to be executed even with 2 available CPU cores. The rest of the presented speedups are relevant to the various combinations of RAM, CPU and GPU frequencies. Another observation that should be highlighted is that eventually the RAM, CPU, GPU frequency is more critical than the number of online CPU cores. This occurs by the fact that although the 15W power mode provides more online CPU cores, the decreased frequencies make it impossible for the algorithm to be executed with success. Moreover, it is worth mentioning that in the case of the accelerated version, even the worst performance (6.85 FPS) is fairly adequate for the underwater robots that move smoothly and with low speed.

The rest of the figures in this section refer to the accelerated version of the algorithm. Figure 5.32a depicts the CPU and GPU average temperatures for the NVIDIA power



(a) (b)

Figure 5.32: (a) CPU and GPU average temperatures for the NVIDIA power modes. (b) RAM, CPU and GPU average utilization percentage for the NVIDIA power modes.

modes. Since the CPU does most of the processing and also handles other irrelevant to the algorithm tasks, exhibits higher temperatures. However, the maximum temperatures for both, the CPU and GPU, fluctuate in reasonable and manageable values and they are not a limiting factor. In addition, for all the modes except the MAXN the temperatures are quite similar. Figure 5.32b shows the average utilization percentage of the RAM, CPU and GPU for the NVIDIA power modes. As it was expected the percentage of memory used in all the power modes is similar and its frequency does not seem to have an impact on it. Since the GPU is responsible for only a small part of the algorithm, its utilization is notably low and similar for all the cases. However, it is observed that the higher utilization takes place in the 15W mode, in which the GPU operates with the lowest frequency, while the lowest utilization corresponds to the MAXN mode, as the GPU functions with the highest frequency. The small GPU utilization percentages suggest that there is a significant amount of additional computational resources that can be exploited to accelerate even more parts of the algorithm. Finally, we notice that for the 30W-2Cores mode the CPU is almost entirely utilized, as the three parallel threads are executed in only 2 CPU cores. This indicates that for this case the system was performing at its limits. For the rest of the power modes the CPU utilization percentage decreases along with the increasing frequency and amount of available cores, as more power remains unexploited. This information is very interesting, as a power mode should be selected by considering the desired spare amount of CPU resources and the extra tasks that are needed to be handled in parallel with the algorithm.

Figure 5.33a presents the average total power that the embedded device consumed for the NVIDIA power modes. The total consumption is calculated by the formula in equation 5.4,

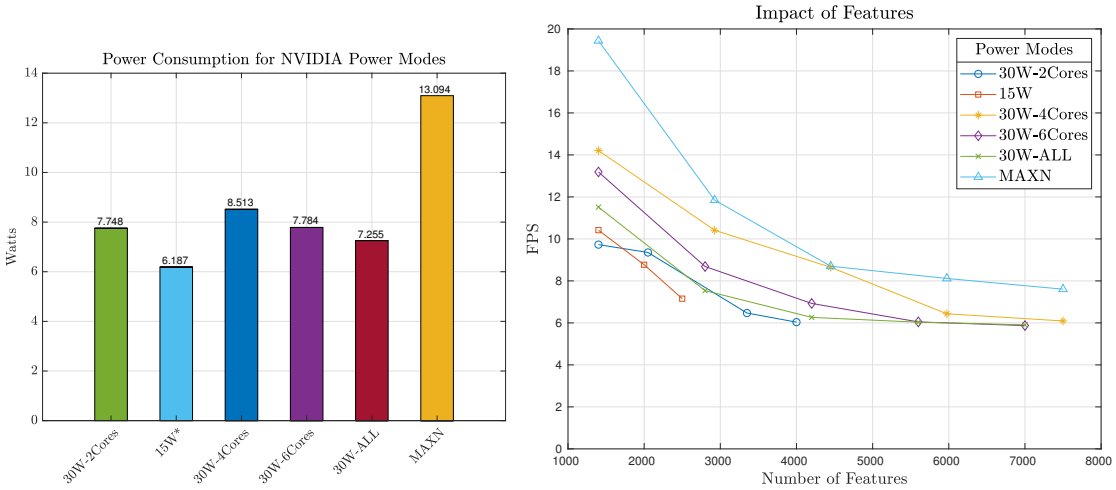
$$Total\ Consumption = GPU + CPU + SOC + VDDRQ + SYS5V \tag{5.4}$$

where GPU denotes the GPU power consumption, CPU the CPU power consumption,

SOC the System on Chip power consumption, VDDRQ the DDR power consumption, and SYS5V the system 5V power consumption. The SOC variable includes power consumption for units like ISP (Image Signal Processor), internal busses, encoders/decoders and controllers. From the average power consumption values in the figure it is evident that the algorithm consumes significantly less power than the maximum available in each mode. Furthermore, the results indicate that the frequency has a stronger impact on the power consumption, compared to the amount of online CPU cores. Apparently, the 30W-2Cores mode, which features high frequencies, consumes more power than the 30W-All mode, which features 8 online cores in lower operating frequency.

Figure 5.33b presents the impact of the number of features on the FPS for each of the NVIDIA power modes. This graph provides a visual comparison and facilitates to determine which of the power mode exhibits higher performance, considering the maximum amount of features. The extreme values define the operating limits of the algorithm for the examined experiment, i.e. the least amount of features required to function properly and the maximum amount that the computational resources of each mode can handle. For the specific experiment, which begins with a moderate robot translation and strong rotations, while the distribution of features along the image is not very good, the least amount of required features to initialize the algorithm was 1400 for all the power modes. In addition, the FPS follow approximately the same trend for all the power modes, with a moderate drop at the beginning and a stabilized tendency towards the end. A possible reason for this is that from an amount of features and on the map is saturated and no new features can be detected. Additionally, as the amount of features increases the number of total keyframes decreases, as new frames do not provide much new information, due to the huge amount of information and map points that are registered in every keyframe. Over a specific amount of features, again, the map is saturated and a certain number of keyframes is needed to represent the map, so the number of keyframes will not be significantly decreased with the increase of features. A notable part of execution time depends on the amount of keyframes registered in the system. Moreover, by observing the graph it is verified again that the operating frequency plays a more important role than the number of online CPU cores. It is apparent that the power modes featuring higher operating frequencies achieve higher FPS. The only exception regards the 30W-2Cores mode which squeezes two of the three parallel threads into a core. Since the algorithm features three parallel threads, a greater amount than 3 cores do not improve the performance. Finally, the 15W mode can handle only up to 2500 features, while the MAXN mode can reach up to 7500. However, during the research for this thesis and the conducted experiments with various datasets, it became obvious that for the common utilized camera resolutions an amount of approximately 3000 features is more than adequate for the algorithm to operate properly.

In Figure 5.34 the evolution of the map size in relevance to the time is depicted for various amounts of features. The map size is computed by the formula 5.1 and by the size of the keyframe and keypoint given by the equations in 5.2. It has to be taken into account that the map size evolution depends on the environment that the robot is operating and on the quantity of new visited places throughout the traversed trajectory. In the examined experiment the robot closed a loop near the end of it and was operating in the same area that was mapped. This means that no matter



(a) (b)

Figure 5.33: (a) Average total power consumption for the NVIDIA power modes. (b) Impact of number of features on the FPS for the NVIDIA power modes.

the amount of time the robot would continue to operate, the size of map would stop increasing. The graph shows that the size grows linearly with the time for all the cases, and near the end of the experiment the map size stops rising. It is interesting that for lower numbers of features the map size follows a steady growth, whereas for bigger features' quantities the map size fluctuates while rising. This is due to the culling mechanism that the algorithm features for discarding redundant information stored in the map. The algorithm generously registers new keyframes and keypoints, however if later some keypoints or keyframes are recognized to provide overlapping information they are discarded. It is reasonable that the culling mechanism is activated more often in cases with more features. It is worth noting that for a moderate quantity of features, like 3500, the map size reaches the 7 MB after approximately 3 minutes of operation and 10.5 m traveled. Assuming that the ROV would be traveling regularly in new places, after a whole day of operation the map size would roughly reach at 3.36 GB. This suggests that the culling mechanism is quite efficient and that the map size is not a limiting factor for underwater operations. Finally, we have to consider that in the examined cases usually the ROVs operate around the same area, so the resulting map size will be negligible no matter the duration of the operation.

