POLITECNICO DI TORINO

Department of Mechanical and Aerospace Engineering (DIMEAS)

Master of Science in Aerospace Engineering
LM-20 (DM270)

Master Thesis

Study of SLAM State of the Art Techniques for UAVs Navigation in Critical Environments

Supervisors

Giorgio Guglieri

Simone Godio

Candidate

Francesca Suriano

April 2021

Abstract

In recent years, Autonomous Navigation for flying robots has become a global challenge. Unmanned Aerial Vehicles (UAVs) have a high potential in both military and civil applications such as aerial reconnaissance, surveillance, search and rescue, product deliveries, agriculture, or infrastructure inspections. Autonomous Navigation of UAVs exploits an onboard Inertial Measurement Unit (IMU) which consists of three-axis accelerometer and gyroscope providing the linear acceleration and the angular velocity of the robot. The problem is that IMU measurements suffer from noise and bias resulting in a drift on the pose estimation. Even if the drift is irrelevant, it will accumulate to a significant value over time. In outdoor navigation, this issue seems to be solved by fusing IMU measurements with data from an onboard GNSS/GPS. Unfortunately, in indoor navigation, there's no possibility to use this technology for the state estimation of the UAV because of the GNSS/GPS signal which is usually degraded or totally not available. This means that Autonomous Navigation of UAVs in GPS-denied environments is still an open challenge. Over the last few years, the scientific community has focused attention on vision-based navigation thanks to the latest innovations in embedded hardware solutions, which allow us to have at the same time a high computational power and low weight to satisfy the payload constraints of aerial platforms. This Master Thesis fits into the context of the Leonardo Drone Contest, a three-year competition launched by Leonardo to motivate young researchers to improve Artificial Intelligence applied to UAVs, in which six Italian universities compete against each other. The purpose of this work is to study the SLAM State of the Art Techniques for UAVs Navigation in Critical environments. Therefore it provides a comparison between different visual-inertial algorithms to assess which one is the best solution in terms of accuracy for navigation in GPS-denied environments. The analyzed algorithms are the highly optimized proprietary VI-SLAM algorithm of the Intel T265 Tracking Camera, a semi-direct monocular VIO algorithm, SVO, an optimization-based VIO algorithm, VINS-Fusion, and the VI-SLAM Systems, ORB SLAM 2 and its improvement, ORB SLAM 3. The above-mentioned algorithms are tested along a path in a small indoor environment full of features with artificial lighting using an INTEL T265 Tracking Camera. The algorithms' performance is evaluated by the study of the localization error with respect to the ground-truth reference path. The results show that the best choice in terms of accuracy is VINS-Fusion. Nevertheless, its excellent performance requires a high level of computational resource usage. Consequently, using this algorithm onboard for state estimation needs a preliminary check on how many computational resources will remain for navigation, control, and other essential tasks.

Acknowledgements

I wish to express my heartfelt gratitude to all the people whose support was a milestone in the completion of this project. First of all, I would like to thank my supervisor, Professor Giorgio Guglieri, for giving me the opportunity to discover the challenging and inspiring world of Autonomous Navigation. His priceless dedication and humanity have been a huge inspiration to me. It was a great privilege and honor to work under his guidance. Secondly, I would like to give a special thanks to my supervisor, Dr. Simone Godio, for having been the best mentor I could have wished for. Without his patience and expertise, this project would not have been possible. Finally, I would like to thank my parents, Anna and Luciano, and my sister, Valeria, for the unconditional love they gave me through the years. I will always be grateful to them for their constant care and support during tough times. I owe my success to them.

Table of Contents

List of Tables

List of Figures

Acronyms

AI Artificial Intelligence

BA Bundle Adjustment

BRIEF Binary Robust Independent Elementary Features

DOF Degrees Of Freedom

DoG Difference of Gaussians

DRAFT DRones Autonomous Flight Team

EKF Extended Kalman Filter

EuRoC European Robotics Challenges

FAST Features from Accelerated Segment Test

GNSS Global Navigation Satellite System

GPS Global Positioning System

HALSOL High Altitude Solar

HDR High Dynamic Range

IMU Inertial Measurement Unit

KLT Kanade-Lucas-Tomasi

LIDAR Laser Imaging Detection and Ranging

MAP Maximum A Posteriori

MAV Micro Aerial Vehicle

MEMS Micro Electro-Mechanical Systems

MER Mars Exploration Rover

MSE Mean Square Error

NASA National Aeronautics and Space Administration

NCC Normalized Cross Correlation

ORB Oriented FAST and rotated BRIEF

PnP Perspective-n-Point

RANSAC RANdom SAmple Consensus

RGB-D Red Green Blue-Depth

RMSE Root Mean Square Error

ROS Robot Operating System

RVIZ Ros VIsualiZation

SAD Sum of Absolute Differences

SfM Structure from Motion

SIFT Scale-Invariant Feature Transform

SLAM Simultaneous Localization and Mapping

SSD Sum of Squared Differences

SURF Speeded Up Robust Feature

SVD Singular Value Decomposition

SVO Semidirect Visual Odometry

STD STandard Deviation

UAV Unmanned Aerial Vehicle

VIO Visual-Inertial Odometry

VINS Visual-Inertial Navigation System

VO Visual Odometry

Chapter 1

Introduction

This chapter starts providing a brief history of Unmanned Aerial Vehicles (UAVs), from the first pilotless aircraft to the modern flying robots, to understand the evolution of their application fields over time. After this historical overview, it highlights the importance of Autonomous Navigation for the current military and civil applications. One of the main challenges of this scientific field of research is that of the Autonomous Navigation of UAVs in GPS-denied environments. For this reason, after exploring some of the possible solutions in literature, the chapter presents visual-inertial navigation as the best choice for indoor environments. In conclusion, the purposes of the Master Thesis are described by placing the research into the context of the Leonardo Drone Contest.

1.1 History of Unmanned Aerial Vehicles

Unmanned Aerial Vehicles (UAVs) are a class of aircraft that don't require an onboard human pilot. They can be fully autonomous or remotely controlled from the ground. The first concept of UAVs dates back to 1849 when Austria launched around 200 hot-air balloons equipped with bombs over the city of Venice. Each balloon, designed by the Austrian artillery lieutenant Franz von Uchatius, contained about 12 kg of explosives. Fortunately, the bombardment was mostly ineffective because of the adverse weather conditions [1]. During the First World War military technology was improved on both sides of the conflict. In 1916 the first pilotless aircraft, called the Ruston Proctor Aerial Target, was developed. It was based on the revolutionary radio-controlled techniques of the engineer Archibald Low [2]. In 1917 Charles F. Kettering designed the unmanned Kettering Aerial Torpedo, commonly known as the Kettering Bug. The Bug, which used a system of pre-set inertial pneumatics and electrical controls to stabilize itself, was launched from a four-wheeled dolly. After a period of time, the engine shut off, and the wings were removed, so that the Bug, which carried about 82 kg of explosives, could fall to the

ground. Although it was considered a great success in military technology, it never saw combat because the war ended before it could be involved in a conflict [1].

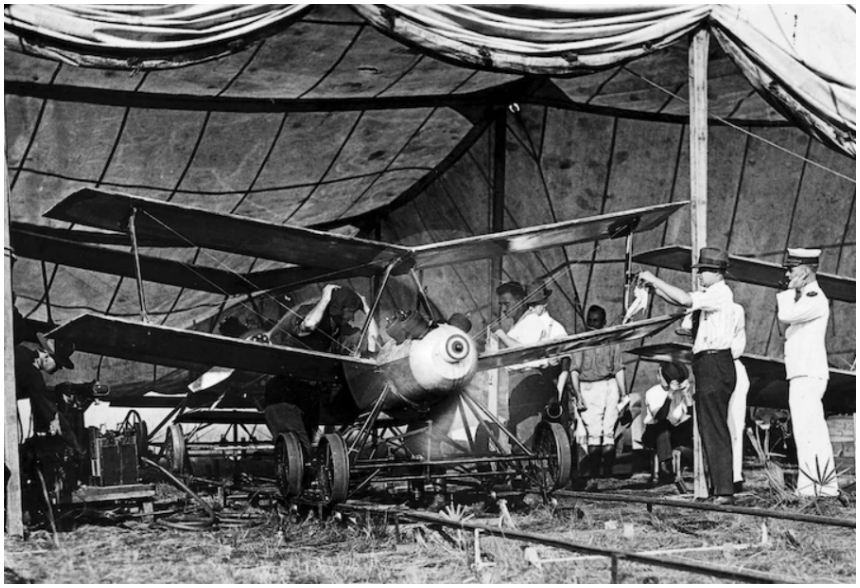

Figure 1.1: The Kettering Bug, a precursor of modern cruise missiles, was the first unmanned aerial torpedo equipped with a system of inertial pneumatics and electrical controls to stabilize itself

During the interwar period, target practice for training pilots was usually done with manned aircrafts trailing gliders as targets. Unfortunately, the above-mentioned practice was ineffective because it couldn't provide a realistic simulation of a live battle. For that reason, the military field felt the need to find a successful method for gunmen in training. It was thought about using unmanned aircrafts as aerial targets. The first pilotless aircraft meant for target practice was the DH.82B Queen Bee. The Queen Bee, derived from the De Havilland Tiger Moth biplane trainer, was conceived as a low-cost radio-controlled target aircraft. It was responsible for the introduction of the term drone in common usage [1, 3]. At the end of 1930s, the US Navy designed the Curtiss N2C-2, a revolutionary UAV that was remotely controlled by a manned aircraft that flew next to it. In the same period, the actor and aviation enthusiast Reginald Denny developed the first mass-produced drone, the OQ-2 Radioplane. Indeed, during the Second World War, the Radioplane Company produced about 15.000 of these target drones for the US Army [2]. However, the most important event of the Second World War in terms of military technology was the development of the V-1 Flying Bomb by the German Army. This aircraft, also known as Buzz Bomb or Doodlebug, was the only production aircraft equipped with a pulsejet engine instead of propellers. It was considered the

world's first cruise missile ever. In June 1944, German began the terror bombing of London with thousands of V-1 Flying Bombs, killing more than 6.000 people in a few months. After the Second World War, military innovations stalled until the Vietnam War [1].

Figure 1.2: The V-1 Flying Bomb, used in 1944 by the German for the terror bombing of London, was the world's first cruise missile ever equipped with a pulsejet for power

The Vietnam War, started in the 1950s, was a scenario of countless important innovations, from the first deployment on a large scale of reconnaissance UAVs to widespread use of unmanned aerial vehicles by the US Army to reduce pilot deaths over enemy territory. In this context, drones started to be involved in a lot of different applications, such as acting as traps in combat, launching missiles against targets, or dropping flyers for psychological warfare. In these years, the need to explore unmanned aerial technology began to interest many other countries around the world [1, 4]. In the 1960s, there was a worldwide spread of RC planes due to the improvement in the transistor technology that allowed to miniaturize them enough to be sold on the commercial market at affordable prices. In this way, aviation enthusiasts could delight themselves in flying RC planes as a hobby in both indoor and outdoor environments. This event paved the way for the development of the global consumer drone market [1]. In later years, the military field continued to focus its attention on unmanned systems technology, but it was often considered too unreliable and expensive to be used. The turning point of military drone technology arrived in 1982 when the Israeli Army decided to deploy a great number of UAVs against the Syrian Forces to triumph with minimal loss of Israeli soldiers.

In the same period, as stated by the Armed Forces Journal International, the United States admitted they carried out more than 3.435 UAV missions during the Vietnam War for decoy and reconnaissance applications [4, 5]. In the 1980s the United States launched the Pioneer UAV Program with the aim of developing an affordable and reliable drone for military purposes. For that reason, in 1986, the United States collaborate with Israel in the design of the AAI RQ-2 Pioneer, a small size composite aircraft difficult to detect either visually and on the radar. It could provide real-time reconnaissance and surveillance, target identification, and combat damage information. In the same period, the military field decided to focus its researches on renewable energy sources, such as solar power. This resulted in the development of HALSOL, the first unmanned solar-powered prototype in history [1]. In the 1990s, the United States developed the MQ-1 Predator, a long-endurance unmanned aircraft for surveillance and reconnaissance applications. Its first flight occurred in 1994. Predators, armed with AGM-114 Hellfire missiles, are still in production for the US Air Force since 1997. The Predator inspired the development of the MQ-9 Reaper [4].

Figure 1.3: The MQ-9 Reaper, a long-endurance unmanned aircraft for surveillance and reconnaissance applications, is a larger and more powerful version of the MQ-1 Predator

In 2006 the Federal Aviation Administration issued the first commercial drone permit so widespread use of drones for non-military purposes began. Governments started adopting drones for disaster relief and border surveillance while companies began using them for commercial applications such as construction inspections or agriculture. The following years saw an incredible take-off of the commercial drone industry [1, 4].

1.2 Autonomous Navigation of UAVs

Figure 1.4: Autonomous Navigation of drones is going to deeply change the world, for example, increasing human safety and sustainability

Nowadays, Unmanned Aerial Vehicles (UAVs) are being increasingly used in both military and civil applications such as intelligence, aerial surveillance and reconnaissance, firefighting, search and rescue, emergency medical aid, product deliveries, agriculture, or infrastructure inspections [6]. It's important to highlight the need for Autonomous Navigation to perform all the above-mentioned activities. In fact, the remote control couldn't always be possible for several reasons, for example, because of the unavailability of a suitable data link or because the precision required for a maneuver is beyond human capabilities [7]. One of the main challenges of Autonomous Navigation is the localization problem in GPS-denied environments. In order to autonomously navigate, performing a large number of tasks, such as motion planning and control, decision making and obstacle sensing and avoidance, the robot needs to be able to localize itself in its environment. For localization Autonomous Navigation exploits an onboard IMU which consists of a three-axis accelerometer and a three-axis gyroscope providing the linear acceleration and the angular velocity of the robot. The problem is that IMU measurements suffer from noise and bias resulting in a significant accumulated error over time on the pose estimation [8]. In outdoor navigation, this issue seems to be solved by fusing IMU measurements with data from an onboard GPS. Unfortunately, in indoor navigation, there's no possibility to use this technology for the state estimation of the UAV because of the GPS signal which is usually degraded or totally absent [9]. One of the possible solutions for the localization of UAVs in an indoor environment is represented by Vicon Motion Capture System. This system, which is used

as an high-performance localization method because of its fast frame rate and submillimeter accuracy, is composed of multiple low-latency cameras tracking the UAVs trajectory through the detection of three or more retroreflective markers that are rigidly attached to the UAVs body. If the drone is simultaneously in the field of view of multiple cameras, its 6DoF pose estimation can be determined with the highest order of positional and angular precision. The main drawback of a Vicon Motion Capture System is that it requires a calibration procedure and the environment preparation before it can be used [10].

Figure 1.5: A Vicon Motion Capture System, used in mobile robotics as an high-performance localization method, is able to achieve the state estimation of the robot with submillimeter accuracy and a measurement update frequency above 100 Hz [10]

The best solution for the UAVs localization problem in an indoor environment is that of Visual-Inertial Odometry, a sensor fusion technique, based on the fusion of the measurements from visual and inertial sensors, that improves the accuracy of the state estimation of an aerial robot in GPS-denied environments because of the complementarity of its sensors. The main advantages of using this technology are the lightness, the low-power requirement, and the cheapness of both visual and inertial sensors with respect to other technologies such as LIDAR (which is forbidden in the Leonardo Drone Contest competition) [11] and the freedom from the necessity to structure the environment before the UAVs navigation inside of it, for example, as required by Vicon Motion Capture Systems [8].

1.3 Leonardo Drone Contest

The purpose of this Master Thesis is to study the SLAM state of art techniques for UAVs navigation in GPS-denied environments by providing a comparison between different visual-inertial odometry algorithms to assess which one is the best solution in terms of accuracy. The research was conducted in an indoor environment with artificial lighting such as the competition area of the Leonardo Drone Contest. In fact, the final goal of this Master Thesis is to put this research at the disposal of the team representing the Polytechnic University of Turin for the improvement of its drone in view of the second competition of the Leonardo Drone Contest.

Figure 1.6: Leonardo Drone Contest is a three-year competition launched by Leonardo in which six Italian universities are involved in the development of an Autonomous Drone Navigation System [12]

The Leonardo Drone Contest is a three-year competition launched by Leonardo to motivate young researchers to improve Artificial Intelligence applied to UAVs, in which six Italian universities compete against one another. The six Italian universities involved in the competition are the Polytechnic University of Turin, the Polytechnic University of Milan, the University of Bologna, Sant'Anna School of Advanced Studies of Pisa, Tor Vergata University of Rome, and the University of Naples Federico II. Each university, represented by a team of students, will be involved for three years in the development and improvement of an Autonomous Drone Navigation System. The universities will challenge each other in a scientific symposium and in a competition of their drones, every year with an increasing level of difficulty. The Leonardo Drone Contest started in 2019 and will end in 2022. The first competition between the university students took place on 17 and

18 September 2020 at the Leonardo Aircraft Division in Turin. The competition area was an indoor environment of 10 by 20 meters in which a reconstruction of a city was done, resulting in a lot of cube-shaped obstacles up to three meters high and ten different landing areas. The goal was to recognize and land on five QR codes along the path for five seconds in a pre-defined sequence. The winner of the first edition of the Leonardo Drone Contest was the Polytechnic University of Milan. The autonomous system designed by the winning team reconstructed the 70% of the competition area performing autonomously three landings in a single flight. Next edition will see an increasing level of difficulty, both scientifically and technologically. There might be different moving obstacles and targets, and changing weather and fields visibility conditions.

Figure 1.7: The team representing the Polytechnic University of Turin monitoring the performance of its autonomous system during the first competition of the Leonardo Drone Contest

The Polytechnic University of Turin participates in the Leonardo Drone Contest through the student team, DRAFT PoliTO, a young and dynamic community of students from different fields and levels of engineering, represented by the Ph.D. students Simone Godio, Luigi Mascolo, Dario Riccobono and Francesco Marino and Professor Giorgio Guglieri. The team, made up of four different research groups working together towards a common goal, developed the only autonomous system equipped with 8 coaxial engines to maximize power while minimizing the size. The hardware architecture of the drone consists of the following components: an Embedded On-Board Computer (OBC) which represents the main processing system; an Electric Power System, including a battery pack and a power distribution board, for main power generation, regulation and distribution; a Propulsion

System, including BLDC motors, ESC and propellers, for flight actuation; a commercial flight controller for Propulsion System control; a sensor suite, including a rangefinder, ultrasonic sensors, and IMUs, for SLAM and proximity sensing; a monocular camera, placed below the robot, for target and TOL pad detection, and finally a stereo camera and a tracking camera, placed on the frontal part of the drone, for depth mapping and visual SLAM. The hardware specifications of the drone are presented in Table 1.1.

Leonardo Drone Contest DRAFT PoliTO Drone Specifications	
Dimensions	500 x 500 x 350 mm
Weight (Max Take-off Weight)	3300 g
Flight autonomy	15 min
Architecture	OctaQuad X8
Thrust-to-Weight ratio	2.4
Electric Power System	Li-Po battery 1200 mAh 14.8 V
Onboard computer	NVIDIA® Jetson Xavier NX™
Sensor suite board	Raspberry Pi 4 Model B
Flight controller	Pixhawk 2.4.8
Navigation cameras	Intel® RealSense™ D435 + T265
Precision landing camera	Raspberry Pi Camera Module v2
Proximity sensors	Ultrasound Adafruit® HC-SR04

Table 1.1: DRAFT PoliTO Drone Hardware Specifications and Components

Chapter 2

Visual-Inertial Odometry

Visual-Inertial Odometry (VIO) is a sensor fusion technique that performs the state estimation of an agent (e.g. an aerial platform) by using the measurements from one or more onboard cameras and Inertial Measurement Units (IMUs). This technique requires the fusion of the measurements from inertial and visual sensors because of the complementarity of their characteristics that compensate for the errors made by each of them. In fact, as stated in [13], cameras working principle of accumulating photons during the exposure time to obtain a 2D image means that they are accurate in scenes characterized by low-speed motion and provide a lot of information of the environment, which is useful for other fundamental tasks, such as place recognition. Unfortunately, at the same time, they suffer from different drawbacks such as a low output rate (up to 100 Hz), a scale ambiguity problem caused by the absence of depth information in monocular cameras, and failure in low-texture environments and in presence of motion blur and High Dynamic Range (HDR). Conversely, Inertial Measurement Units (IMUs), which consist of a three-axis accelerometer and a three-axis gyroscope providing the linear acceleration and the angular velocity of the robot, are scene-independent, so they don't suffer from the above-mentioned drawbacks of cameras. This means that they are ideal for state estimation in low-textured, high-speed, and HDR environments. In addition, their high output rate reaches up to 1000 Hz. The problem is that IMU measurements suffer from noise and bias resulting in significant accumulated error overtime on the pose estimation. For these reasons, visual and inertial sensor fusion can provide accurate localization of aerial robots in different situations. The main advantages of using Visual-Inertial Odometry are the high accuracy, low-power requirements, small size, and cheapness of both visual and inertial sensors [14]. This chapter provides an overview of Visual Odometry and Inertial Measurement Unit, describing how visual and inertial navigation individually perform the pose estimation of an agent. In conclusion, the main Visual-Inertial Odometry paradigms are presented to provide the reader with the basics to understand how the algorithms described

in the next chapter are implemented.

2.1 Visual Odometry

Visual Odometry (VO) is a method for the ego-motion estimation of an agent (e.g. an aerial robot) based on the examination of a sequence of images acquired from a single or multiple onboard cameras [15]. This technique is considered a subset of Simultaneous Localization and Mapping (SLAM), a process in which a robot simultaneously builds a map of its environment and localizes itself within the map. The main difference between the two techniques is that, while Visual Odometry incrementally estimates the pose of the robot focusing on local consistency, the SLAM purpose is to obtain global consistency of the path through loop closing, which consists in recognizing previously mapped locations to reduce the accumulated drift on the pose estimation [16]. There are three main different Visual Odometry approaches: feature-based methods, based on the extraction and matching (or tracking) of features over the frames for motion estimation, appearance-based (or direct) methods, based on the intensity information of all the pixels of the images, and hybrid methods, a combination of the previous two approaches. Feature-based methods require the ability of extracting and matching features among the images but have higher speed and accuracy with respect to appearance-based methods. For this reason, the largest part of Visual Odometry pipelines is feature-based [15]. The implementation of Visual Odometry can be done with different types of sensors, for example, monocular, stereo, omnidirectional or RGB-D cameras [17]. The assumptions required for optimal performance of Visual Odometry are the presence of enough lighting and texture in the environment, the dominance of static scene over dynamic objects, and sufficient scene overlap between adjacent frames [15].

2.1.1 History of Visual Odometry

In this section, a historical overview of the early years of research in the field of Visual Odometry is presented, describing some of the most important pioneering works in this area. The problem of the ego-motion estimation of an agent from visual input was first addressed in the 1980s by Moravec at Stanford University [16]. Moravec used a planetary rover equipped with a *slider stereo*, a monocular camera sliding on a rail, in a stop-and-go system. At each stop, the camera would slide horizontally on the rail capturing a total of 9 frames at equidistant intervals. Then

the robot would start a process of features extraction and matching between all the images acquired from the camera. The features matching was done using the Normalized Cross Correlation (NCC), a similarity measure used to compare the local appearance of the corresponding features. This process allowed the reconstruction of the 3D structure of the environment. The motion estimation was obtained by aligning the reconstructed 3D points observed from consecutive robot positions. This research was a milestone for the computer vision community not only because it presented the first Visual Odometry pipeline but also because it described one of the earliest corner detectors, called the Moravec corner detector, a predecessor of Harris detector [15]. In later years, Matthies and Shafer in [18] improved the above-mentioned work using an error model based on 3D Gaussian distributions instead of the scalar model used in Moravec's implementation [16]. The term Visual Odometry was chosen by Nister et al. in [19] for its similarity to wheel odometry. In fact, as wheel odometry incrementally estimates the robot path through the integration of the number of turns of its wheels, Visual Odometry incrementally estimates the pose of the robot through the analysis of the changes that motion induces in the pictures captured from its onboard cameras [15]. Nister et al. introduced a lot of innovations in the panorama of Visual Odometry. For example, they first addressed the problem of motion estimation in presence of outliers, proposing a RANSAC scheme for outlier removal and introduced a highly accurate RANSAC-based motion estimation procedure using the 3D to 2D reprojection error instead of the Euclidean distance error between 3D points for both monocular and stereo Visual Odometry implementations [16]. Most of the research in Visual Odometry is well-known for its application in the Mars space mission, started in 2003 and still in progress, in which NASA's twin Mars Exploration Rovers (MERs), Spirit and Opportunity, were sent on Mars to explore the planet geology [16, 20]. Another pioneering research was that of Scaramuzza and Siegwart who presented in [21] a monocular omnidirectional Visual Odometry implementation for outdoor navigation of ground vehicles proposing a fusion of motion estimation from two different approaches, a feature-based method and an appearance-based method [16]. Later, Kaess et al. in [22] introduced a novel approach to Visual Odometry based on sparse flow separation in which the sparse flow was separated into flow of close features and flow of distant features by disparity based on a threshold depending on the vehicle speed. The set of distant features was involved in recovering the rotation component with a two-points algorithm and the set of close features was involved in recovering the translation component with a one-point algorithm. The reason for adopting a sparse flow separation was that small changes in the robot translation have an imperceptible influence on distant features [16]. Finally, Alcantarilla et al. in [23] introduced the sparse flow separation method in their Extended Kalman Filter Simultaneous Localization and Mapping (EKF-SLAM) implementation with the aim of improving the robustness and the accuracy of their model [16].

2.1.2 Formulation of the Visual Odometry problem

In this section, the formulation of the Visual Odometry problem is presented, as discussed in [15]. Imagine a robot that is capturing pictures with a rigidly-attached onboard camera at discrete time instants k while moving in the environment. For a monocular camera, the set of pictures captured at discrete times k is indicated by the relation:

$$I_{0:n} = \{I_0, ..., I_n\} \tag{2.1}$$

For a stereo camera, the sets of right and left pictures captured at every time instant are denoted by the relations:

$$\begin{aligned} I_{r,0:n} &= \{I_{r,0}, ..., I_{r,n}\} \\ I_{l,0:n} &= \{I_{l,0}, ..., I_{l,n}\} \end{aligned} \tag{2.2}$$

In this particular case, the origin is assumed to match the coordinate frame of the left camera, as showed in Figure 2.1.

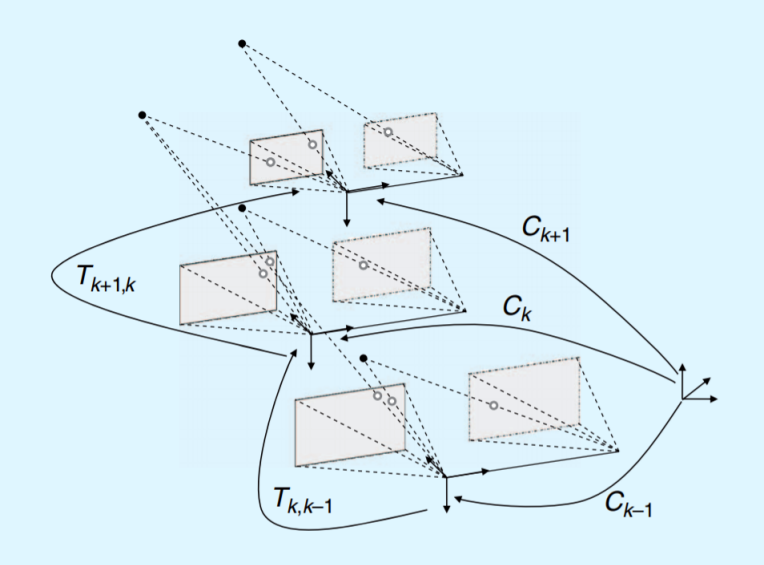

Figure 2.1: Illustration of the Visual Odometry problem. The relative poses $T_{k,k-1}$ between consecutive camera frames are computed performing feature matching. The global poses C_k with respect to the initial coordinate frame are obtained by concatenating the relative tranformations.