5.9 Conclusion

Five of the most recent and popular VO/VIO algorithms were selected to evaluate their performance on the underwater domain. To examine them four underwater datasets that were available online were employed, while an additional underwater dataset was produced in-house, simulating an underwater structured area. The ORB-SLAM2 had remarkable performance on the two sequences of the UWSim simulated dataset. In both of the sequences it managed to keep track throughout the whole experiment and it closed

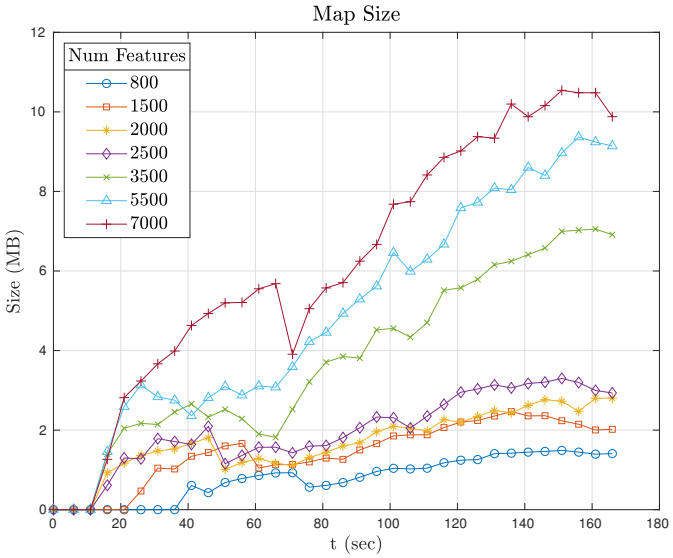


Figure 5.34: Evolution of map size in MB with relevance to the time.

the loops successfully. The drift was limited in very low levels and the APE was below 19 cm. The translations and rotations were adequately estimated for both of the sequences. Moreover, the robustness of the ORB-SLAM2 was tested on four different turbidity levels, managing to keep track and close the loop for the first three. The turbidity of the water had a negative impact on the speed of the algorithm, as more time was required to extract salient features from the noisy scene. The fourth level of turbidity caused the algorithm to lose track, due to the significant delay that was introduced in the process of extracting good features. However, it was proved that if more computational resources were available the algorithm would be able to still extract features despite the noise in the images.

The Aqualoc dataset, which was a challenging one with camera oscillations, provided also IMU measurements and it was used to evaluate the VIO algorithms. With the provided dataset's calibration parameters it was not possible to properly execute the VINS-Fusion, OKVIS and ROVIO algorithms. However, the ORB-SLAM2 and Visual-Inertial ORB-SLAM were deployed successfully and the impact of augmenting a visual algorithm with inertial data was examined. It was observed that most of the drift during motion estimation was caused from the miss-estimation of the rotations. The comparison between ORB-SLAM2 and Visual-Inertial ORB-SLAM showed that enhancing a visual-only algorithm with IMU data results not only to a system that estimates the absolute scale, but also increases the accuracy of the motion estimation. Additionally, it became evident that with proper configuration the inertial enhanced algorithm can achieve the same speed of execution as the visual-only one. However, the algorithm did not manage to recognize places that have been already visited, thus loop closing was not accomplished. Although some trajectories were estimated roughly, in general the APE was quite high. By observing the footage of this dataset it became obvious that mounting strategically the lights on the ROV in relevance to the camera is quite important. A

light source was placed very close to the camera, enhancing in that way the intensity of floating particles and producing additional noise.

The impact of applying image processing filters on the underwater datasets and the affect on the performance of the algorithms was assessed. Various contrast enhancement and noise reduction filters were tested. Although the result was obviously better for the human eye, the performance of the algorithms was decreased. The deployment of such filters cause significant change on the information that surrounds the feature points, essentially making it more difficult to match features from frame to frame.

ORB-SLAM2 failed to keep track throughout the whole Underwater Caves dataset, due to extensive noise and parts that lack of salient features. On the other hand, on the Archaeological dataset the performance of the algorithm was better, managing to keep track during the whole experiment. However, the algorithm did not achieve loop closing and the mean APE was around 1.6 m. Nevertheless, the shape of the estimated trajectory was very similar to the groundtruth. The conducted experiments suggest that the FPS decrease with the increase of the amount of features and the amount of pyramid levels, while the mean APE is not affected. Furthermore, the increase of the scale factor for the pyramid levels results into more FPS. The change of FAST thresholds did not seem to affect the number of FPS.

A complete VIO system was implemented, aiming the underwater areas that are occupied by man-made installations. The system's architecture is comprised by a machine vision camera, an IMU, a micro-controller and an embedded device from NVIDIA. The functionality of the system and the communication / synchronization among the components was analyzed in detail. For the machine vision camera to interact with the rest of the hardware, a low level interface was implemented in C++ and the camera was configured properly. During the experiments with the prototype some flaws were identified regarding the time-stamping system and the whole framework was improved to be more robust for the VIO scope. Various calibration methods were investigated for the IMU and the camera. The *kalibr* and *kalibr_allan* had the best performance, so they were used to calibrate the devices individually and also together, by calculating the hand-eye calibration. Moreover, it was examined if the low frame rates of the VIO system and the type of synchronization that it uses would have an impact on the performance. During calibration, the first indications that the low frame rates of the camera and the IMU would have a negative impact on the accuracy of the system became obvious. The frame rates were lower than the suggested from the *kalibr* and the hand-eye calibration proved to be not very accurate.

The examined VIO system was initially evaluated with an experiment on the air. Also in this case the VINS-Fusion, OKVIS and ROVIO algorithms did not perform properly. ORB-SLAM2 and Visual-Inertial ORB-SLAM were executed successfully, with the last one exhibiting remarkable performance. Also in this experiment the version of ORB-SLAM2 that is augmented with an IMU was proved to produce much better results, achieving a mean APE of 1.78 cm. For the underwater experiments a new camera orientation was proposed, pointing forward with a slight inclination downwards. Since there was no available underwater dataset with such a camera orientation or representing an underwater area with man-made installations, an artificial environment was set up in an indoor tank. Several experiments were conducted in the created underwater environment

and datasets were produced in a ROS format. The Visual-Inertial ORB-SLAM was evaluated with these datasets. The algorithm managed to keep track throughout the whole experiments and estimated roughly the traversed trajectory. However, the performance is not what was expected as in one of the best cases an APE of 11.5 cm was achieved and did not accomplish loop closing. The reasons for this were narrowed down to the low frame rates of the camera and the IMU, the moderate accuracy of the hand-eye calibration, the type of synchronization between the camera and the IMU messages, the uneven distribution of the features in the scene and finally the pure rotations and the oscillating motions that the ROV executed. However, Visual-Inertial ORB-SLAM was proved to be robust against moving objects in the scene by successfully identifying them as outliers, while also could capture with detail the motions of the ROV, setting it a promising solution for underwater station-keeping.

The Visual-Inertial ORB-SLAM was optimized on the NVIDIA embedded device with CUDA and OpenCV accelerated algorithms. The NVIDIA Xavier offers a range of different power modes with various combinations of operating frequencies, online cores and power, so the performance of the algorithm was evaluated for all of them. By utilizing the embedded GPU the algorithm achieved in some cases even double the speed of the sequential version. This was proved to be vital in some cases, in which the sequential version was failing due to limited computing resources. More precisely, although the algorithm consists of three parallel threads, it executed successfully in a mode with only 2 cores by utilizing also the resources of the GPU. With a moderate number of features the accelerated version performed between 6.8 - 12.5 FPS, which is quite good for the type of the ROV's motions. The measured temperatures on the embedded device were in low and manageable levels, reaching at most the 32°C. The utilization of RAM and GPU fluctuates in low levels as well, with the RAM utilization being around 22% and for the GPU around 7%. The CPU utilization depends a lot on the number of online cores and the operating frequency, ranging between 37 - 97%. The power consumption of the embedded device is far less than the maximum provided and it ranges between 6.1 - 13.1 Watts. The results showed that the operating frequency plays a more important role compared to the number of online cores for the algorithm execution speed and power consumption. Finally, according to the results the map size grows approximately linearly with relevance to the time as long as the robot regularly visits new areas. The algorithm handles the map size efficiently and its overall size remains in affordable values for a moderate embedded system.

6 Conclusions & Future Work

In this MSc thesis a Visual Inertial Odometry (VIO) system was deployed and evaluated for underwater vehicles navigation in subsea structured areas that are occupied by man-made installations. Remotely Operated underwater Vehicle (ROV)s are widely used for various applications that take place subsea. In many of these applications, the precise localization and positioning of the underwater vehicle is essential. While ROVs can carry a number of different sensors, many of them can not be exploited for efficient motion estimation, due to the costs, complexity, weight and inaccuracy. Taking into account that camera sensors are relevantly cheap, small, light-weight, have low power requirements and most of the robots carry one, Visual Odometry (VO) was selected for the underwater motion estimation. For the camera setup, the monocular approach was chosen because among others it offers mass reduction, less electronics, less connections, less power consumption, less mounted objects on the ROV, requires easier and cheaper housing, and the approach is more portable to various ROVs. For the camera a wide angle lens was selected, providing more observable features in the Field of View (FOV) for a longest period of time. The type of camera that was chosen is a machine vision camera, as it offers high and custom resolutions and Frames Per Second (FPS), robust design to withstand extreme conditions, reliability, technical support and extensive documentation, low level hardware and firmware control, precise data tracking and timing and it is programmable. For the machine vision camera to interact with the rest of the hardware, a low level interface was implemented in C++ and the camera was configured properly. The monocular VO approach was augmented with an Inertial Measurement Unit (IMU), that provides three axis accelerations and velocities, so that the absolute scale can be estimated as well. For the camera and IMU synchronization a micro-controller was also employed. On top of that, one of the most recent and powerful embedded systems of NVIDIA, the Xavier Jetson, with a high performance Central Processing Unit (CPU) and Graphics Processing Unit (GPU) was utilized to complete the proposed VIO system.

This thesis proposed a different orientation for the camera, compared to the most common ones that use a downward looking camera to observe the seabed. The proposed orientation has the camera pointing forward with a slight inclination downwards, so that features can be extracted from the seabed and from the man-made installations as well. The subsea environment is harsh and unpredictable, introducing the effects of attenuation, scattering and distortion. The floating particles and the turbidity of the water make it difficult to extract and track features robustly, while a lot of subsea areas lack of features. Due to the nature of the underwater environment the feature-based methods were preferred for motion estimation, as they are more robust and invariant to dynamic conditions.