Two consecutive camera positions at time instants k and k-1 are related by the following rigid body transformation, $T_{k,k-1}$:

$$T_{k,k-1} = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix} \tag{2.3}$$

where $R_{k,k-1}$ is the rotation matrix and $t_{k,k-1}$ is the translation vector. The set

$$T_{1:n} = \{T_1, ..., T_n\} \tag{2.4}$$

contains all the subsequent relative transformations. For the sake of simplicity, from this point on, T_k will be used instead of $T_{k,k-1}$. Finally, the set of camera poses

$$C_{0:n} = \{C_0, ..., C_n\} \tag{2.5}$$

contains all the transformations of the camera with respect to the initial coordinate frame, positioned on the right in Figure 2.1. In particular, C_0 is the initial camera pose with respect to the reference frame, which can be arbitrarily set by the user. The final camera pose C_n can be computed by concatenating all the previous relative transformations $T_{1:n}$ as follows:

$$C_n = C_{n-1}T_n \tag{2.6}$$

The main goal of Visual Odometry is to estimate the global camera path $C_{0:n}$ through a concatenation of all the relatives transformations $T_{1:n}$. This means that Visual Odometry incrementally estimates the camera trajectory.

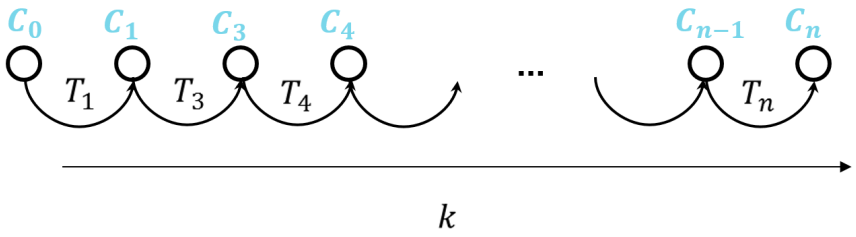

Figure 2.2: Illustration of the global camera path - The camera trajectory C_n is estimated incrementally by concatenating all the relative transformations T_k pose after pose

2.1.3 Fundamentals of Visual Odometry

As mentioned in the introduction, there are three main different Visual Odometry approaches: the feature-based approach, based on the extraction and matching (or tracking) of features over the frames for motion estimation, the appearance-based (or direct) approach, based on the intensity information of all the pixels of the images and the hybrid approach, a combination of the previous two approaches. The feature-based approach requires the ability of extracting and matching features among the images but has higher speed and accuracy with respect to appearance-based methods. For this reason, in this section, a detailed description of the feature-based Visual Odometry approach is provided. The feature-based Visual Odometry pipeline is a block diagram composed of the following four fundamental steps:

1. Feature detection
2. Feature matching (or tracking)
3. Motion estimation
    - 3-D-to-3-D
    - 3-D-to-2-D
    - 2-D-to-2-D
4. Pose optimization

Feature detection and matching consist in selecting points of interest and finding image correspondences across all the frames captured from the onboard camera. Image correspondences are considered the image features of different frames that are the reprojection of the same 3-D point of the environment. There are two different approaches to perform the above-mentioned activity: feature matching and feature tracking. The first one is the process of individually extracting features in all frames and matching those features according to a similarity metric by exploiting feature descriptors. This approach is particularly useful when a large motion or viewpoint change is expected between two consecutive frames. Conversely, the second one is the process of extracting features in the first frame and tracking those features in the following frames using local search techniques, such as correlation. This approach is useful when a small change in motion or viewpoint is expected between two consecutive frames. In the last decades, the majority of the research in the field of Visual Odometry preferred the feature matching approach because the focus has shifted to large-scale environments in which significant changes in the

appearance of features can be noticed because of the observation of those features over longer sequences. In the following, every step of the feature-based Visual Odometry pipeline is described in detail.

2.1.3.1 Feature detection

The first step of the Visual Odometry pipeline is that of feature detection. In this step, the search of the points of interest is performed in the image by a feature detector. There are three different types of features: corners, edges, and blobs. A corner is defined as a point at the intersection of two or more edges where image intensity has a significant variation in all directions. An edge is defined as a segment through which rapid change in the image intensity can be observed. And finally, a blob is defined as an image pattern that differs from its neighborhood in terms of intensity, colors, and texture. A feature detector is an identification algorithm that includes two different steps. The first one consists in applying a feature-response function on the image, such as the corner response function in the Harris detector or the difference-of-Gaussian (DoG) operator in the SIFT detector. The second one consists in applying non-maxima suppression on the output of the first step in order to identify all the local maxima of the feature-response function. The output of the non-maxima suppression represents the extracted features. As discussed in [15], an ideal feature detector must have the following properties: localization accuracy, repeatability, computational efficiency, robustness, distinctiveness, and invariance to both photometric (e.g. illumination) and geometric changes (e.g. rotation, scale, and distortion). Feature detectors can be divided into corner detectors (such as Moravec, Harris, and FAST) and blob detectors (such as SIFT and SURF). Every feature detector has its advantages and disadvantages. For example, corner detectors are faster to compute but less distinctive compared to blob detectors. For this reason, the choice of the feature detector must be done according to several factors such as the computational constraints, the type of the environment, and the motion baseline depending on the particular situation. After the first step, feature description can be performed in the image on the extracted features. In this step, the area around an extracted feature is converted into a feature descriptor, assigning the feature a distinctive measure that improves the accuracy of the feature matching. The simplest feature descriptor is the local appearance of the feature, measured considering the intensity of the pixels around the feature point. Unfortunately, the local appearance of the feature changes if there is a variation for example in orientation, scale, and viewpoint. For this reason, most feature descriptors currently used in literature, such as SIFT and SURF, are invariant to rotation, scale, illumination, and viewpoint changes. In the following, the most common detectors and descriptors used in Visual Odometry, that is Harris, FAST, SIFT, SURF, and BRIEF, are briefly described.

Harris Corner Detector

Harris Corner Detector, based on Moravec Corner Detector, was first introduced by Harris and Stephens in 1988 in their paper "A Combined Corner and Edge Detector" [24]. In this case, the evaluation of the distinctiveness is performed by measuring the amount of change that occurs in the intensity of the pixels when moving every pixel window by a displacement (u,v) in a given direction [25]. The aforementioned measurament is done by computing the sum squared difference (SSD) of the intensity of the pixels before and after the shift using the change function $E(u, v)$:

$$E(u, v) = \sum_{x,y} w(x, y)[I(x + u, y + v) - I(x, y)]^2 \tag{2.7}$$

where $w(x, y)$ is the window function which is a rectangular or Gaussian function that assigns weights to the pixels of the window. The extracted features of the image are those pixels whose value of the change function $E(u, v)$ exceeds a determined threshold in all directions. The feature detection is performed by maximizing the change function $E(u, v)$. Applying Taylor Expansion to the second term of the function, after some mathematical steps, we get the final equation:

$$E(u, v) = \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix} \tag{2.8}$$

where the matrix M is defined as:

$$M = \sum_{x,y} w(x, y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix} \tag{2.9}$$

The eigenvectors of the matrix M indicate the directions for both the largest and smallest increases in SSD. The corresponding eigenvalues indicate the amount of the above-mentioned increases. For each window it is possible to compute a score, R:

$$R = det(M) - k(trace(M))^2 \tag{2.10}$$

where:

- $det(M) = \lambda_1 \lambda_2$
- $trace(M) = \lambda_1 + \lambda_2$

This means that the magnitude of the eigenvalues of M establishes whether a region is a corner, edge or flat. In fact:

- When $|R|$ is small, which means that λ_1 and λ_2 are both small, the region is flat.
- When $R < 0$, which means that $\lambda_1 >> \lambda_2$ or vice versa, the region is an edge.
- When R is large, which means that λ_1 and λ_2 are both large, the region is a corner.

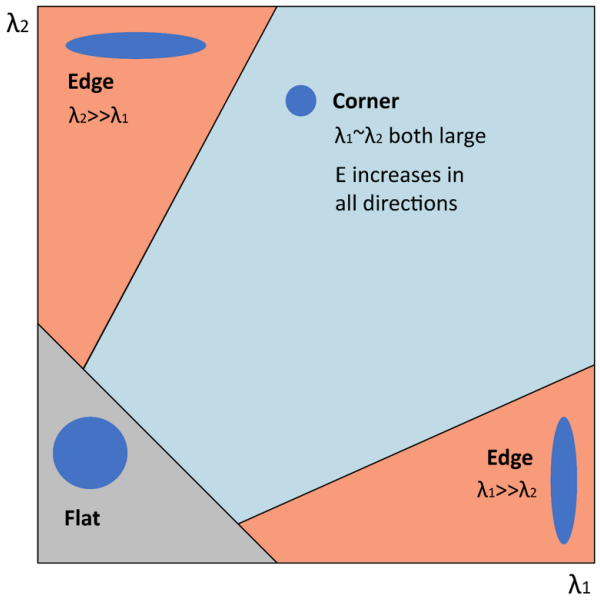

Figure 2.3: Harris Corner Detector - Classification of image points using the eigenvalues of the matrix M

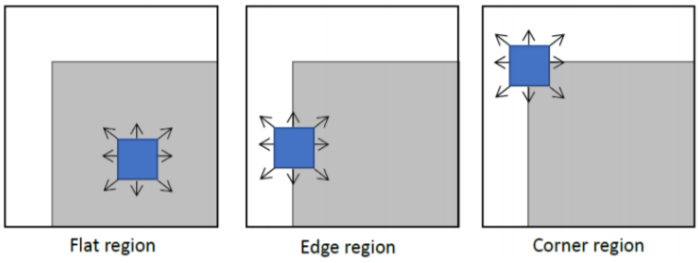

Figure 2.4: Representation of the Harris window on different regions

FAST Corner Detector

FAST (Features from Accelerated Segment Test) is a corner detector that was first introduced by Rosten and Drummond in 2006 in their paper "Machine Learning for High-Speed Corner Detection" [26]. It is one of the most computationally efficient feature extraction methods. In fact, it is considered ideal for real-time applications because of its high-speed performance. This corner detector is based on the SUSAN method [27] in which a circumference of 16 pixels is placed around the center pixel. The intensity of each pixel of the circumference is compared to that of the center pixel. The number of pixels belonging to the circumference that have an intensity comparable to that of the center pixel decides whether a region is a corner, an edge or flat [16]. In the following, the FAST Corner Detector algorithm is described in detail.

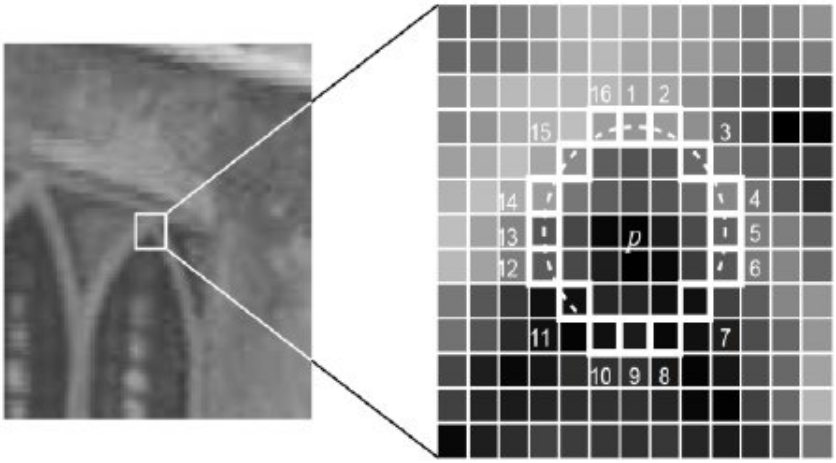

Figure 2.5: FAST Corner Detector illustration. In the image p is the candidate corner, white-highlighted squares are the 16 pixels used in the feature detection, and finally the dashed line is a line that passes through 12 consecutive pixels that are brighter than the center pixel

The first step of the algorithm is the selection of a candidate pixel p, whose intensity is denoted by I_p, in the image above. After the first step, a Bresemham circumference of radius 3 including 16 pixels of the image is placed around the central pixel p to classify whether the candidate point is actually a corner. The pixels of the circumference are denoted by integer numbers from 1 to 16. The candidate pixel p is classified as a corner if there exists a set of 12 contiguous pixels in the circumference which are all brighter than $I_p + t$ or all darker than

$I_p - t$, where t is a settled threshold. A high-speed test could be applied to remove non-corner points. The high-speed test operates by examining the intensity of four example pixels (1,5,9 and 13) of the circumference. The candidate point can be a corner if and only if at least three of the four pixels satisfy the above-mentioned threshold criterion. The test starts by examining the intensity of pixels 1 and 9. If the threshold criterion is not satisfied for one or both pixels, it is certain that the candidate point p is not a corner. Else if the threshold criterion is verified for both pixels, the intensity of pixels 5 and 13 must be examined to check if there are at least three of the four pixels that satisfy the criterion. If the previous condition is verified, the full segment test criterion can then be applied to all the 16 pixels of the circumference to draw the final conclusion. As mentioned before, the candidate point p is classified as a corner if there are 12 contiguous pixels in the circumference that fall in the established threshold criterion. The procedure must be repeated for all the pixels in the image. The main advantage of this method is a high computational efficiency. At the same time, this algorithm presents a lot of weaknesses, first of all, a great sensitivity to noise.

SIFT Feature Detector and Descriptor

SIFT (Scale Invariant Feature Transform) is a feature detector and descriptor which was first introduced by Lowe in 2004 in his paper "Distinctive Image Features from Scale-Invariant Keypoints" [28]. This algorithm is considered one of the most robust feature extraction techniques available in literature mainly because of its invariance to rotation, scale, illumination, and viewpoint changes. The SIFT algorithm is articulated in four fundamental steps. The first step consists in estimating scale-space extrema using a Difference of Gaussian (DoG) function that searches over all scales and image locations for potential scale and orientation invariant interest points. The second step consists in keypoint localization in which a detailed model is applied to the keypoint candidates in order to determine their location and scale. After this process, a keypoint refinement is performed to remove low contrast points. The third step consists in a keypoint orientation assignment based on local image gradient. In conclusion, the final step consists in feature description, based on image gradient magnitude and orientation, that is robust to significant levels of local shape distortion and change in illumination [29]. The main drawback of this algorithm is that it is computationally expensive. In the following, a qualitative description of the feature detection and description steps is provided. Considering the illustration of Figure 2.6, the original image placed at the top left of (a) is smoothed four times with a Gaussian filter characterized by four different standard deviations. This process is repeated several times after downsampling the image of factor two. At this point, the Difference of Gaussian

(DoG) images are obtained as the difference between successive Gaussian-blurred images. Finally, feature detection is performed by finding the local minima or maxima of Difference of Gaussian (DoG) images across scales and space [30].

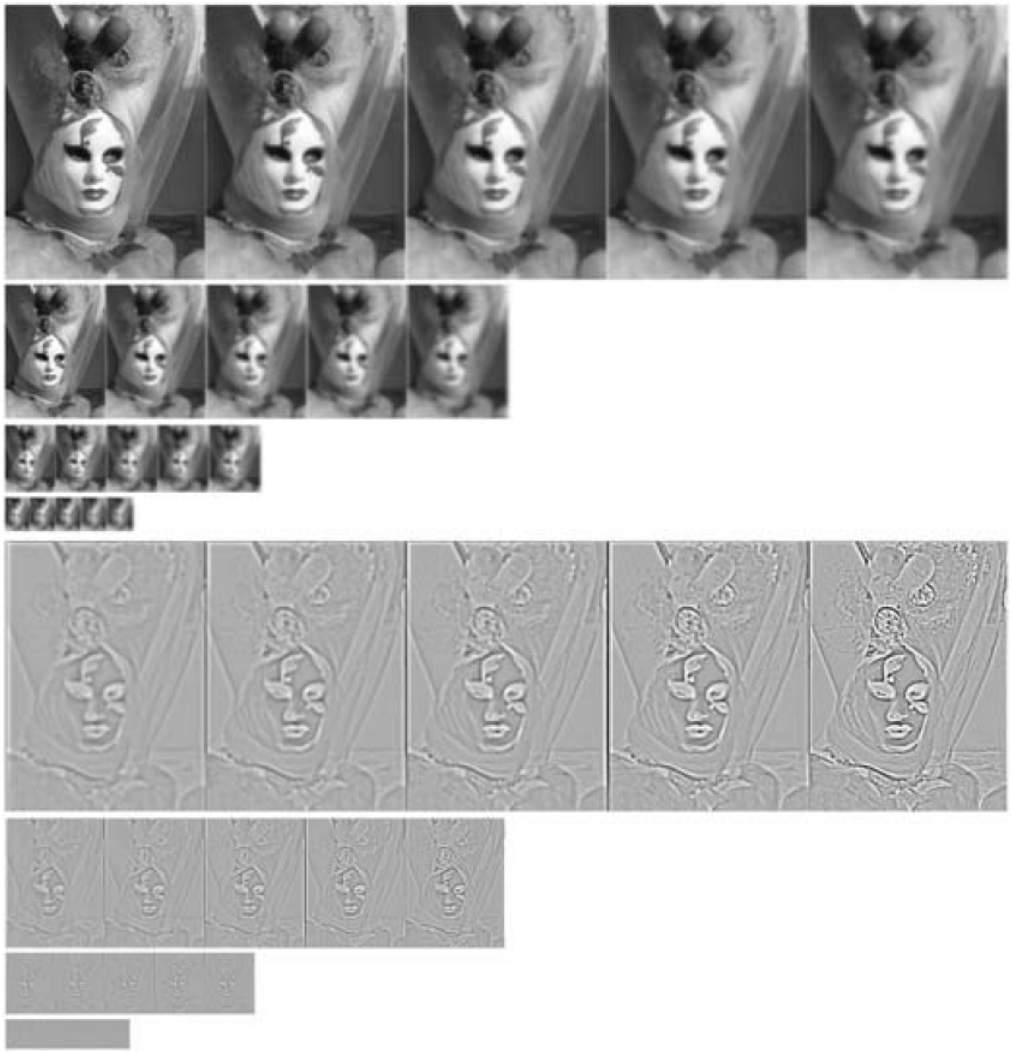

Figure 2.6: SIFT Feature Detection consists in finding the local minima or maxima of the output of the convolution of the upper and lower scales of the image with a Difference of Gaussian (DoG) operator across scales and space

The feature description step consists in the assignment of a feature descriptor to all the extracted features. The SIFT descriptor is a histogram of local gradient orientations. The region around the keypoints is divided into a 4x4 grid. For each subregion, a histogram of eight gradient orientations is constructed. The information is stored in a 4 x 4 x 8 = 128 elements description vector. The SIFT descriptor can be computed either for corners or blobs. However, the performance is better on blobs than on corners because corners are defined as the points at the intersection of edges. For this reason, corner descriptors are necessarily less distinctive compared to blob descriptors, which are placed in highly textured areas of the picture [30]. In Figure 2.7 an example of the output of this feature extraction technique is presented, showing the orientation and scale of each extracted feature.

Figure 2.7: An illustration of the output of the SIFT feature extraction technique showing the orientation and scale of each extracted feature

SURF Feature Detector and Descriptor

SURF (Speeded Up Robust Features) is a scale and rotation invariant feature detector and descriptor which was first presented by Bay et al. in their paper "SURF: Speeded Up Robust Features" at the European Conference on Computer

Vision in 2006 [31]. This method has a lot of points in common with the SIFT feature detector and descriptor. At the same time, there are several differences that let SURF outperform SIFT in terms of speed and computational efficiency. This is achieved by employing box filters for the approximation of the Gaussian and image integrals for the computation of the image convolution in order to approximate the Hessian matrix. The keypoint detection is performed by analyzing the determinant of the approximated Hessian matrix to find the local minima or maxima across the scales. The feature description algorithm is largely inspired by the previously proposed SIFT scheme. The dominant orientation of the points of interest is estimated by calculating the sum of all Haar wavelet responses in both x- and y-directions in a circular region around the keypoint. The description of the region around the point of interest is performed by extracting a square region, centered on the point and oriented along the previously assigned orientation. This region is then divided into 4x4 subregions in which the Haar wavelet responses are computed at 5x5 regularly spaced sample points and weighted with a Gaussian kernel to obtain a robust performance against, for example, deformation and noise. This results in a 64 elements description vector against the 128-D vector of the SIFT descriptor. This is one of the reasons because SURF has a better performance than SIFT in terms of computational efficiency [16].

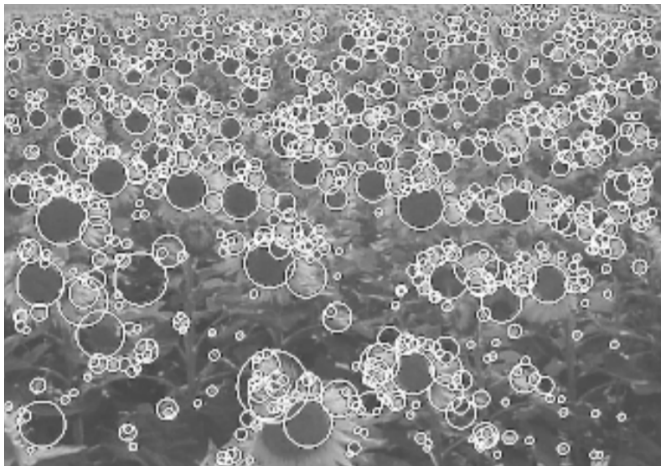

Figure 2.8: An illustration of the output of the SURF feature extraction technique showing the orientation and scale of the detected features

BRIEF Feature Descriptor

BRIEF (Binary Robust Independent Elementary Features) is the first binary feature descriptor introduced in 2010 by Calonder et al. in their paper "BRIEF: Binary Robust Independent Elementary Features" [32]. This descriptor creates a binary feature vector that describes the patch around a keypoint by using a relatively small number of pairwise intensity comparisons. For this reason, BRIEF easily outperforms the previous descriptors in terms of speed and recognition rate in absence of large in-plane rotation changes. A detailed representation of the feature description process is provided in [33]. The descriptor, operating on the image at the pixel level, is highly noise-sensitive. Consequently, the reduction of this noise sensitivity is performed by smoothing the image with a Gaussian kernel, as shown in the figure below.

Figure 2.9: Before and after the performance of the image smoothing with a Gaussian kernel to reduce the noise sentitivity of BRIEF

This process results in the improvement of the stability and repeatability of the descriptors. The goal is that of creating a binary feature vector of the binary test responses, where the binary test is defined as:

$$\tau(p, x, y) = \begin{cases} 1 & : p(x) < p(y) \\ 0 & : p(x) \geq p(y) \end{cases} \tag{2.11}$$

where p(x) and p(y) are respectively the intensity of the patch at the pixels x and y. Choosing a set of n (x,y)-location pairs, where n is the length of the binary feature vector, uniquely defines a set of binary tests. The random pairs that need to be chosen for creating the binary feature vector are selected in the patch by using one

of the five possible sampling geometries, shown in the figure below.

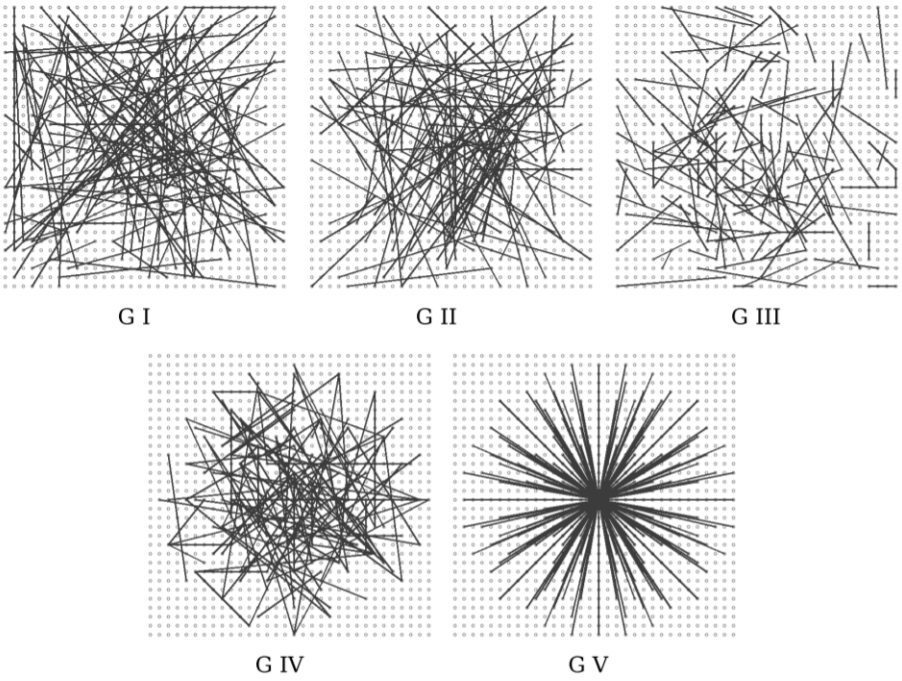

Figure 2.10: An illustration of the five possible sampling geometries for the selection of the random pairs (x,y) needed to create the binary feature vector

- G I: the pixels of the random pair are both sampled from a uniform distribution or spread of S/2 around the keypoint.
- G II: the pixels of the random pair are both sampled from a Gaussian distribution or spread of $0.04 \cdot S^2$ around the keypoint.
- G III: the first pixel of the random pair (x) is sampled from a Gaussian distribution centered around the keypoint with a standard deviation or spread of $0.04 \cdot S^2$ and the second pixel of the random pair (y) is sampled from a Gaussian distribution centered around the first pixel with a standard deviation or spread of $0.01 \cdot S^2$.
- G IV: the pixels of the random pair are both drawn from discrete locations of a coarse polar grid introducing a spatial quantization.
- G V: the first pixel of the random pair (x) is at (0, 0) and the second pixel of the random pair (y) is sampled from discrete locations of a coarse polar grid.

The BRIEF descriptor needs to be used in conjunction with a feature detector such as Harris, FAST, SIFT, and SURF. In recent years, the ORB (Oriented FAST and Rotated BRIEF) feature detector and descriptor, based on FAST detector and BRIEF descriptor, has been introduced by Rublee et al. in their paper "ORB: An efficient alternative to SIFT or SURF" [34]. In the following, a comparison between the different feature extraction techniques described in this section is provided.

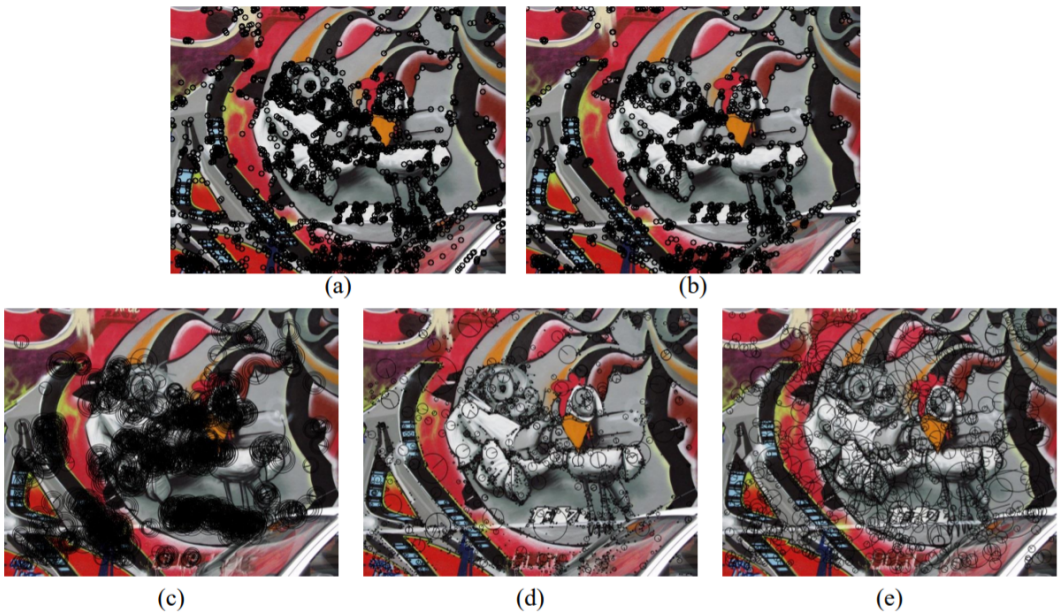

Figure 2.11: A comparison of the different features extraction techniques described in this section using an image obtained from the Oxford dataset [35]: (a) FAST (b) HARRIS, (c) ORB, (d) SIFT (e) SURF [16].

2.1.3.2 Feature matching (or tracking)

The second step of the Visual Odometry pipeline is feature matching (or tracking). The feature matching approach consists in independently extracting features in all frames and searching for correspondences across the frames by comparing feature descriptors on the basis of a similarity metric. This approach is particularly useful when a large motion or viewpoint change is expected between two consecutive frames. In the event that the feature descriptor is the local appearance of the feature then data association is based on visual similarity performed with the Sum of Squared Differences (SSD) or the Normalized Cross Correlation (NCC), which are described in detail in [16].