To evaluate the VIO system and the algorithms, there was the need of underwater datasets. There is a significant quantity of datasets for the ground and air domains,

however for the underwater case they are much more limited. Also most of the underwater datasets with camera and IMU measurements feature a downward looking camera pointing to the seabed, while none was found taking place in an area with man-made structures. For these reasons, an artificial underwater environment was created in an indoor tank and datasets were produced in a Robot Operating System (ROS) format.

Initially, a VO algorithm was implemented in C++ with OpenCV based on an open source approach. With this implementation various popular feature detectors/descriptors were evaluated and some first indicative conclusions were drawn for their performance. This stage made evident that more complete algorithms, integrating additional techniques for better accuracy, are required for qualitative results in the underwater domain. Hence, five of the most recent VO/VIO algorithms were selected to be evaluated along with the proposed VIO system and the underwater datasets.

ORB-SLAM2 performed quite well in some smooth underwater cases, achieving to keep track for the whole sequence and to close a loop. On the other hand, in more challenging underwater experiments with significant noise and intense oscillating motions it failed to achieve loop closing, but it estimated roughly the traversed trajectory. In a dataset that had the worse image quality ORB-SLAM2 failed even to keep track for the whole trajectory. It became evident that in more turbid water the execution speed of the algorithm decreased while attempting to extract salient features from the scene. Moreover, the results showed that most of the drift in trajectory estimation was caused due to miss-estimated rotations. The VINS-Fusion, OKVIS and ROVIO algorithms did not perform properly most probably due to their high sensitivity in their configuration regarding the IMU parameters and calibration. During the experiments and the investigation it was noticeable how important is the precise IMU and camera calibration, and how significant is to estimate accurately the configuration parameters that are associated with them. For the IMU especially in turned out that there is some ambiguity in deriving some of its parameters. Although the aforementioned three VIO algorithms did not provide usable results, the Visual-Inertial ORB-SLAM exhibited remarkable performance in some experiments and the advantages of augmenting the ORB-SLAM2 with an IMU became obvious. Compared to ORB-SLAM2, the Visual-Inertial ORB-SLAM not only estimated the absolute scale, but it performed with higher accuracy as well. Also it was not so sensitive as the other VIO algorithms in the parameters of its configuration.

The impact of applying image processing filters on the underwater datasets and the affect on the performance of the algorithms was assessed. Various contrast enhancement and noise reduction filters were embedded in the VIO pipelines and were tested. Although the result was obviously better for the human eye, the performance of the algorithms was decreased.

Eventually, the Visual-Inertial ORB-SLAM was proved to be the most suitable algorithm for the examined case and it was deployed along with the VIO system. Various calibration methods were investigated and the VIO system was calibrated with the most suitable one, to operate both, on air and in water. The system was initially tested on air and exhibited very good performance, achieving an Absolute Position Error (APE) of 1.78 cm. However, in the underwater experiments its performance was weaker. The algorithm managed to keep track throughout the whole experiments and estimated roughly the traversed trajectory, but at the best case the APE was 11.5 cm. The reasons for this

were narrowed down to the low frame rates of the camera and the IMU, the moderate accuracy of the hand-eye calibration, the type of synchronization between the camera and the IMU messages, the uneven distribution of the features in the scene and finally the pure rotations and the oscillating motions that the ROV executed. However, Visual-Inertial ORB-SLAM was proved to be robust against moving objects in the scene by successfully identifying them as outliers, while also could capture with detail the motions of the ROV, setting it a promising solution for underwater station-keeping.

The Visual-Inertial ORB-SLAM was optimized on the NVIDIA embedded device with CUDA and OpenCV accelerated algorithms. By utilizing the embedded GPU the algorithm achieved in some cases even double the speed of the sequential version. This was proved to be vital in some cases, in which the sequential version was failing due to limited computing resources. With a moderate number of features the accelerated version performed between 6.8 - 12.5 FPS, which is quite good for the type of the ROV's motions. The measured temperatures on the embedded device were in low and manageable levels, reaching at most the 32°C. The utilization of RAM and GPU fluctuated in low levels as well, with the RAM utilization being around 22% and for the GPU around 7%. The CPU utilization depended a lot on the number of online cores and the operating frequency, and ranged between 37 - 97%. The power consumption of the embedded device was far less than the maximum provided and it ranged between 6.1 - 13.1 Watts. Furthermore, according to the results the produced map size grows approximately linearly with relevance to the time as long as the robot regularly visits new areas. The algorithm handled the map size efficiently and its overall size remained in affordable values for a moderate embedded system.

To conclude, a complete VIO system, including hardware and software, was deployed and tested for underwater scenarios. In most cases it managed to keep track of the ROV motion during the whole trajectory, however the accuracy in the best case was around 11.5 cm. The accelerated version of the algorithm was executed at 6.8 - 12.5 FPS on the embedded device. Although the aforementioned values are not ideal, considering the trade-off between cost - accuracy and that other means of underwater navigation achieve the same results or even worse, the precision of trajectory estimation is adequate. In addition, even the worst number of FPS is also adequate taking into account the smooth motions of a ROV. The deployed system was capable of estimating the traversed trajectory underwater in a structured environment and showed promising results for underwater station keeping. Finally, the flaws of the system have been identified and with further improvement more satisfying results can be achieved.

6.1 Future Work

As future work the frame rates of the VIO system should be examined. More precisely the operating frequency of the SBG IMU should be increased to 200 Hz, which is considered to be an ideal value. Moreover, the FPS of the FLIR machine vision camera should be increased as well, ideally at 20 FPS for post processing data. On a real time scenario, the FPS should be close to maximum obtained FPS of 12.5 Hz. To achieve the 200 Hz update rate the IMU configuration has to be further investigated, while any component of the VIO framework that interacts with the IMU messages should be revisited. Due to

the long cables that are involved in underwater operations the IMU should still transmit over those at 100 Hz, hence it should be taken into account how to provide at 200 Hz data in the VIO algorithm, and transmit at 100 Hz over the long cables. For the camera to operate at 20 FPS some bottlenecks have to be bypassed, like the read/write speed of memory and any possible CPU overloading. By the time these update rates are achieved the calibration methods that were utilized in this thesis should be conducted again to observe if indeed these new update rates improve its accuracy. To evaluate its accuracy the translation vector of the hand-eye calibration matrix should be examined and ideally should match the translation values of the 3D CAD model. Afterwards, the experiments should be conducted again with the new calibration and the new update rates, to determine the level of their impact on the trajectory estimation accuracy. Furthermore, the waterproof housing, that contains the VIO system, should be mounted closer to the body of the ROV to overcome the undesired oscillations. On top of these, the underwater environment in the tank should be reformed with scaled down models of real structures, so that features are better distributed along the scene, and the impact of this factor on the algorithm's performance should be assessed.

The VIO system was tested offline, in the sense that initially the dataset was recorded and stored into memory, and afterwards it was provided as input to the VIO algorithm. The intermediate step was to refine the acquired data in case of losses and to pack it into a ROS bag format. As future work the VIO system should be tested online by providing the incoming data from the IMU and the camera to the algorithm in real-time, and determine the impact of any possible frame losses. In addition, this thesis examined the performance of the algorithm for station keeping, however this was done only roughly by assessing the visual results. The output estimated motion of the algorithm should be provided as input to a control algorithm that is responsible for the ROV thrusters manipulation, and its performance should be tested on the field.

During this thesis it was determined that the other VIO algorithms that were examined, failed to properly execute probably due to the low update rates of the VIO system and the inaccurate hand-eye calibration. Once the enhanced update rates are achieved these algorithms should be examined again, and if they function properly their performance in the underwater scenarios should be compared.

Although the tracking thread of the Visual-Inertial ORB-SLAM is the one that has to be executed in real-time and its performance is critical, it was observed that in some cases the mapping thread was so slow that it probably made the algorithm to loose the track. Since the GPU utilization is quite low, it should be further investigated if parts of the mapping thread can be accelerated as well.

Some further future work regards the mapping. The mapping process should be investigated for efficient and robust mapping techniques, aiming to increase the performance and accuracy of navigation and positioning. In the same domain, methods for efficient underwater map construction should be examined, for creating high quality visual representation of the field, so that it can be used in various applications, like planning for structures installment in the seabed. In the same context, object detection and collision avoidance should be examined as well, while areas of the mapped field, should be classified as critical or safe. The purpose of these is the ROV to be capable of navigating efficiently around these structured areas. The aforementioned proposals aim

to complement the implemented system of this thesis. The ultimate goal will be to have a complete system, with which the ROVs that can carry a UPS battery will be capable to autonomously navigate away from the underwater structures in case that the tether is cut and the communication is lost.

Last but not least, machine learning should be considered as well. The performance of machine learning methods should be researched for motion and scale estimation, and evaluate their performance in underwater cases. Machine learning methods could also be utilized prior to an underwater operation, to identify the characteristics of the environment that the ROV is going to operate and tune the VO algorithm with suitable parameters for the specific case.

Bibliography

[1] Y. Sato, T. Maki, K. Masuda, T. Matsuda, and T. Sakamaki, "Autonomous docking of hovering type auv to seafloor charging station based on acoustic and visual sensing," in *2017 IEEE Underwater Technology (UT)*. IEEE, 2017, pp. 1–6.

[2] G. Antonelli, *Underwater Robots*. Springer, 2018.