Sum of Square Differences (SSD)

The Sum of Squared Differences (SSD) is a similarity measure computed by using the squared differences between corresponding pixels in two different frames. This measure is expressed through the following equation:

$$SSD = \sum_{i=-n}^{n} \sum_{j=-n}^{n} [I_1(u_1 + i, v_1 + j) - I_2(u_2 + i, v_2 + j)]^2 \tag{2.12}$$

where I_1 and I_2 are the first and second image patches centered at (u_1, v_1) and (u_2, v_2) respectively. Alternatively to the Sum of Squared Differences (SSD) it is possible to use the Sum of Absolute Differences (SAD) which differs from the previous approach in the computation of the absolute differences instead of the squared differences between corresponding pixels in two different frames. The Sum of Absolute Differences (SAD) has a greater computational efficiency but less robustness compared to the Sum of Square Differences (SSD).

Normalized Cross Correlation (NCC)

The Normalized Cross Correlation (NCC) is one of the most common similarity measures in literature computed by normalizing the sum of the product of the corresponding pixels intensities by a factor based on the intensity of the above-mentioned pixels. This measure is expressed through the following equation:

$$NCC = \frac{\sum_{i=-n}^{n} \sum_{j=-n}^{n} I_1(u_1 + i, v_1 + j) \cdot I_2(u_2 + i, v_2 + j)}{\sqrt{\left(\sum_{i=-n}^{n} \sum_{j=-n}^{n} I_1(u_1 + i, v_1 + j)^2\right) \cdot \left(\sum_{i=-n}^{n} \sum_{j=-n}^{n} I_2(u_2 + i, v_2 + j)^2\right)}} \tag{2.13}$$

The Normalized Cross Correlation (NCC) is an index of similarity, this means that a high value of this index corresponds to a great similarity between two different image patches. This method outperforms the previous approaches in presence of changes in brightness between the compared images but it is slower because it deals with complex operations such as multiplications and divisions.

Conversely, when the feature descriptor is not based on the local appearance of the feature, such as SIFT, the most common similarity measure used in literature is the Euclidean distance. The process of feature matching results in the choice of the best correspondence of a feature in the second frame as the feature with the closest descriptor, for example, in terms of similarity. The problem is that the features of the second frame could match with more than one feature in the first frame. This problem is solved by performing the mutual consistency check which consists in mutually matching features of the first frame with features of the second frame and vice versa in order to choose only pairs of correspondences that mutually have each other as a preferred match. Unfortunately, this approach is computationally expensive because its computational cost is quadratic with the number of features. For this reason, when the number of features is large, it is necessary to consider a computationally efficient method. The best approach is the constrained matching which consists in searching for corresponding features in predicted regions of the second frame. The predicted regions are calculated as an error ellipse from the uncertainty of the motion model and the 3-D feature position. If the 3-D position of the features is not available, it is possible to perform the epipolar matching, a process that consists in searching for a feature correspondence along the epipolar line of the second frame. The epipolar line is the straight line at the intersection between the epipolar plane, defined by a feature point and the two camera centers, and the image plane, as shown in Figure 2.12 in the Motion Estimation section [30]. The feature tracking approach consists in extracting features in the first frame and then tracking this set of features across the next frames using a local search technique such as correlation. This approach is suitable for Visual Odometry applications in which a small change in motion or viewpoint is expected between two consecutive frames. In this particular case, a visual similarity measure such as the Sum of Squared Differences (SSD) or the Normalized Cross Correlation (NCC) can be used. Otherwise, when the feature tracking is performed over long image sequences, an affine distortion model, resulting in the Kanade–Lucas–Tomasi (KLT) feature tracker, is required for each feature.

KLT Feature Tracker

The Kanade-Lucas-Tomasi (KLT) feature tracker is a feature tracking algorithm, based on the early work of Lucas and Kanade [36], that was first presented by Kanade and Tomasi in the technical report "Detection and Tracking of Point Features" [37]. In the following, the problem statement for this feature tracker is proposed. Let's assume that I and J are the first and the second grayscaled images in which feature tracking will be performed. The functions I(x,y) and J(x,y) are respectively the intensity values of I and J at $[x\ y]^T$. Defined $\mathbf{u} = [u_x\ u_y]^T$ as the

point of the first image I to be tracked on the second image, the goal is to find $\mathbf{v}$ on the second image J, where I($\mathbf{u}$) and J($\mathbf{v}$) are similar. Remember that $\mathbf{v} = \mathbf{u} + \mathbf{d}$ where $\mathbf{d} = [d_x \; d_y]^T$ is the image velocity (or optical flow) at $\mathbf{u}$. This process is equivalent to minimize the residual function, $\varepsilon(\mathbf{d})$:

$$\varepsilon(\mathbf{d}) = \sum_{x=u_x-\omega_x}^{u_x+\omega_x} \sum_{y=u_y-\omega_y}^{u_y+\omega_y} [I(x,y) - J(x+d_x, y+d_y)]^2 \tag{2.14}$$

where the integration window size is $(2\omega_x + 1) \times (2\omega_y + 1)$. The distribution of the features in the frames influences the accuracy of the motion estimation results. The main requirements for robust and stable motion estimation are a large number of features and a uniform distribution of features in the image. These requirements can be satisfied by applying the feature detector to every cell of the grid in which the image has been partitioned until a minimum threshold of features is reached in every subregion of the image [19]. A general rule is to set the minimum threshold to 1000 features for a 640×480 pixel image [30].

2.1.3.3 Motion estimation

The most important step of the Visual Odometry pipeline is motion estimation. The motion estimation step consists in computing the relative transformation T_k between the previous image I_{k-1} and the current image I_k from the sets of corresponding features f_{k-1} and f_k at consecutive time instants. The full trajectory of the agent is recovered by concatenating all the relative transformations computed during the motion estimation step. There are three different approaches of motion estimation depending on the dimensions of the feature correspondences:

- 3-D-to-3-D, in which the sets of corresponding features f_{k-1} and f_k are both specified in 3-D.
- 3-D-to-2-D, in which the features f_{k-1} are specified in 3-D and the features f_k are their corresponding 2-D reprojections on the image I_k.
- 2-D-to-2-D, in which the sets of corresponding features f_{k-1} and f_k are both specified in 2-D image coordinates.

3-D-to-3-D Motion Estimation

The 3-D-to-3-D Motion Estimation is performed from corresponding 3-D-to-3-D features obtained by triangulating the 3-D points from the 2-D image correspondences at every time instant. The camera motion is computed by determining the aligning transformation of the two sets of 3-D features. The transformation is estimated by minimizing the 3-D Euclidean distance between the corresponding 3-D features as shown in the following equation:

$$T_k = \underset{T_k}{\operatorname{argmin}} \sum_i \|\tilde{X}_k^i - T_k \tilde{X}_{k-1}^i\| \tag{2.15}$$

where $\tilde{X}_k$ and $\tilde{X}_{k-1}$ are the corresponding 3-D-to-3-D feature points and i is the minimum number of feature points required to constrain the transformation. The minimum number of feature pairs is chosen according to the systems degrees of freedom (DOF) and the model used for motion estimation. In this case, the minimal solution involves three 3-D-to-3-D corresponding feature points. In the following, the algorithm for the 3-D-to-3-D motion estimation, presented in [15], is described.

Algorithm 1 3-D-to-3-D Motion Estimation Algorithm

1: Capture two image pairs $I_{l,k-1}$, $I_{r,k-1}$ and $I_{l,k}$, $I_{r,k}$ with a stereo camera
2: Extract and match features between the consecutive left frames $I_{l,k-1}$ and $I_{l,k}$
3: Triangulate the matched features for each stereo pair
4: Compute the transformation T_k from the 3-D-to-3-D corresponding feature points X_{k-1} and X_k
5: Estimate the current pose by computing $C_k = C_{k-1}T_k$
6: Repeat the entire procedure from 1

3-D-to-2-D Motion Estimation

The 3-D-to-2-D Motion Estimation, known as Perspective-n-Point (PnP), is performed from corresponding 3-D-to-2-D features X_{k-1} and p_k. The 3-D feature point X_{k-1} can be triangulated, in the stereo case, from stereo image pairs and, in the monocular case, from the sets of features p_{k-1} and p_{k-2} thus using image correspondences across three frames for motion estimation. The transformation is estimated by minimizing the image reprojection error as shown in the equation below:

$$T_k = \underset{T_k}{\operatorname{argmin}} \sum_i \|p_k^i - \hat{p}_{k-1}^i\|^2 \tag{2.16}$$

where p_k^i is the observed feature point in the current frame I_k and $\hat{p}_{k-1}^i$ is the reprojection of the 3-D point X_{k-1}^i into the frame I_k after applying the transformation T_k. The minimal case solution, called Perspective-3-Point (P3P), involves three 3-D-to-2-D feature correspondences. In the following, the algorithm for the 3-D-to-2-D motion estimation, presented in [15], is described.

Algorithm 2 3-D-to-2-D Motion Estimation Algorithm

1: Do only once:
  1.1: Capture two frames I_{k-2} and I_{k-1}
  1.2: Extract and match features between the captured frames
  1.3: Triangulate the matched features from the captured frames
2: Do at every iteration:
  2.1: Capture a new frame I_k
  2.2: Extract and match features with the frame I_{k-1}
  2.3: Compute the transformation T_k from the 3-D-to-2-D correspondences
  2.4: Triangulate new feature matches between I_k and I_{k-1}
  2.5: Repeat from 2.1

2-D-to-2-D Motion Estimation

The 2-D-to-2-D Motion Estimation is performed from 2-D-to-2-D feature correspondences in those situations in which the triangulation of the 3-D points is not possible, for instance, in the monocular case for the estimation of the transformation between the first frames. This approach, which is based on the epipolar geometry shown in Figure 2.12, uses the epipolar constraint for the computation of the transformation T_k between two consecutive camera frames. The epipolar constraint is formulated by the following equation:

$$\tilde{p'}^T E \tilde{p} = 0 \tag{2.17}$$

where $\tilde{p}$ and $\tilde{p'}$ are the feature correspondences in two consecutive frames and E is the essential matrix, expressed by the relationship:

$$E \simeq \hat{t}_k R_k \tag{2.18}$$

where R_k is the rotation matrix and $\hat{t}_k$ is the skew-symmetric translation matrix, given by:

$$\hat{t}_k = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \tag{2.19}$$

The essential matrix can be computed from 2-D-to-2-D feature correspondences solving the epipolar constraint with the Singular Value Decomposition (SVD). The minimal solution, presented in [38], involves 5-point correspondences. The second step is the decomposition of the essential matrix into the rotation and translation matrices again using the Singular Value Decomposition (SVD). The four different solutions, which can be disambiguated with the triangulation of a single point, are the following:

$$\begin{aligned} R_k &= U(\pm W^T)V^T \\ \hat{t}_k &= U(\pm W)SU^T \end{aligned} \tag{2.20}$$

where:

$$W^T = \begin{bmatrix} 0 & \pm 1 & 0 \\ \mp 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.21}$$

The final step consists in performing a non-linear optimization of the rotation and translation parts using the correct solution obtained from the decomposition of the essential matrix as initial value. The solution is determined by searching for the transformation that minimizes the reprojection error of the triangulated points in each image [15]:

$$T_k = \begin{bmatrix} R_k & t_k \\ 0 & 1 \end{bmatrix} = \underset{X^i, C_k}{\operatorname{argmin}} \sum_{i,k} \|p_k^i - g(X^i, C_k)\|^2 \tag{2.22}$$

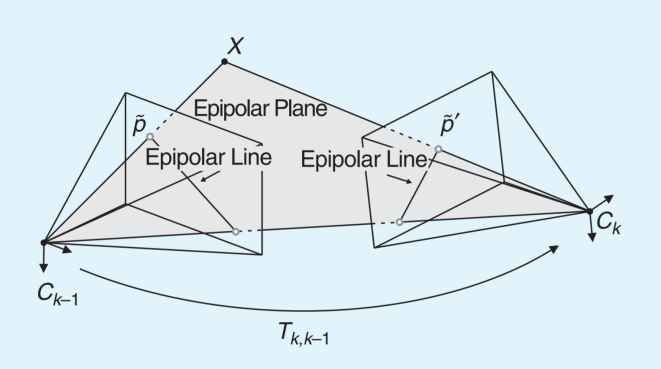

Figure 2.12: Illustration of the epipolar geometry - The epipolar constraint determines the line on which the feature correspondence p' of p lies in the second frame

At this point, the relative scales for the subsequent transformations need to be computed as the absolute scale of the translation cannot be estimated from two frames. The relative scale, after triangulating a pair of 3-D points from a couple of consecutive images, is given by the relation:

$$r = \frac{\|X_{k-1,i} - X_{k-1,j}\|}{\|X_{k,i} - X_{k,j}\|} \tag{2.23}$$

A good rule of thumb is that of considering the mean value of the scale ratios computed for many point pairs. The translation component of the essential matrix is then multiplied by this scale factor at each iteration. In the following, the algorithm for the 2-D-to-2-D motion estimation, presented in [15], is described.

Algorithm 3 2-D-to-2-D Motion Estimation Algorithm

1: Capture a new image I_k
2: Extract and match features between the captured frames I_{k-1} and I_k
3: Compute the essential matrix from the captured frames I_{k-1} and I_k
4: Extract R_k and t_k from the essential matrix
5: Compute the relative scale to rescale t_k
6: Create T_k
7: Estimate the current pose by computing $C_k = C_{k-1}T_k$
8: Repeat the entire procedure from 1

The first approaches of motion estimation presented in this section require the triangulation of 3-D points. The 3-D points are found at the intersection of the back-projected rays from the image correspondences of no less than two frames. In real conditions, the back-projected rays never intersect due to the presence of image noise, camera model and calibration errors, and feature matching uncertainty, thus the point at a minimal distance from the intersecting rays is considered. The triangulated 3-D points show a great uncertainty if the images are captured at nearby positions compared to the scene distance. For this reason, the 3-D-to-3-D approach, which minimizes the 3-D position error instead of the image reprojection error, has the worst accuracy compared to the other approaches. This problem can be fixed by skipping some of the frames captured from the onboard camera until the uncertainty of the 3-D points falls below a determined threshold. This important process, which is called keyframe selection, should be performed at each iteration before updating the motion.

RANSAC for Outlier Removal

The previous step of feature matching normally results in false matches, called outliers, for several reasons such as illumination changes, motion blur, and the presence of dynamic objects in the scene misleading the algorithm. The RANdom SAmpling Consensus (RANSAC) is the most common technique used for robust and accurate motion estimation in presence of outliers. The idea of this method is that of verifying the hypotheses formulated from random samples of points on the remaining points in order to choose as a solution the hypothesis that exhibits the highest consensus with the remaining points [30]. The number of iterations required for a correct solution is given by the following relation:

$$N = \frac{log(1-p)}{log(1-(1-\varepsilon)^s)} \tag{2.24}$$

where s is the minimum number of matched features required for motion estimation, ε is the percentage of outliers in the data and p is the desired probability of success. RANSAC is a probabilistic and non-deterministic algorithm which provides a reliable solution only with some probability that increases as the number of iteration grows. In the following, the RANSAC algorithm, described in [39], is presented.

Algorithm 4 RANSAC Algorithm

1: Let A be a set of N feature correspondences
2: Repeat until reaching the maximum number of iteration:
  2.1: Randomly select a sample of s points from A
  2.2: Compute the transformation using the selected points
  2.3: Apply the transformation to the remaining points of A
  2.4: Compute the distance between the transformed points and their corresponding matches
  2.5: Construct the inliers set (i.e. count the number of points whose distance is below a certain threshold)
  2.6: Store the inliers
3: The set with the highest number of inliers is chosen as the solution of the problem
4: Re-estimate the transformation using the inliers

The influence of outliers on motion estimation, described by Scaramuzza in his presentation "Tutorial on Visual Odometry" [39], is shown in the figure below.

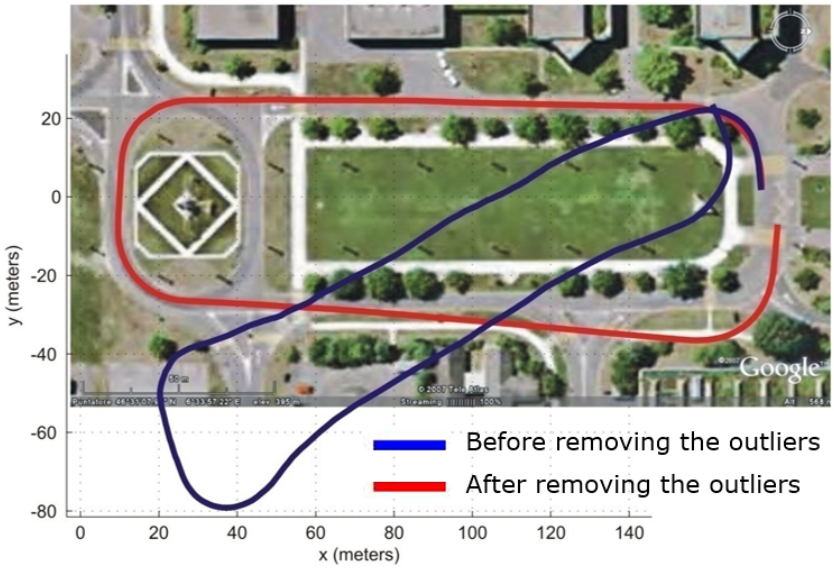

Figure 2.13: Illustration of the influence of outliers on motion estimation - The blue and red lines represent respectively the odometry before and after the outlier removal is performed

2.1.3.4 Pose optimization

The final step of the Visual Odometry pipeline is the local optimization of the camera pose. The most common approaches of pose optimization are the Pose-Graph Optimization and the Windowed (or Local) Bundle Adjustment. In the following, the above-mentioned approaches are described in detail.

Pose-Graph Optimization

Motion estimation is performed by concatenating the transformations obtained from consecutive frames. The transformations can be computed also between non-adjacent frames in order to be used as an additional constraint in pose-graph optimization. The pose-graph optimization is an optimization method used in Visual Odometry to refine the camera poses using a pose graph model. The pose graph consists of nodes, representing the camera poses, and edges, representing the rigid-body transformations between the camera poses, which are considered the constraints between the nodes.

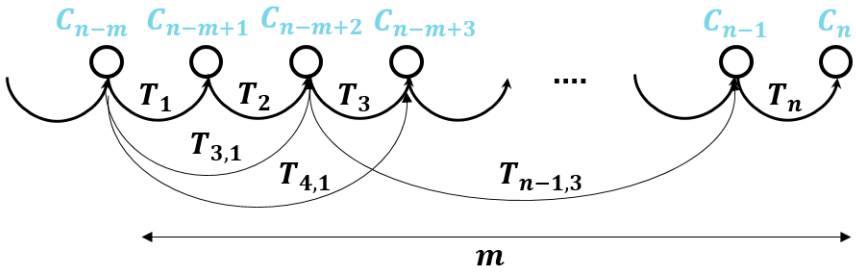

Figure 2.14: Illustration of Pose-Graph Optimization - Pose-graph optimization is an optimization method that refines the camera poses using a pose graph model, which consists of nodes, representing the camera poses, and edges, representing the rigid-body transformations between the camera poses [39]

This optimization method considers also the formation of loop constraints between non-adjacent nodes by reobserving a landmark from a different location after not seeing it for a long period of time. This kind of constraint can be determined by finding visual similarity between the current and the past images captured from the onboard camera with global or local image descriptors. The goal of pose-graph optimization is that of finding the concatenation of poses that best satisfies the above-mentioned constraints. This non-linear optimization problem consists in

finding the camera poses that minimize the following cost function:

$$C^* = \underset{C}{\operatorname{argmin}} \sum_{i,j} \|C_i - T_{i,j} C_j\|^2 \tag{2.25}$$

where $C^* = [C_1^*, ..., C_n^*]^T$ is the vector of the optimized poses and $T_{i,j}$ is the transformation between the poses i and j. The outlined problem needs to be solved using a non-linear optimization algorithm, such as the Levenberg-Marquardt algorithm [30, 16].

Windowed (or Local) Bundle Adjustment

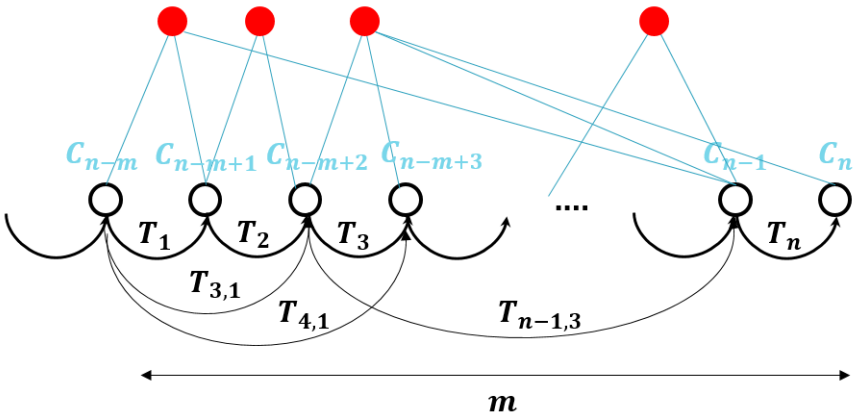

Figure 2.15: Illustration of Windowed Bundle Adjustment - Windowed Bundle Adjustment is an optimization method that jointly optimizes the camera pose and the 3-D point parameters

Windowed bundle adjustment is an optimization method, similar to pose-graph optimization, that jointly performs the optimization of the camera pose and the 3-D-landmark parameters over a window of the last m keyframes. The goal of windowed bundle adjustment is that of finding the concatenation of poses that best satisfies the constraints between the frames incorporated in the window. This non-linear optimization problem consists in finding the camera pose and the 3-D structure parameters that minimize the image reprojection error:

$$C^* = \underset{X^i, C_k}{\operatorname{argmin}} \sum_{i,k} \|p_k^i - g(X^i, C_k)\|^2 \tag{2.26}$$

where p_k^i is the X^i corresponding image point observed in the kth frame and $g(X^i, C_k)$ is its image reprojection according to the camera pose C_k. This optimization problem, as well as the previous one, needs to be solved using a non-linear optimization algorithm, such as the Levenberg-Marquardt algorithm. The choice of a window of the last m keyframes is done to make this optimization method suitable for real-time applications in which low computational resources are available. The computational cost of bundle adjustment is $O(qN + lm)$ where N and m are respectively the number of points and camera poses and q and l are the number of point and camera pose parameters.

2.2 Inertial Measurement Unit

An Inertial Measurement Unit (IMU) is a device that typically consists of a three-axis gyroscope and a three-axis accelerometer providing the angular velocity and the linear acceleration of an agent, respectively. These measurements are processed to perform by dead reckoning the pose estimation of the agent without the need for external references. The most common inertial system for aerial robots is the strapdown system. In a strapdown system, gyroscopes and accelerometers are rigidly mounted on the platform, reducing the mechanical complexity of the system at the cost of higher computational complexity. In fact, in this configuration, the sensor measurements are performed in the body frame, thereby requiring the transformation of the measurements into the global frame to perform the pose estimation. The best sensors used in strapdown systems are Micro Electro-Mechanical Systems (MEMS) sensors because of their small size, lightness, cheapness, low-power consumption, and short start-up time. In sections 2.2.1 and 2.2.2, MEMS gyroscopes and accelerometers are described in detail, presenting the error characteristics of this category of sensors. In section 2.2.3, the strapdown inertial navigation algorithm is presented, describing how the errors that affect the sensor measurements propagate through the algorithm [40].

2.2.1 MEMS Gyroscopes

A MEMS gyroscope is a sensor, produced using silicon micro-machining techniques, that is based on the Coriolis effect. The Coriolis effect asserts that a mass m in motion at velocity v within a frame of reference that rotates at angular velocity ω experiences a fictitious force:

$$F_c = -2m(\omega \times v) \tag{2.27}$$

The Coriolis effect is measured using vibrating elements of different geometries, such as vibrating wheel and tuning fork gyroscopes. The simplest geometry is composed of a mass that is excited into vibration along a drive axis. At the moment in which the gyroscope is subjected to a rotation, a secondary vibration arises along the sense axis, perpendicularly to the drive axis, because of the Coriolis effect. The measurement of this secondary rotation leads to the computation of the angular velocity. The above-described vibrating mass gyroscope is presented in Figure 2.16.

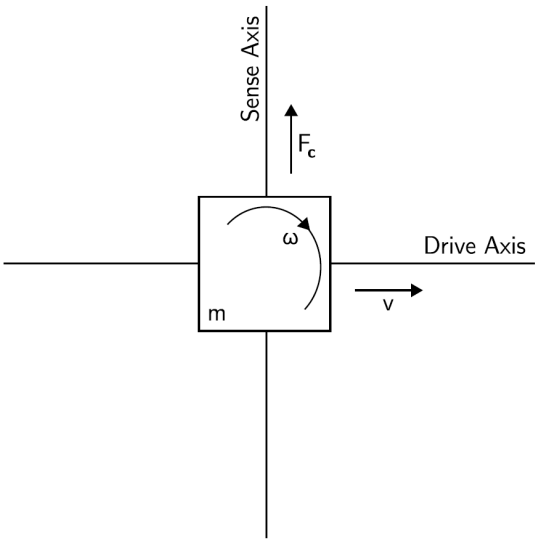

Figure 2.16: Illustration of a vibrating mass gyroscope - The secondary vibration induced along the sense axis because of the Coriolis effect allows the computation of the angular velocity

There are different types of errors that arise in MEMS gyroscopes affecting the measurements of the angular velocity. In the following a brief description of the main errors is provided [40].

- **Constant bias error**

  The bias of a gyroscope is the measurement of the angular velocity when the gyroscope is not undergoing a rotation. This constant error, considered the offset between the gyroscope measurement and the true value, when integrated, causes an error that grows linearly with time. The compensation for this error is done by subtracting the bias from the sensor measurement.

- **Angle random walk error**

  The measurements of a MEMS gyroscope are affected by a thermo-mechanical noise fluctuating at a higher rate than the sampling rate of the gyroscope that results in a white noise sequence of zero-mean uncorrelated random variables. The main goal is that of understanding the effects of the noise on the integration of the measurements. Let's consider N_i as the i-th random variable in the white noise sequence which is identically distributed with zero-mean $E(N_i) = E(N) = 0$ and finite variance $Var(N_i) = Var(N) = \sigma^2$. The integration of the white noise signal $\varepsilon(t)$ using the rectangular rule results in the relation:

$$\int_0^t \varepsilon(\tau)d\tau = \delta t \sum_{i=1}^{n} N_i \tag{2.28}$$

  where n is the number of samples and δt is the time between samples. The properties of expectations for mean and variance $E(aX+bY) = aE(X)+bE(Y)$ and $Var(aX + bY) = a^2Var(X) + b^2Var(Y) + 2abCov(X, Y)$ led to the following results:

$$E\left(\int_0^t \varepsilon(\tau)d\tau\right) = \delta t \cdot n \cdot E(N) = 0 \tag{2.29}$$

$$Var\left(\int_0^t \varepsilon(\tau)d\tau\right) = \delta t^2 \cdot n \cdot Var(N) = \delta t \cdot t \cdot \sigma^2 \tag{2.30}$$

  The obtained results show that gyroscope white noise introduces in orientation an angle random walk error with a standard deviation that grows with the square root of time.

$$\sigma_\theta(t) = \sqrt{Var(t)} = \sigma \cdot \sqrt{\delta t \cdot t} \tag{2.31}$$

  The most common measurement used to evaluate the effects of the white noise on the attitude estimation is the angle random walk (ARW):

$$ARW\ [rad/\sqrt{s}] = \sigma_\theta(1) \tag{2.32}$$

- **Bias stability**

  The bias of a MEMS gyroscope starts fluctuating because of the flicker noise that arises in electronic components. The effects of the flicker noise are only appreciable at low frequencies because at high frequencies the white noise is predominant. Bias fluctuations can be modeled with good approximation as a bias random walk with a standard deviation that grows with the square root of time. The effects of the noise on the orientation computed from the integration of the gyroscope signal is a second-order angle random walk. The most common measurement for manufacturers to evaluate the effects of the flicker noise is bias stability. The bias stability measurement, which determines the change of the bias over a period of time of 100 seconds in fixed conditions, can sometimes be expressed in terms of a bias random walk measurement:

$$BRW\left[rad/s^{\frac{3}{2}}\right] = \frac{BS\,[rad/s]}{\sqrt{t\,[s]}} \tag{2.33}$$

- **Temperature effects**

  Bias fluctuations can also be introduced because of temperature variations attributed to changes in the environment and sensor overheating. The effect of these bias fluctuations is an error in orientation that grows linearly with time.