[3] J. Horgan and D. Toal, "Computer vision applications in the navigation of unmanned underwater vehicles," in *Underwater vehicles*. IntechOpen, 2009.

[4] "Wikipedia: Pinhole camera model," Available online: https://en.wikipedia.org/wiki/Pinhole_camera_model, accessed: 2019.

[5] "Opencv: Camera calibration and 3d reconstruction," Available online: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#camera-calibration-and-3d-reconstruction, accessed: 2019.

[6] D. Nistér, O. Naroditsky, and J. Bergen, "Visual odometry," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, vol. 1. Ieee, 2004, pp. I–I.

[7] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE robotics & automation magazine*, vol. 18, no. 4, pp. 80–92, 2011.

[8] H. P. Moravec, "Obstacle avoidance and navigation in the real world by a seeing robot rover." STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, Tech. Rep., 1980.

[9] M. O. Aqel, M. H. Marhaban, M. I. Saripan, and N. B. Ismail, "Review of visual odometry: types, approaches, challenges, and applications," *SpringerPlus*, vol. 5, no. 1, p. 1897, 2016.

[10] M. Shah, "Fundamentals of computer vision," *University of Central Florida*, 1997.

[11] M. C. Deans and M. Hebert, "Bearings-only localization and mapping," Ph.D. dissertation, Citeseer, 2005.

[12] M. Li, B. H. Kim, and A. I. Mourikis, "Real-time motion tracking on a cellphone using inertial sensing and a rolling-shutter camera," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 4712–4719.

[13] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert, "Learning monocular reactive uav control in cluttered natural environments," in *2013 IEEE international conference on robotics and automation*. IEEE, 2013, pp. 1765–1772.

[14] M. Borg, "Omnidirectional visual tracking," MSc dissertation, University of Reading, 2003.

[15] K. Konolige, "Projected texture stereo," in *2010 IEEE International Conference on Robotics and Automation.* IEEE, 2010, pp. 148–155.

[16] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 647–663, 2012.

[17] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison *et al.*, "Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera," in *Proceedings of the 24th annual ACM symposium on User interface software and technology.* ACM, 2011, pp. 559–568.

[18] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "Dtam: Dense tracking and mapping in real-time," in *2011 international conference on computer vision.* IEEE, 2011, pp. 2320–2327.

[19] Guillermo Gallego, Tobi Delbruck, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi,Stefan Leutenegger, Andrew Davison, Jorg Conradt, Kostas Daniilidis, Davide Scaramuzza, "Event-based vision: A survey." [Online]. Available: http://rpg.ifi.uzh.ch/docs/EventVisionSurvey.pdf

[20] A. Yamashita, Y. Shirane, and T. Kaneko, "Monocular underwater stereo-3d measurement using difference of appearance depending on optical paths," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems.* IEEE, 2010, pp. 3652–3657.

[21] T. P. Pachidis and J. N. Lygouras, "Pseudostereo-vision system: A monocular stereo-vision system as a sensor for real-time robot applications," *IEEE Transactions on Instrumentation and Measurement*, vol. 56, no. 6, p. 25472560, 2007.

[22] K. Yousif, A. Bab-Hadiashar, and R. Hoseinnezhad, "An overview to visual odometry and visual slam: Applications to mobile robotics," *Intelligent Industrial Systems*, vol. 1, no. 4, pp. 289–311, 2015.

[23] S. Lazebnik. Structure from motion. lecture slides. Accessed: 2019. [Online]. Available: http://slazebni.cs.illinois.edu/spring19/lec17_sfm.pdf

[24] D. Scaramuzza. Visual odometry and slam: past, present, and the robust-perception age. Accessed: 2019. [Online]. Available: https://www.rsj.or.jp/databox/international/iros16tutorial_2.pdf

[25] H. Oleynikova. Visual slam slides. Accessed: 2019. [Online]. Available: http://www.ros.org/presentations/2010-08-Helen-VSLAM.pdf

[26] L. A. Clemente, A. J. Davison, I. D. Reid, J. Neira, and J. D. Tardós, "Mapping large loops with a single hand-held camera." in *Robotics: Science and Systems*, vol. 2, no. 2, 2007.

[27] C. G. Harris, M. Stephens *et al.*, "A combined corner and edge detector." in *Alvey vision conference*, vol. 15, no. 50. Citeseer, 1988, pp. 10–5244.

[28] J. Shi and C. Tomasi, "Good features to track," Cornell University, Tech. Rep., 1993.

[29] D. G. Lowe *et al.*, "Object recognition from local scale-invariant features." in *iccv*, vol. 99, no. 2, 1999, pp. 1150–1157.

[30] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[31] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *European conference on computer vision*. Springer, 2006, pp. 404–417.

[32] "Opencv 3.0 - tutorials - introduction to surf," Available online: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html#surf, accessed: 2019.

[33] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *European conference on computer vision*. Springer, 2006, pp. 430–443.

[34] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *European conference on computer vision*. Springer, 2010, pp. 778–792.

[35] E. Rublee, V. Rabaud, K. Konolige, and G. R. Bradski, "Orb: An efficient alternative to sift or surf." in *ICCV*, vol. 11, no. 1. Citeseer, 2011, p. 2.

[36] B. Fan, Z. Wang, F. Wu *et al.*, *Local image descriptor: modern approaches*. Springer, 2015, vol. 108.

[37] S. Leutenegger, M. Chli, and R. Siegwart, "Brisk: Binary robust invariant scalable keypoints," in *2011 IEEE international conference on computer vision (ICCV)*. Ieee, 2011, pp. 2548–2555.

[38] M. Voisin-Denoual, "Monocular Visual Odometry for Underwater Navigation," Master's thesis, KTH Royal Institute of Technology, Sweden, 2018.

[39] "Opencv: Optical flow." Available online: https://docs.opencv.org/3.4/d7/d8b/tutorial_py_lucas_kanade.html, accessed: 2019.

[40] Opencv documentation, feature matching with flann. Accessed: 2019. [Online]. Available: http://www.swarthmore.edu/NatSci/mzucker1/opencv-2.4.10-docs/doc/tutorials/features2d/feature_flann_matcher/feature_flann_matcher.html

[41] C. Tomasi and T. Kanade, "Detection and tracking of point features," International Journal of Computer Vision, Tech. Rep., 1991.

[42] T. Watanabe, "A fuzzy ransac algorithm based on reinforcement learning concept," *2013 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pp. 1–6, 2013.

[43] A. Bab-Hadiashar and D. Suter, "Robust optic flow computation," *International Journal of Computer Vision*, vol. 29, no. 1, pp. 59–77, 1998.

[44] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artificial intelligence*, vol. 17, no. 1-3, pp. 185–203, 1981.

[45] B. D. Lucas, T. Kanade *et al.*, "An iterative image registration technique with an application to stereo vision," 1981.

[46] J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *European conference on computer vision*. Springer, 2014, pp. 834–849.

[47] J. Engel, J. Sturm, and D. Cremers, "Semi-dense visual odometry for a monocular camera," in *Proceedings of the IEEE international conference on computer vision*, 2013, pp. 1449–1456.

[48] C. Forster, M. Pizzoli, and D. Scaramuzza, "Svo: Fast semi-direct monocular visual odometry," in *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2014, pp. 15–22.

[49] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 3, pp. 611–625, 2017.

[50] C. Forster, Z. Zhang, M. Gassner, M. Werlberger, and D. Scaramuzza, "Svo: Semidirect visual odometry for monocular and multicamera systems," *IEEE Transactions on Robotics*, vol. 33, no. 2, pp. 249–265, 2017.

[51] F. Fraundorfer and D. Scaramuzza, "Visual odometry: Part ii: Matching, robustness, optimization, and applications," *IEEE Robotics & Automation Magazine*, vol. 19, no. 2, pp. 78–90, 2012.

[52] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[53] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[54] F. Hidalgo, C. Kahlefendt, and T. Bräunl, "Monocular orb-slam application in underwater scenarios," in *2018 OCEANS-MTS/IEEE Kobe Techno-Oceans (OTO)*. IEEE, 2018, pp. 1–4.

[55] L. Silveira, F. Guth, P. Drews-Jr, P. Ballester, M. Machado, F. Codevilla, N. Duarte-Filho, and S. Botelho, "An open-source bio-inspired solution to underwater slam," *IFAC-PapersOnLine*, vol. 48, no. 2, pp. 212–217, 2015.

[56] M. Ferrera, J. Moras, P. Trouvé-Peloux, and V. Creuze, "Real-time monocular visual odometry for turbid and dynamic underwater environments," *Sensors*, vol. 19, no. 3, p. 687, 2019.

[57] J. Jung, J.-H. Li, H.-T. Choi, and H. Myung, "Localization of auvs using visual information of underwater structures and artificial landmarks," *Intelligent Service Robotics*, vol. 10, no. 1, pp. 67–76, 2017.

[58] A. Kim and R. M. Eustice, "Real-time visual slam for autonomous underwater hull inspection using visual saliency," *IEEE Transactions on Robotics*, vol. 29, no. 3, pp. 719–733, 2013.

[59] S. Rahman, A. Q. Li, and I. Rekleitis, "Sonar visual inertial slam of underwater structures," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–7.

[60] ——, "Svin2: Sonar visual-inertial slam with loop closure for underwater navigation," *arXiv preprint arXiv:1810.03200*, 2018.

[61] M. Nawaf, D. Merad, J.-P. Royer, J.-M. Boï, M. Saccone, M. Ben Ellefi, and P. Drap, "Fast visual odometry for a low-cost underwater embedded stereo system," *Sensors*, vol. 18, no. 7, p. 2313, 2018.

[62] K. Sukvichai, K. Wongsuwan, N. Kaewnark, and P. Wisanuvej, "Implementation of visual odometry estimation for underwater robot on ros by using raspberrypi 2," in *2016 International Conference on Electronics, Information, and Communications (ICEIC)*. IEEE, 2016, pp. 1–4.

[63] M. Rossi, P. Trslić, S. Sivčev, J. Riordan, D. Toal, and G. Dooly, "Real-time underwater stereofusion," *Sensors*, vol. 18, no. 11, p. 3936, 2018.

[64] F. Shkurti, I. Rekleitis, and G. Dudek, "Feature tracking evaluation for pose estimation in underwater environments," in *2011 Canadian Conference on Computer and Robot Vision*. IEEE, 2011, pp. 160–167.

[65] A. Q. Li, A. Coskun, S. M. Doherty, S. Ghasemlou, A. S. Jagtap, M. Modasshir, S. Rahman, A. Singh, M. Xanthidis, J. M. OKane *et al.*, "Experimental comparison of open source vision-based state estimation algorithms," in *International Symposium on Experimental Robotics*. Springer, 2016, pp. 775–786.

[66] J. Delmerico and D. Scaramuzza, "A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 2502–2509.

[67] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, vol. 35, no. 10, pp. 1157–1163, 2016.

[68] V. Creuze, "Monocular odometry for underwater vehicles with online estimation of the scale factor," in *IFAC 2017 World Congress*, 2017.

[69] A. C. Duarte, G. B. Zaffari, R. T. S. da Rosa, L. M. Longaray, P. Drews, and S. S. Botelho, "Towards comparison of underwater slam methods: An open dataset collection," in *OCEANS 2016 MTS/IEEE Monterey*. IEEE, 2016, pp. 1–5.

[70] M. Ferrera, J. Moras, P. Trouvé-Peloux, V. Creuze, and D. Dégez, "The aqualoc dataset: Towards real-time underwater localization from a visual-inertial-pressure acquisition system," *arXiv preprint arXiv:1809.07076*, 2018.

[71] A. Mallios, E. Vidal, R. Campos, and M. Carreras, "Underwater caves sonar data set," *The International Journal of Robotics Research*, vol. 36, no. 12, pp. 1247–1251, 2017.

[72] E. Royer, M. Lhuillier, M. Dhome, and J.-M. Lavest, "Monocular vision for mobile robot localization and autonomous navigation," *International Journal of Computer Vision*, vol. 74, no. 3, pp. 237–260, 2007.

[73] "Machine vision rapidly replacing consumer slr cameras in industrial imaging." Available online: https://www.adimec.com/machine-vision-rapidly-replacing-consumer-slr-cameras-in-industrial-imaging/, accessed: 2019.

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

[75] R. Mur-Artal and J. D. Tardós, "Visual-inertial monocular slam with map reuse," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 796–803, 2017.

[76] S. Leutenegger, P. Furgale, V. Rabaud, M. Chli, K. Konolige, and R. Siegwart, "Keyframe-based visual-inertial slam using nonlinear optimization," *Proceedings of Robotis Science and Systems (RSS) 2013*, 2013.

[77] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart, "Robust visual inertial odometry using a direct ekf-based approach," in *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2015, pp. 298–304.

[78] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.

[79] C. Zhang, Y. Liu, F. Wang, Y. Xia, and W. Zhang, "Vins-mkf: A tightly-coupled multi-keyframe visual-inertial odometry for accurate and robust state estimation," *Sensors*, vol. 18, no. 11, p. 4036, 2018.

VO Algorithm & Feature Descriptors/Detectors Comparison

A

In this chapter the initial approach for a monocular Visual Odometry (VO) system is described. A VO algorithm is enhanced to support various feature detectors/descriptors and to utilize them for motion estimation. In this chapter mainly some popular detectors/descriptors are evaluated and compared, by considering metrics about their performance, accuracy and quality. The algorithm did not produce results of adequate quality for the underwater domain, so the performance of the feature descriptors/detectors is evaluated on a ground dataset. For this reason the results of these experiments were separated from the ones that regard the underwater domain.

A.1 Monocular VO Algorithm Implementation

The VO algorithm was implemented in C++ along with OpenCV, and is based on the published approach available on Github[1]. The published approach uses the FAST feature detector and Kanade-Lucas-Tomasi (KLT) tracking to track features from frame to frame. However, the code was enhanced by the author of this thesis to support the following feature detectors/descriptors:

- Scale Invariant Feature Transform (SIFT)
- Speeded-Up Robust Features (SURF)
- Oriented FAST and Rotated BRIEF (ORB)
- Binary Robust Invariant Scalable Keypoints (BRISK)
- Accelerated-KAZE (A-KAZE)
- Features from Accelerated Segment Test (FAST)
- Shi-Tomasi

while additionally to the KLT tracking method, features can be also matched from frame to frame using their descriptors. Furthermore, the feature detection has been optimized by detecting features in a uniformly distributed manner along the image. Moreover, various metrics have been deployed to provide information about the performance of each detector. Finally, a real-time visualizer was implemented in Python, illustrating a grid with proper measurement units and depicting the estimated trajectory against the groundtruth data. In Algorithm 1 a brief pseudo-code of the implementation is presented, while the algorithm is described below.

[1] `https://github.com/avisingh599/mono-vo`

The user can initially choose any of the aforementioned descriptors/detectors, while it is also possible to configure all the parameters that come along with them. The incoming frames are read one by one and if they are distorted the provided calibration parameters are used to undistort them. In case the image is in RGB format it is converted to grayscale as well. Two major methods are implemented to keep track of features along the traversed trajectory. The first one is the KLT tracking, further described in 2.4.8, which can be used along with the FAST or Shi-Tomasi detectors. After the features are detected the tracker uses a patch around the feature and attempts to find its location in the next frame, considering the pixel intensities. The other option is to detect features with one of the rest mentioned detectors and calculate their descriptors. In the next incoming frame features are detected again and their descriptors are calculated as well. The descriptors from the two last frames are compared and features are matched. For the SIFT and SURF the FLANN matcher is used, while for the ORB, A-KAZE and BRISK the HAMMING matcher, since their descriptors are binary. FLANN stands for Fast Library for Approximate Nearest Neighbors, which features optimized algorithms for faster searching of nearest neighbor and is more efficient than brute force matching for large datasets. After the matching is done an extra step is implemented to keep only the best matches. For every feature the first two best matches are calculated. If the quality of the first match is bigger than a threshold times the quality of the second match, then it is considered a good match and it is stored, otherwise it is discarded. To optimize the performance of the VO algorithm an additional step was implemented. It is best for the accuracy of the VO to detect features distributed along the whole image, because if the considered features are gathered to certain areas, and specifically towards the center of the image, then VO is biased and struggles to detect rotations in motion. To force the detectors to search for features evenly distributed, the image is split into a grid, while the user is able to choose the number of rows and columns to form this grid. The detectors are then forced to search for a number of features in every area formed by the grid.

At this point, feature correspondences have been determined and the locations of these features are known for both of the frames. This information is used as input to the Nister's 5-point algorithm, and along with Random Sample Consensus (RANSAC) the Essential Matrix is computed. The Nister's algorithm is used to solve an amount of non-linear equations, and only the minimum number of points are required, while the Essential Matrix has only five Degrees of Freedom (DoF). Ideally, only five points would be adequate to estimate the motion between two frames. However, the feature correspondences include also faulty matches. To overcome this problem the RANSAC algorithm is utilized. RANSAC is executed in a number of iterations, and for each iteration it randomly samples five points from the correspondences, the Essential Matrix is computed and it is checked if the other points are inliers with the derived Essential Matrix. The Essential Matrix that satisfies most of the points is finally used. More detailed information about the Essential Matrix, motion estimation and RANSAC, is available at 2.4.3 and 2.4.9.

Afterwards, the Essential Matrix is decomposed using Singular Value Decomposition (SVD) and the rotation matrix R and the translation vector t are obtained. The

trajectory is accumulatively estimated based on the following equations:

$$R_{pos} = RR_{pos} \tag{A.1}$$

and

$$t_{pos} = t_{pos} + tR_{pos} \tag{A.2}$$

where R_{pos} and t_{pos} are the accumulatively estimated rotation and translation up to that frame, and R and t are the estimated ones in the current frame. Here it has to be mentioned that the translation vector t has to be scaled correspondingly before estimating the new t_{pos}. In the current implementation the scale is derived by the provided groundtruth data. Otherwise, extra measurements should be used by another sensor, for example an Inertial Measurement Unit (IMU) or odometer.

In every frame the algorithm tracks features previously detected, so from frame to frame features get lost. In every iteration it is checked if the current number of tracked features is below a threshold. If the number of features is significantly decreased, then new features are detected and tracked from the start.

In many cases computer vision algorithms are enhanced with some heuristics. In this implementation the dominant motion is considered to be the forward one. If the car is stable but another object passes in front of it, the VO algorithm will wrongly detect a side-way motion. By considering the forward motion as the dominant one, the other types of motion are discarded and the algorithm becomes more robust. For the underwater environment and the Remotely Operated underwater Vehicle (ROV) the motion is characterized by 6 DoF and it is more difficult to apply similar restrictions. While fish are presented and objects might float, faulty detection of motion might take place. However, ROVs usually come with an IMU, hence if the VO algorithm assumes a motion, this can be verified by checking also the linear accelerations provided by the IMU.