- **Calibration errors**

  Calibration errors, including errors in scale factors, alignment, and linearities of the gyroscope, induce a bias in the sensor measurements which is only observable when the gyroscope is rotating. The effects of this bias on the orientation is an error of magnitude proportional to the rate and duration of the motion.

2.2.2 MEMS Accelerometers

A MEMS accelerometer is a sensor that consists of a mass in a spring-like structure vibrating in a capacitive divider. The displacement of the mass is converted by the capacitive divider into an electric signal proportional to the acceleration applied to the mass along the input axis. In this case, damping is performed by the gas sealed in the device. The above-described vibrating mass accelerometer is presented in Figure 2.17.

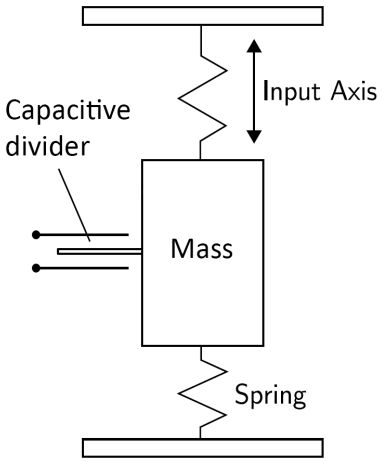

Figure 2.17: Illustration of a vibrating mass accelerometer - The displacement of the mass is converted by the capacitive divider into an electric signal proportional to the acceleration applied to the mass along the input axis

As seen in the previous section, there are different types of errors that arises in MEMS accelerometers affecting the measurements of linear acceleration. In the following a brief description of the main errors is provided [40].

- **Constant bias error**

  The bias of an accelerometer is the measurement of the linear acceleration when the sensor is not undergoing an acceleration. This constant error, considered the offset between the accelerometer measurement and the true value, when double integrated, causes an error in position that grows quadratically with time. The compensation for this error is done by subtracting the bias from the sensor measurement. Unfortunately, the bias estimation is complicated because of a component of gravity acting on the accelerometer that appears as a bias. This problem is solved by performing the bias measurement during calibration procedures in which the orientation of the device is known a priori.

- **Velocity random walk error**

  The measurements of a MEMS accelerometer are affected by a thermo-mechanical noise fluctuating at a higher rate than the sampling rate of the accelerometer that results in a white noise sequence of zero-mean uncorrelated random variables. The white noise sequence introduces a velocity random walk error in the sensor measurements. The goal is that of determining the effects of the white noise on the position calculated by double integrating the accelerometer measurements. Let's consider N_i as the i-th random variable in the white noise sequence which is identically distributed with zero-mean $E(N_i) = E(N) = 0$ and finite variance $Var(N_i) = Var(N) = \sigma^2$. The double integration of the white noise signal $\varepsilon(t)$ using the rectangular rule results in the relation:

$$\int_0^t \int_0^t \varepsilon(\tau) d\tau d\tau = \delta t \sum_{i=1}^{n} \delta t \sum_{j=1}^{i} N_j = \delta t^2 \sum_{i=1}^{n} (n - i + 1) N_i \tag{2.34}$$

  where n is the number of samples and δt is the time between samples. The properties of expectations for mean and variance $E(aX+bY) = aE(X)+bE(Y)$ and $Var(aX + bY) = a^2Var(X) + b^2Var(Y) + 2abCov(X, Y)$ led to the following results:

$$E\left(\int_0^t \int_0^t \varepsilon(\tau) d\tau d\tau \right) = \delta t^2 \sum_{i=1}^{n} (n - i + 1) E(N_i) = 0 \tag{2.35}$$

$$Var\left(\int_0^t \int_0^t \varepsilon(\tau) d\tau d\tau \right) = \delta t^4 \sum_{i=1}^{n} (n - i + 1)^2 Var(N_i) \approx \frac{1}{3} \cdot \delta t \cdot t^3 \cdot \sigma^2 \tag{2.36}$$

  The obtained results show that accelerometer white noise introduces a second-order random walk error in position with a standard deviation that grows proportionally to $t^{\frac{3}{2}}$.

$$\sigma_s(t) \approx \sigma \cdot t^{\frac{3}{2}} \cdot \sqrt{\frac{\delta t}{3}} \tag{2.37}$$

- **Bias stability**

  The bias of a MEMS accelerometer starts fluctuating because of the flicker noise that arises in electronic components. The effects of the flicker noise are only appreciable at low frequencies because at high frequencies the white noise is predominant. Bias fluctuations can be modeled with good approximation as a bias random walk with a standard deviation that grows with the square root of time. The effects of the noise on the integration of the accelerometer signal is a third-order random walk in position with a standard deviation that grows proportionally to $t^{\frac{5}{2}}$.

- **Temperature effects**

  For an accelerometer, as well as for a gyroscope, bias fluctuations can be introduced because of temperature variations attributed to changes in the environment and sensor overheating. The effect of these bias fluctuations is an error in position that grows quadratically with time.

- **Calibration errors**

  Calibration errors, including errors in scale factors, alignment, and linearities of the accelerometer, induce a bias in the sensor measurements which is observable even when the accelerometer is stationary because of the presence of gravitational acceleration.

2.2.3 Strapdown Inertial Navigation

The most common inertial system for an aerial robot is the strapdown system. In a strapdown system, gyroscopes and accelerometers are rigidly mounted on the platform, reducing the mechanical complexity of the system at the cost of higher computational complexity. In fact, in this configuration, the sensor measurements are performed in the body frame, thereby requiring the transformation of the measurements into the global frame to perform the pose estimation. The strapdown inertial navigation algorithm is presented in Figure 2.18. The gyroscope and the accelerometer measure respectively the angular velocity and the linear acceleration of the platform along the three main axes. The orientation of the system with respect to the global frame is computed by integrating the angular velocity using the original orientation as the initial condition. After performing the attitude estimation, the linear acceleration is projected into the global frame using the calculated orientation. The position of the system with respect to the global frame is computed by double integrating the linear acceleration after subtracting

gravity acceleration along the vertical axis. In this section, the strapdown inertial navigation algorithm is described in detail showing how errors that affect the sensor measurements propagate through the algorithm.

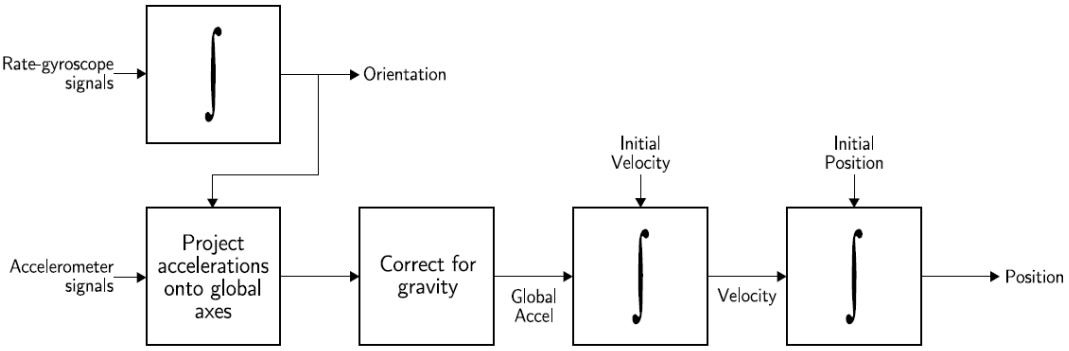

Figure 2.18: Strapdown inertial navigation algorithm - The linear accelerations measured by the accelerometer require to be projected onto the global frame using the orientation determined by the integration of the gyroscope measurements

Orientation

The orientation of the system with respect to the global frame is computed by integrating the angular velocity measurement performed by the gyroscope along the three main axes. This measurement is denoted by the relation:

$$\tilde{\omega}_B(t) = \omega_B(t) + b^g(t) + n^g(t) \tag{2.38}$$

where b^g and n^g are respectively the bias and the noise that arises in the gyroscope. There are different attitude representations in literature such as Euler angles, quaternions or direction cosines. In this section, the direction cosines representation is chosen for the description of the attitude estimation. The orientation of the system with respect to the global frame is represented by the rotation matrix:

$$R = R_x R_y R_z \tag{2.39}$$

where:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & sin\phi \\ 0 & -sin\phi & cos\phi \end{bmatrix} \tag{2.40}$$

$$R_y = \begin{bmatrix} cos\theta & 0 & -sin\theta \\ 0 & 1 & 0 \\ sin\theta & 0 & cos\theta \end{bmatrix} \tag{2.41}$$

$$R_z = \begin{bmatrix} cos\psi & sin\psi & 0 \\ -sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.42}$$

in which ϕ, θ and ψ are the rotations around the x,y and z axes. The rate of change of the rotation matrix is defined as:

$$\dot{R}(t) = \lim_{\delta t \to 0} \frac{R(t + \delta t) - R(t)}{\delta t} \tag{2.43}$$

in which $R(t + \delta t)$ can be written as:

$$R(t + \delta t) = R(t)A(t) \tag{2.44}$$

where $A(t)$ is the rotation matrix that relates the body frame in time instants t and $t + \delta t$. The small angle approximation can be used assuming that the rotations $\delta\phi$, $\delta\theta$ and $\delta\psi$ of the body frame between time instants are small. In this case, the rotation matrix $A(t)$ can be written as:

$$A(t) = I + \delta\Psi \tag{2.45}$$

where:

$$\delta\Psi = \begin{bmatrix} 0 & -\delta\psi & \delta\theta \\ \delta\psi & 0 & -\delta\phi \\ -\delta\theta & \delta\phi & 0 \end{bmatrix} \tag{2.46}$$

The differential equation obtained by substitution can be written as:

$$\dot{R}(t) = R(t) \lim_{\delta t \to 0} \frac{\delta\Psi}{\delta t} = R(t)\Omega(t) \tag{2.47}$$

where $\Omega(t)$ is the skew-symmetric matrix of the angular velocity vector expressed by:

$$\Omega(t) = \begin{bmatrix} 0 & -\tilde{\omega}_{bz}(t) & \tilde{\omega}_{by}(t) \\ \tilde{\omega}_{bz}(t) & 0 & -\tilde{\omega}_{bx}(t) \\ -\tilde{\omega}_{by}(t) & \tilde{\omega}_{bx}(t) & 0 \end{bmatrix} \tag{2.48}$$

The solution to the differential equation has the following form:

$$R(t) = R(0) + exp\left(\int_0^t \Omega(t)dt\right) \tag{2.49}$$

where $R(0)$ is the initial orientation of the system. Since the IMU provides samples of the angular velocity at a certain frequency, it is necessary to use an integration model for the integration of the sampled measurements. Using the rectangular rule, the solution for a time interval $[t, t + \delta t]$ can be written as:

$$R(t + \delta t) = R(t) + exp\left(\int_t^{t+\delta t} \Omega(t)dt\right) = R(t) + exp(B) \tag{2.50}$$

where:

$$B = \begin{bmatrix} 0 & -\tilde{\omega}_{bz}\delta t & \tilde{\omega}_{by}\delta t \\ \tilde{\omega}_{bz}\delta t & 0 & -\tilde{\omega}_{bx}\delta t \\ -\tilde{\omega}_{by}\delta t & \tilde{\omega}_{bx}\delta t & 0 \end{bmatrix} \tag{2.51}$$

The attitude update equation is obtained by performing the Taylor expansion of the exponential term:

$$R(t + \delta t) = R(t)\left(I + B + \frac{B^2}{2!} + \frac{B^3}{3!} + \frac{B^4}{4!} + ...\right) = R(t)\left(I + \frac{sin\sigma}{\sigma}B + \frac{1 - cos\sigma}{\sigma^2}B^2\right) \tag{2.52}$$

where $\sigma = |\tilde{\omega}_B \delta t|$. In the previous sections, the different types of errors that arise in gyroscopes affecting the measurements of angular velocity are described in detail. The integration of the measurements performed by the attitude algorithm allows the propagation of the errors through the algorithm resulting in an unreliable attitude estimation. For a MEMS gyroscope, the most important sources of error in orientation are uncorrected bias and white noise errors. The uncorrected bias introduces in orientation an error that grows linearly with time. White noise causes an angle random walk error with a standard deviation that grows with the square root of time. In addition, there are quantization errors that arise in the attitude algorithm because of the quantization of the angular velocity samples and the integration model used for updating the orientation.

Position

The accelerometer provides the measurement of the linear acceleration along the three main axes. The linear acceleration measurement is denoted by the relation:

$$\tilde{a}_B(t) = a_B(t) + b^a(t) + n^a(t) \tag{2.53}$$

where b^a and n^a are respectively the bias and the noise that arises in the accelerometer. After performing the attitude estimation, the linear acceleration, measured in the body frame, is projected into the global frame using the calculated orientation:

$$\tilde{a}_G(t) = R(t)\tilde{a}_B(t) \tag{2.54}$$

The velocity of the system with respect to the global frame is computed by integrating the linear acceleration after subtracting the acceleration of gravity along the vertical axis:

$$\tilde{v}_G(t) = v_G(0) + \int_0^t [\tilde{a}_G(t) - g_G]\, dt \tag{2.55}$$

where $v_G(0)$ is the initial velocity of the system and g_G is the gravity acceleration. The position of the system with respect to the global frame is computed by integrating the velocity using the original position of the system as the initial condition:

$$\tilde{p}_G(t) = p_G(0) + \int_0^t \tilde{v}_G(t)\, dt \tag{2.56}$$

Since IMU provides samples of linear acceleration at a certain frequency, as for the attitude algorithm, it is necessary to use an integration model for the integration of the sampled measurement. Using the rectangular rule, the following update equations for velocity and position are obtained:

$$\tilde{v}_G(t+\delta t) = \tilde{v}_G(t) + \delta t \cdot [\tilde{a}_G(t+\delta t) - g_G] \tag{2.57}$$

$$\tilde{p}_G(t+\delta t) = \tilde{p}_G(t) + \delta t \cdot \tilde{v}_G(t+\delta t) \tag{2.58}$$

In the previous sections, the different types of errors that arise in accelerometers affecting the measurements of linear acceleration are described in detail. The double integration of the measurements performed by the position algorithm allows the propagation of the errors through the algorithm resulting in a large drift in position. The drift in position is also caused by the errors that arise in gyroscopes because the orientation obtained from the attitude algorithm is used to project the linear acceleration into the global frame. There are different types of problems caused by an inaccurate projection of the linear acceleration into the global frame such as an incorrect subtraction of the gravitational acceleration and the integration of the measurements in the wrong direction. This results in a significant accumulated error in position over time.