Additionally to the VO, some visualizations were implemented as well (Figure A.1). The tracked features are illustrated with green circles in every frame, while on the right side of the frames the detector/descriptor that is used is shown. More information, like Frames Per Second (FPS), number of features currently tracked, and number of keypoints initially detected are depicted in real-time. Furthermore, a real-time visualizer was implemented in Python. The estimated trajectory from the C++ code is logged in real-time, while a real-time animation is executed in parallel in Python. The Python implementation reads the computed motion and prints the traversed trajectory. Along with the traversed trajectory the groundtruth data are presented as well.

A.2 Monocular VO Implementation Results

The KITTI dataset was utilized to examine the performance and accuracy of feature detectors and descriptors offered by OpenCV libraries, since the produced results on underwater datasets were not of adequate quality. Many sequences of the provided footage were examined, however here only the results of the sequences “00” and “01” are presented, as they contain two representative environments, taking place in inner-city and in rural areas. Due to the big length of the footage, approximately the 2/3 of the

Algorithm 1 Monocular VO algorithm

```
Choose feature detector/descriptor
Read frame
Undistort frame
Convert frame to grayscale
Detect features
if (!KLT tracking) then
    Calculate descriptors for the features
end if
for (Number of frames) do
    Read frame
    Undistort frame
    Convert frame to grayscale
    if (KLT tracking) then
        Track features from previous frame to current one with KLT
    else
        Split the frame into a grid
        Detect features in the current frame
        Calculate descriptors for the new features
        Match descriptors from the previous and the current frame
        Keep only the best matches
    end if
    Compute the essential matrix using Nister's 5-point algorithm and RANSAC
    Estimate R and t from the essential matrix
    Calculate the scale
    if (Current number of features < threshold) then
        Detect new features
        if (!KLT tracking) then
            Calculate descriptors for the features
        end if
    end if
    Log R and t
    Visualize trajectory
end for
```

whole sequence "00" was utilized, while the whole footage of sequence "01" was used. All the aforementioned feature detectors and descriptors and feature tracking methods were used.

The experiments were conducted on the "Laptop 1" setup. For all the cases the same re-detect threshold was considered, as well as the same amount of rows and columns to split the input image into a grid. For the case of Shi-Tomasi and ORB features it is possible to choose a maximum number of features, which was set to 1000. The rest of the detectors were extracting as many features as possible. However, it is possible to set a maximum threshold by configuring the threshold of determining the best feature

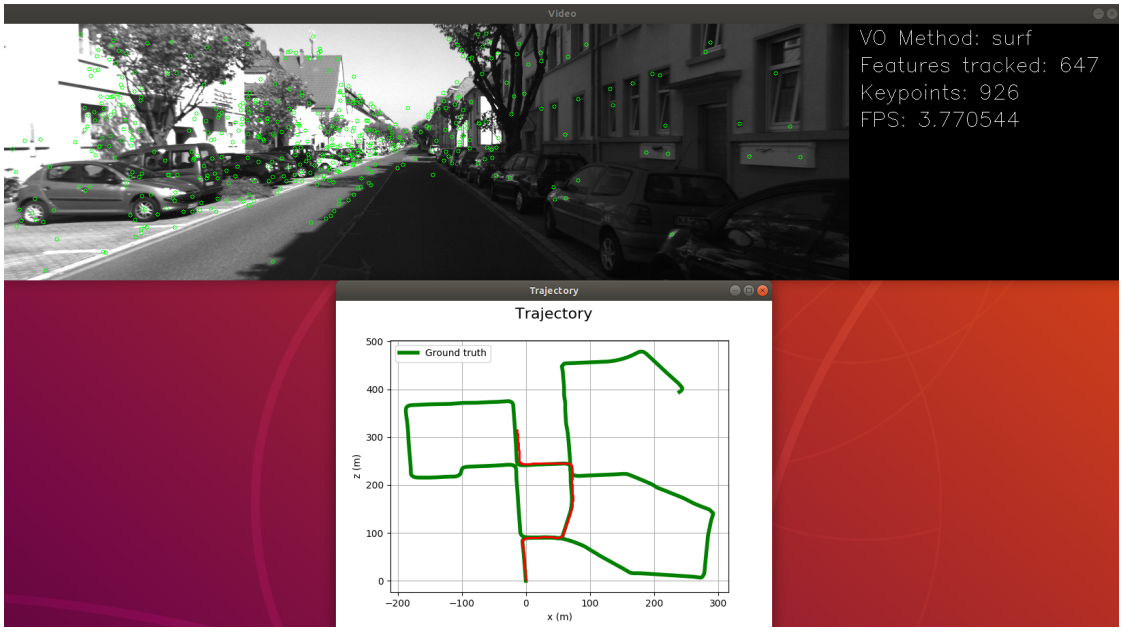


Figure A.1: Monocular VO execution, along with the visualizations. The green line represents the groundtruth trajectory, while the red line illustrates the estimated one in real-time.

matches. Setting an appropriate threshold can really limit the number of features. The re-detection threshold was set to 500 features for all the cases, and the image was split into a grid of 6 rows and 6 columns. All the feature detectors were used with their default settings, except from the thresholds regarding the detection of features. In some detectors it was slightly configured and adapted to the specific sequence of frames for a better performance.

Since seven in total methods were examined, some results, regarding the same measurements, have been split in two graphs for the sake of clarity. The resulting trajectories using the examined methods are illustrated in Figure A.2 and Figure A.3. With a rough observation on the aforementioned figures it can be derived that SIFT, BRISK, Shi-Tomasi, and SURF follow quite accurately the reference trajectory. On the other hand, ORB fails to accurately estimate a rotation and deviates a lot after that point, while FAST and A-KAZE totally miss the actual trajectory after a rotation in the middle of the experiment. In the point where these two feature detectors fail, the car was motionless for a while in a crossroad. These feature detectors were not able to estimate the actual direction of the car after that moment. Here it becomes quite obvious how prone to failure VO approaches are, since the error is accumulative throughout the traversed trajectory. Not only from time to time the accuracy becomes worse, due to the added errors, but if a motion is badly estimated the rest of the calculated motions become useless, at least as far as the whole traversed trajectory is concerned.

The results in the Figure A.4a and Table A.1 show the Absolute Position Error (APE). The cases of FAST and A-KAZE significantly failed and had to be mentioned

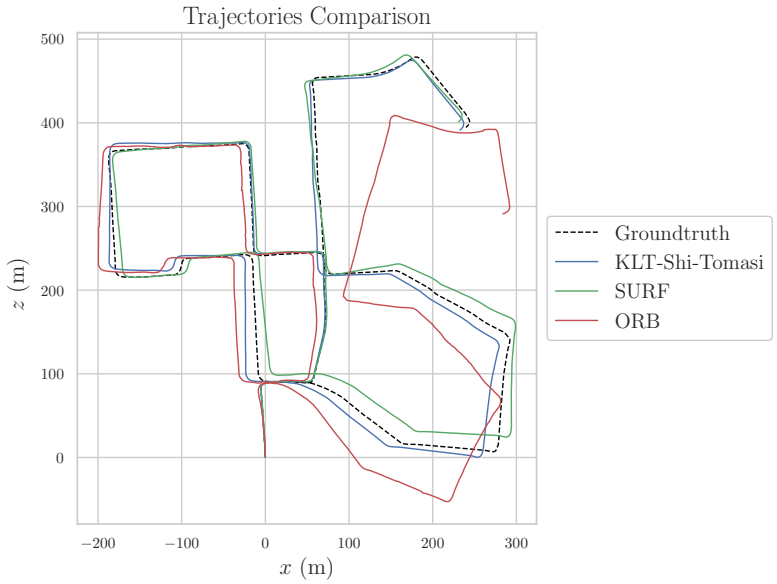


Figure A.2: Trajectories comparison using the Shi-Tomasi, SURF and ORB features.

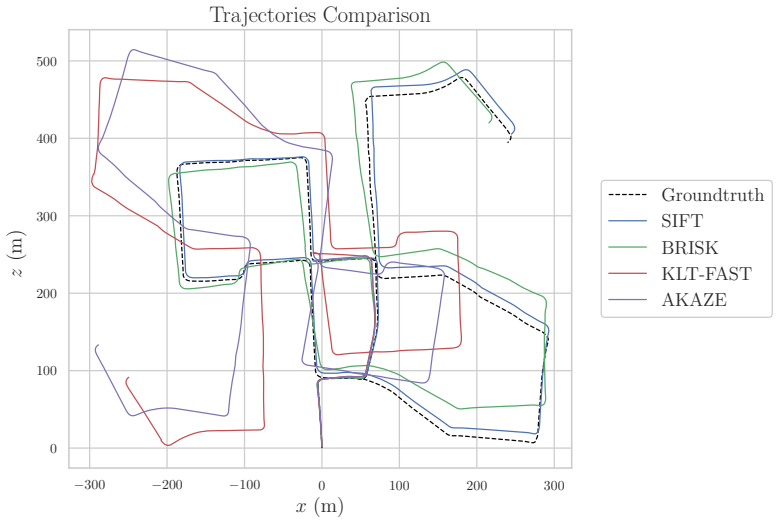
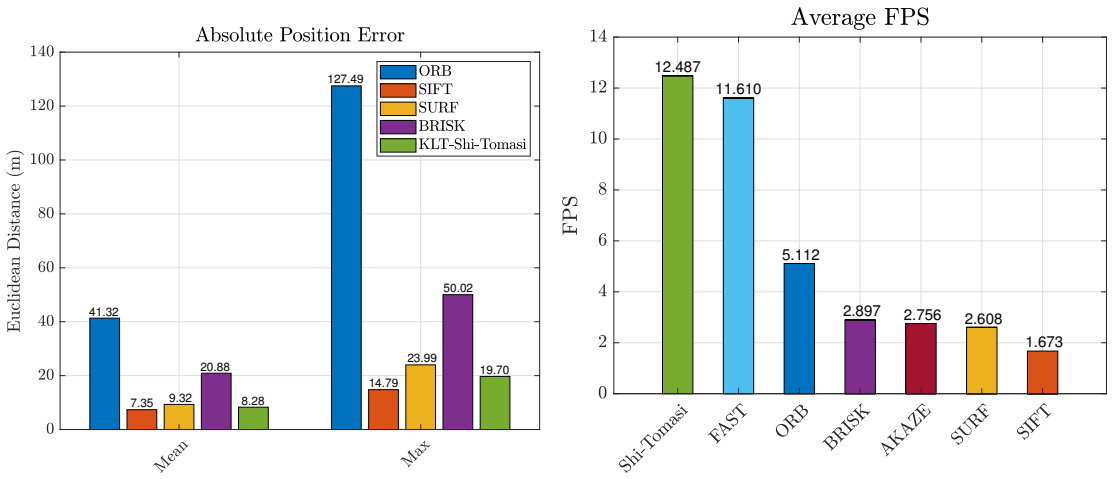


Figure A.3: Trajectories comparison using the SIFT, BRISK, FAST and AKAZE, in relevance to the groundtruth.

in a table to avoid degrading the clarity of the graph. Obviously these two are the worst ones in the specific experiment. From the rest of the features, ORB estimates wrongly a rotation and from that point on the APE unavoidably increases, but up to that point was performing adequately well. SIFT, SURF and Shi-Tomasi exhibit the best performance, verifying our expectations and performing according to the theory that indicates the robustness of these features.

In VO and Visual Simultaneous Localization And Mapping (V-SLAM) algorithms



(a) (b)

Figure A.4: (a) Mean and max values of absolute position error for ORB, SIFT, SURF, BRISK and Shi-Tomasi features. (b) Average FPS for all the examined features.

Table A.1: AKAZE and FAST APE

	Mean	**Max**
AKAZE	299.6737	725.0829
FAST	302.1520	728.9635

usually there is a trade-off between accuracy and performance. This can be seen also by comparing the Figures A.4a and A.4b. While it is clear that Shi-Tomasi and FAST methods exhibit the best performance in terms of speed, it has to be noted that these two feature detectors are executed along with an appearance-based method for finding feature correspondences from frame to frame. The computation complexity is significantly less in these cases, however the appearance-based methods are not so robust. In the examined experiment the visibility is quite good and there are not so rapid motions, and this is the reason that the Shi-Tomasi performs so accurately, while FAST fails with a high value of APE. On the other hand, SIFT and SURF, that are known to be of the most robust detectors/descriptors, lack of efficiency in speed, but they indeed exhibit the most accurate motion estimations. Somewhere in the middle lie the ORB and BRISK features. In this experiment BRISK keeps a good balance between speed and accuracy, but ORB is distinctively the faster method that uses feature descriptors. Of course ORB shows a notable APE, but this is due to the miss-estimation of a rotation.

Figure A.5b shows for all the feature types the number of keyframes. A keyframe in this experiment is considered a frame, during which the number of features, that were tracked up to this frame, has been significantly decreased and it is required to detect new features. So the frames, during which new features are detected for tracking, are called keyframes. Figure A.5a illustrates the average number of keypoints for all the examined feature types. Keypoints are considered to be the features that are detected on a keyframe. In Figure A.6b the average feature life is presented, meaning the number

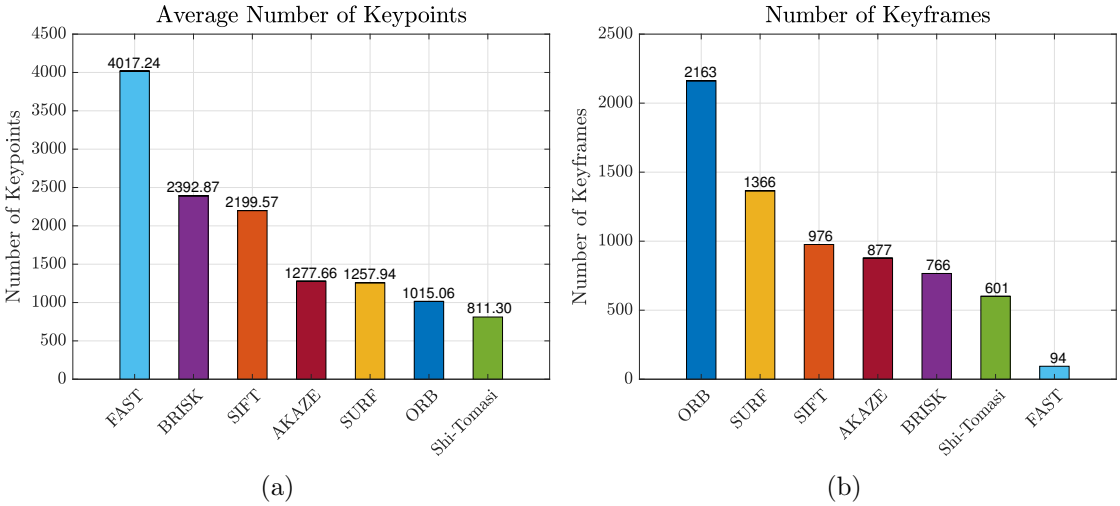


Figure A.5: (a) Average number of keypoints and (b) number of keyframes for all the examined features.

of frames that every feature is being tracked and not lost. The good matches, i.e. the number of successfully tracked keypoints from the keyframe to the next frame, are depicted in A.6a. It is evident that the number of keyframes is inversely proportional to the average feature life. The ORB features exhibit the worse performance in terms of duration, while the best ones are the FAST and Shi-Tomasi, that are used with the appearance-based performance. It seems that there is not a clear connection between the feature life and the resulting accuracy, as SIFT and SURF have one of the lowest values in this metric. However, the short duration of SIFT and SURF features, along with their robustness, indicate that these methods are quite strict when they match features and that is why they estimate the motions with high accuracy. The average number of keypoints is proportional to the number of good matches, keeping in mind that the ORB and Shi-Tomasi were restrained to 1000 features per keyframe. It is noticeable that the greater number of detected keypoints does not mean also better accuracy, while it is worth mentioning that Shi-Tomasi detects the least number of keypoints and at the same time exhibits very good accuracy in motion estimation.

In Figure A.7 is illustrated another experiment from the KITTI dataset, that takes place in a rural area. The reason it is presented here is to highlight an important observation about the appearance-based methods. Considering the first half part of the trajectory, it is evident that the appearance-based algorithms severely fail to capture the strong and lasting rotation that the car executes. In general, the intense rotations are one of the most tricky and hard parts for VO algorithms, because there is little parallax to the motion. The experiment takes place in a rural area, so the number of salient features is notably reduced and concentrated only in specific areas of the image. Another point to mention is that the specific footage starts with a rotation, meaning that even slight deviations in motion estimation lead to disastrous results for the rest of the trajectory. While Shi-Tomasi performed very well in the first experiment, here it fails, whereas all the other methods that utilize feature descriptors manage to estimate

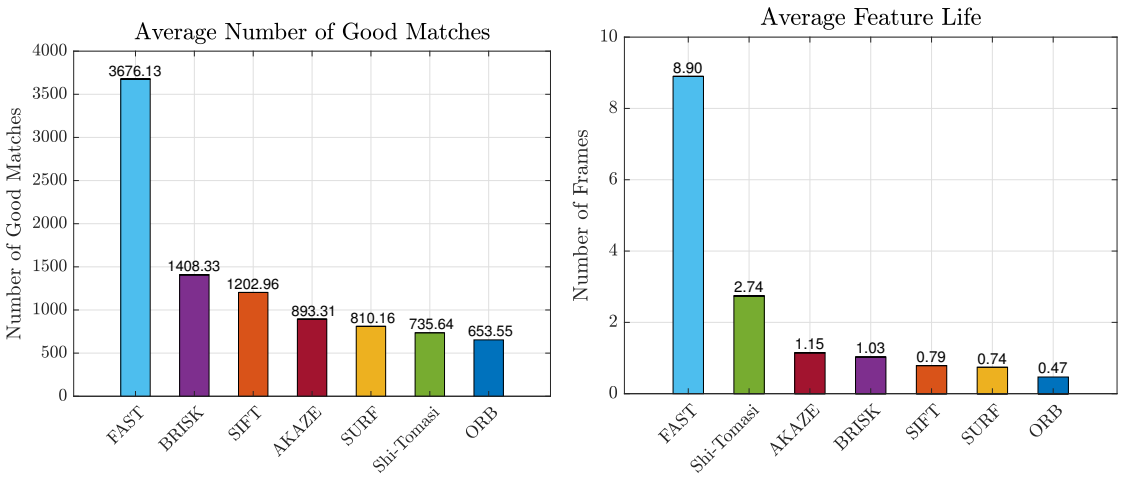


Figure A.6: (a) Average number of good matches and (b) average feature life for all the examined features.

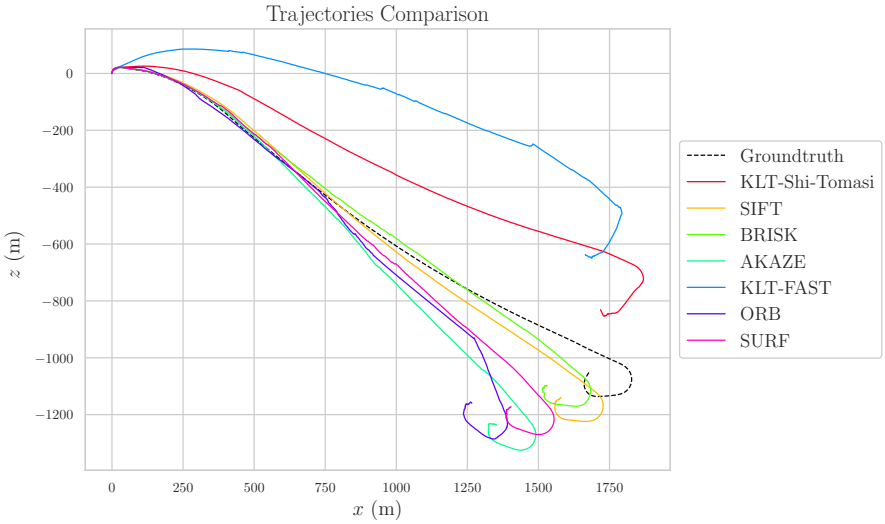


Figure A.7: Trajectories comparison for sequence "01" of the KITTI dataset, examining all the feature types.

correctly the initial intense rotation.

A.3 Conclusion

In VO and V-SLAM domains there is a huge amount of parameters and factors that can affect the resulting performance. Every experiment, featuring its proprietary movements and environment, might lead to a different result, even if the same feature descriptor/detector is used with the exact same parameters. This experiment was a rough

estimation of the performance of seven different feature types, implemented in OpenCV, in relevantly good conditions, with adequate visual clarity and smooth motions. However, some general conclusions can be derived, like the speed of each feature type, the number of features that are detected in a scene and how various types of motions are handled. The appearance-based methods for tracking exhibit the best performance, compared to feature descriptors for motion estimation, as it was expected. Tracking an amount of pixels intensities around a feature is faster than calculating a mathematical or binary descriptor for each of them and comparing them. In spite of that, the appearance-based methods are not so robust against rotation, scale, viewpoint and illumination variations and they have to be wisely used. In case of a challenging environment that is characterized by the aforementioned variations or extensive noise the appearance-based methods will exhibit significantly bad performance. Hence, their speed could be exploited in cases that are not so challenging. Moreover, for the specific experiment it seems that the average feature life does not necessarily have an impact on the accuracy of motion estimation. In addition, the amount of features that are detected and tracked do not have a direct impact on the accuracy of motion estimation either. It is more important that the detected features are well distributed along the image and that they are of adequate quality to survive scene variations. Among the evaluated detectors/descriptors the ORB exhibits the best performance in terms of speed. On top of that, it is invariant to scale, illumination and viewpoint changes. Considering also the advantages and robustness of using feature descriptors for motion estimation, ORB could be an efficient solution for challenging scenes if it could be further supported by other mechanisms, like loop closing or Bundle Adjustment (BA) to enhance its accuracy and consistency. This attempt to deploy a VO algorithm mainly helped the author to gain a more thorough understanding on the domain of VO and on all the examined feature detectors/descriptors. It was also evident after this that more sophisticated and complete VO algorithms should be utilized in challenging environment, such as the underwater, for proper results. The research and results of this attempt offered the required knowledge to narrow down the investigation of a wide range of VO algorithms to the ones that would be suitable for the subsea cases.

EuRoC Results

B

Figure B.1 shows the APE statistics for the examined Visual Inertial Odometry (VIO) algorithms. Visual-Inertial ORB-SLAM exhibits a balanced performance, with the smallest max error and a very low mean, around 25 cm. The ROVIO has the best mean in APE, however along with VINS-Fusion exhibit the highest deviation and has the highest max value, reaching the 1 m.

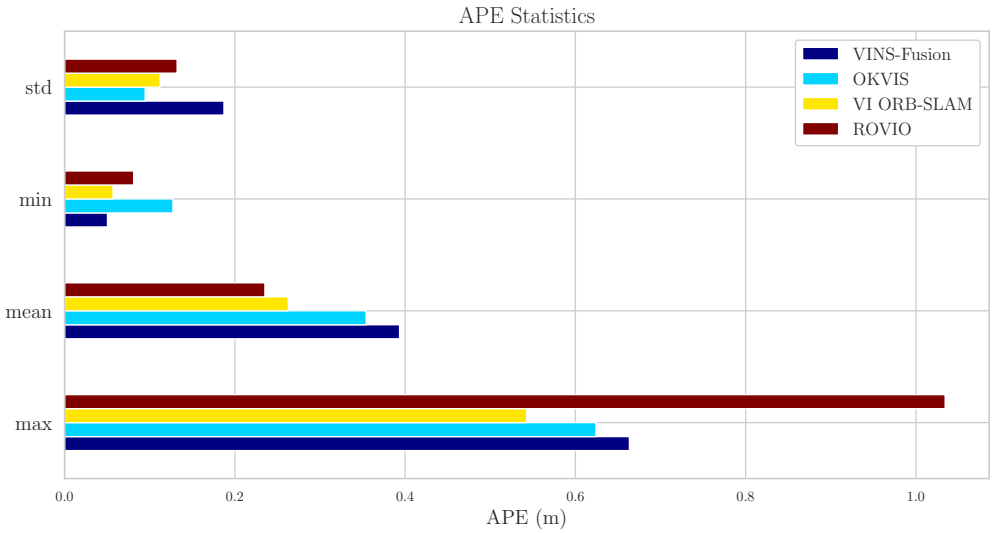


Figure B.1: Comparison among the VINS-Fusion, OKVIS, Visual-Inertial ORB-SLAM and ROVIO regarding the absolute position error statistics. The experiments were conducted on the EuRoC MH_01 dataset.

C

ORB-SLAM Visual Results

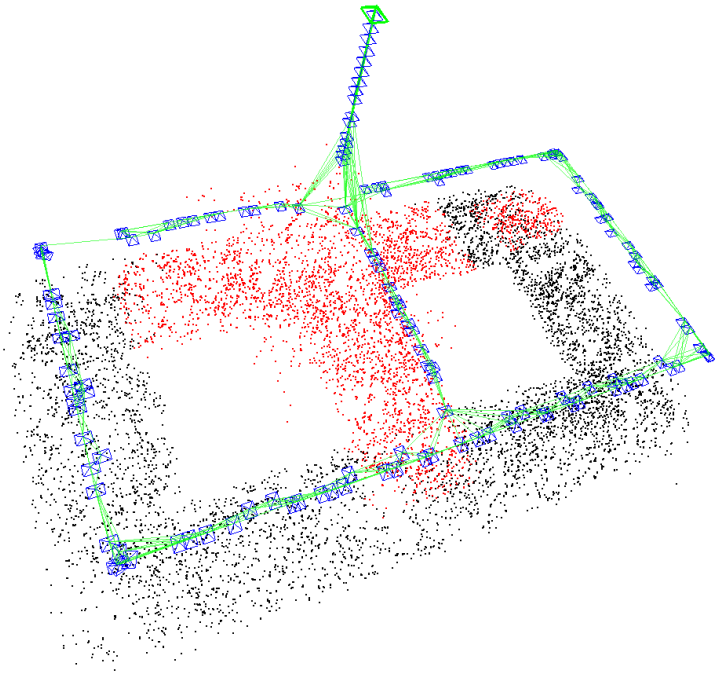


Figure C.1: Map with 3D point cloud produced by ORB-SLAM2 during the UWSim dataset of sequence "Second proposed".

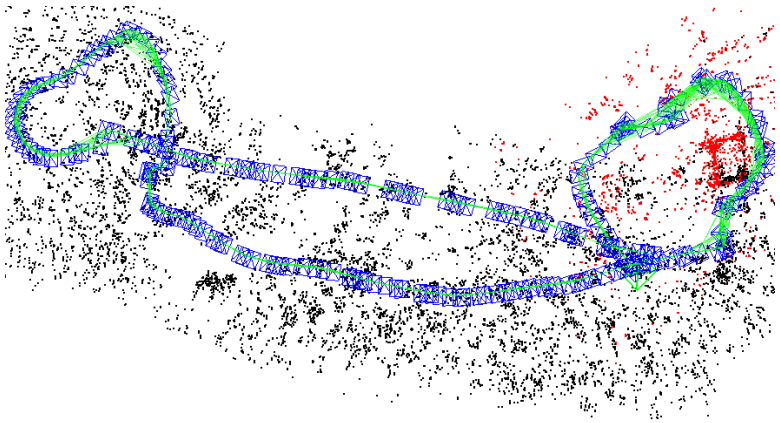


Figure C.2: Map with 3D point cloud produced by ORB-SLAM2 during the Archaeological dataset of sequence 01.