2.3 Visual-Inertial Odometry Techniques

The Visual-Inertial Odometry algorithms can be classified in loosely-coupled and tightly-coupled approaches according to the type of information used for visual and inertial sensor fusion. The loosely-coupled approach consists in fusing the motion estimates obtained by processing visual and inertial measurements separately. Conversely, the tightly-coupled approach consists in performing the motion estimation by using directly the raw camera and IMU measurements, that is feature positions and IMU readings of angular velocity and linear acceleration. The tightly-coupled approach improves the accuracy at the cost of a higher implementation complexity compared to the loosely-coupled approach. The higher accuracy of the tightly-coupled approach is caused by several reasons. First of all, the IMU can be used for guided feature matching resulting in a greater number of inliers essential for robust motion estimation. Secondly, minimizing the reprojection error and the IMU error together places an additional constraint on the pose estimation. Finally, the tightly-coupled approach, in contrast to the loosely-coupled approach, considers the visual and inertial information coupling that allows for the drift compensation. The differences between the two paradigms are illustrated in Figure 2.19.

These approaches can be further classified in full smoothers, fixed-lag smoothers, and filtering methods according to the number of camera poses involved in the estimation. Full smoothers, also called batch nonlinear least-squares algorithms, estimate the entire history of the camera poses, fixed-lag smoothers, also called sliding window estimators, estimate a window of the last m poses, and finally filtering methods estimate just the latest pose. The main drawback of fixed-lag smoothers and filtering methods is that, in contrast to full smoothers that allow re-linearization, they marginalize older states locking the linearization error, resulting in a less accurate estimation. In recent years, the research has focused the attention on optimization-based methods such as full and fixed-lag smoothers rather than filtering methods because of the improvements in the computing power of computers [13].

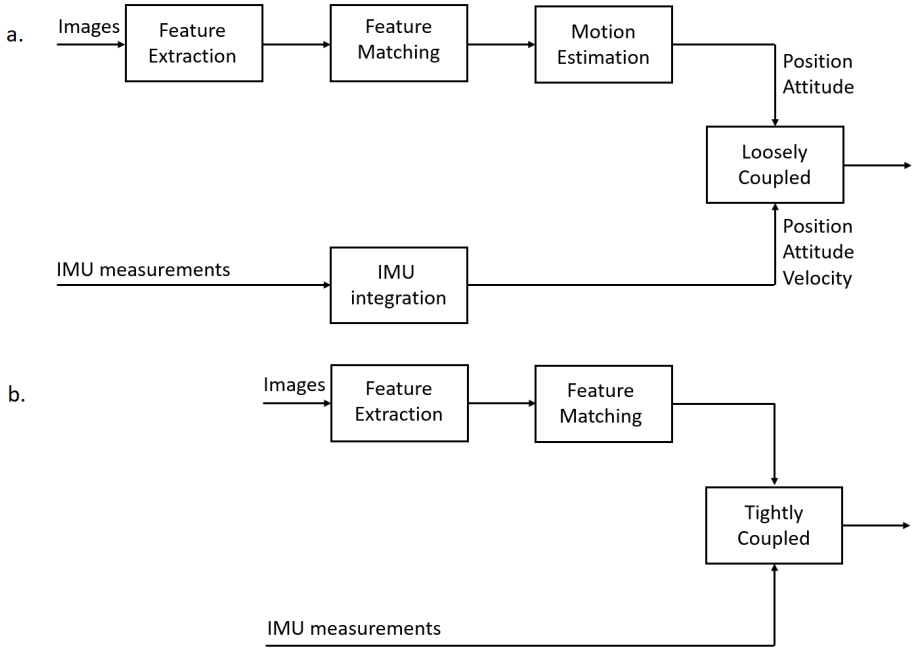

Figure 2.19: Illustration of loosely-coupled (a) and tightly-coupled approaches (b) - The loosely-coupled approach uses the output of the individual systems while the tightly-coupled approach uses the internal states

Chapter 3

Visual-Inertial Odometry State of the Art Algorithms

The main purpose of this Master Thesis is to provide a comparison between different visual-inertial odometry algorithms to assess which one is the best solution in terms of accuracy for UAVs navigation in GPS-denied environments. The state of the art algorithms chosen for the research are:

- Intel T265 Proprietary Algorithm
- SVO
- VINS-Fusion
- ORB-SLAM 2
- ORB-SLAM 3

These algorithms were chosen because they turned out to be the most user friendly among the renowned pipelines in the panorama of visual-inertial odometry. They are all feature-based algorithms except for SVO that uses a semi-direct approach for pose estimation. This chapter provides a detailed description of the open source algorithms among the ones mentioned above.

3.1 SVO

The Semi-direct Visual Odometry (SVO) algorithm is a visual-inertial odometry algorithm that was first presented by Forster et al. in their paper "SVO: Fast

Semi-Direct Monocular Visual Odometry" [41]. This algorithm uses a semi-direct approach for state estimation. The semi-direct approach is a combination of the feature-based approach and the direct approach. This approach is convenient for several reasons. First of all, it requires less computational resources compared to the other state of the art algorithms because it doesn't require the process of feature extraction and matching at every frame for motion estimation. In addition, it achieves sub-pixel level of accuracy operating directly on the intensity of pixels in the image. Finally, this algorithm is suitable for every situation as it supports different types of cameras.

3.1.1 Algorithm architecture

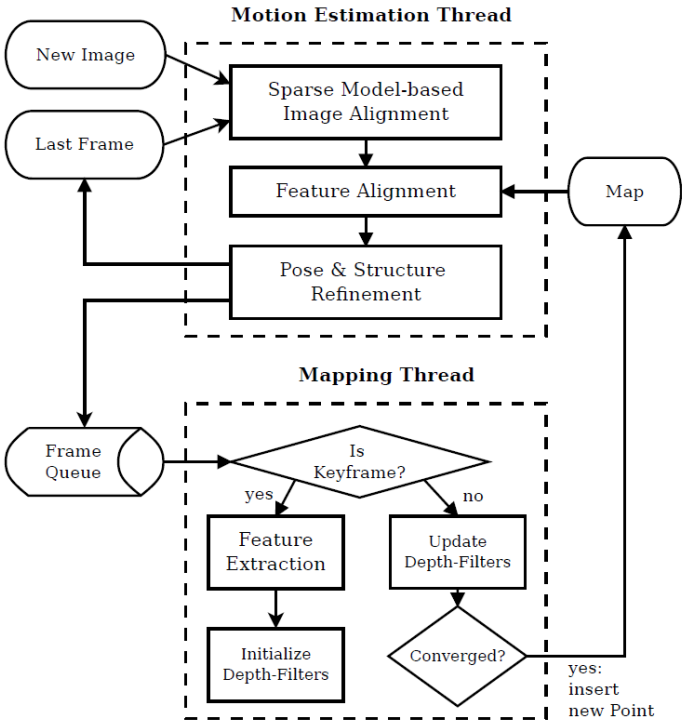

Figure 3.1: The architecture of the algorithm is composed of two parallel threads: the motion estimation thread and the mapping thread [41].

The architecture of the algorithm is organized in two separated threads: the first one is the motion estimation thread and the second one is the mapping thread. The motion estimation thread is articulated in three different steps. The first step is the sparse model-based image alignment that consists in the computation of

the transformation between the current and the previous frame by minimizing the photometric error between pixels observing the same 3D point. The second step is feature alignment that consists in the optimization of the feature positions in the image. The final step consists in the refinement of pose and structure by minimizing the reprojection error generated in the second step. The mapping thread is designated for mapping the environment. This thread uses a Bayesian filter for depth estimation. The filter is initialized with a high uncertainty in depth in the areas of the keyframes in which there is a small number of 3D-to-2D correspondences. In the successive frames, the update of the depth estimate is performed until the filter has converged. The filter convergence occurs when the uncertainty in depth falls below a certain threshold. The 3D point is then introduced in the map that is ready to be employed in the motion estimation thread. In the following sections, the implementation of the algorithm is described in detail.

3.1.2 Motion Estimation

The algorithm performs motion estimation by using a semi-direct approach. The process of motion estimation is articulated in three different steps: the first one is the sparse model-based image alignment, the second one is the feature alignment, and the last one is the pose and structure refinement. In this section, the three steps of the motion estimation thread are described in detail.

Sparse model-based image alignment

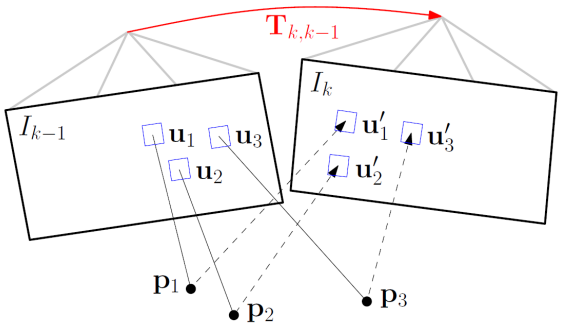

Figure 3.2: The sparse model-based image alignment consists in the computation of the transformation $T_{k,k-1}$ that minimizes the photometric error between pixels observing the same 3D point [41].

The sparse model-based image alignment consists in the computation of the transformation between the current and the previous frame by minimizing the photometric error of the patches observing the same 3D point:

$$T_{k,k-1} = \underset{T_{k,k-1}}{\operatorname{argmin}} \frac{1}{2} \sum_{i \in \bar{\mathcal{R}}} \|\delta I(T_{k,k-1}, u_i)\|^2 \tag{3.1}$$

This nonlinear least squares problem can be solved by using the Gauss-Newton algorithm. Consider an estimate of the transformation $\hat{T}_{k,k-1}$. The incremental update $T(\xi)$ to the estimate can be parameterized by using the twist coordinates $\xi = (\omega, v)^T$. The computation of the update step $T(\xi)$ is performed by using the inverse compositional formulation. The best update step $T(\xi)$ can be determined by setting to zero the derivative of the previous equation:

$$\sum_{i \in \bar{\mathcal{R}}} \nabla \delta I(\xi, u_i)^T \delta I(\xi, u_i) = 0 \tag{3.2}$$

This system of equations can be solved by linearizing the intensity residual as follows:

$$\delta I(\xi, u_i) \approx \delta I(0, u_i) + \nabla \delta I(0, u_i) \cdot \xi \tag{3.3}$$

Substituting this expression in the system of equations we derive the normal equations:

$$J^T J \xi = -J^T \delta I(0) \tag{3.4}$$

where J is the Jacobian matrix obtained by stacking the Jacobians which are calculated by using the chain rule. For example, the Jacobian $J_i = \nabla \delta I(0, u_i)$ is computed as:

$$\frac{\partial \delta I(\xi, u_i)}{\partial \xi} = \left.\frac{\partial I_{k-1}(a)}{\partial a}\right|_{a=u_i} \cdot \left.\frac{\partial \pi(b)}{\partial b}\right|_{b=p_i} \cdot \left.\frac{\partial T(\xi)}{\partial \xi}\right|_{\xi=0} \cdot p_i \tag{3.5}$$

These equations can be solved for ξ. The transformation $T_{k,k-1}$ is computed by applying the inverse of the update step $T(\xi)$ to the estimate of the transformation $\hat{T}_{k,k-1}$.

Feature alignment

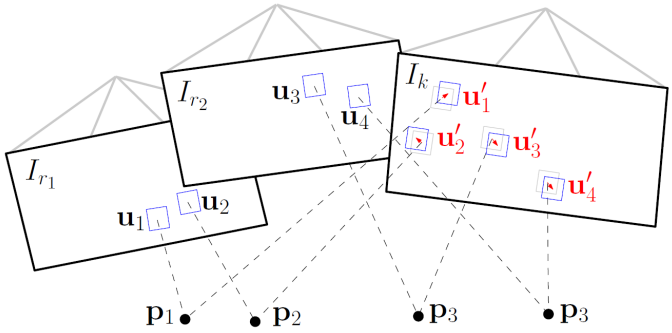

Figure 3.3: The feature alignment consists in the optimization of the feature positions obtained by the back-projection of the transformation $T_{k,k-1}$ [41].

The transformation $T_{k,k-1}$ determines by back-projection an estimate of the feature positions $u_i^{'}$ of the 3D points of the map in the image. The feature alignment step consists in the optimization of feature positions. The keyframe of reference r observing every point from the nearest viewpoint is determined. The optimization of the feature position $u_i^{'}$ is performed through the minimization of the photometric error between corresponding pixels in the current frame and in the reference frame. The optimized feature position is defined as:

$$u_i^{'} = \underset{u_i^{'}}{\operatorname{argmin}} \frac{1}{2} \| I_k(u_i^{'}) - A_i \cdot I_r(u_i) \|^2 \tag{3.6}$$

where A_i is the distortion applied to the patches of the reference frame. The distortion needs to be applied because the reference frame is placed at a greater distance with respect to the current frame compared to the previous frame. This nonlinear least squares problem can be solved by using the Lucas-Kanade optical flow method. The feature alignment step defines the feature positions in the image with a sub-pixel level of accuracy by violation of the epipolar constraints.

Pose and structure refinement

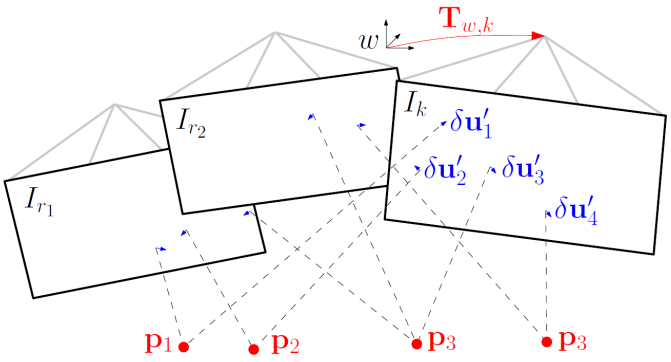

Figure 3.4: The camera pose and the structure of the 3D points are refined by minimizing the reprojection error determined during the feature alignment step [41].

The pose and structure refinement consists in the optimization of the camera pose and the position of 3D points by minimizing the reprojection error created during the feature alignment. The reprojection error is defined as:

$$\|\delta u_i\| = \|u_i - \pi(T_{k,w} \ {}_w p_i)\|^2 \tag{3.7}$$

The process of pose and structure refinement is articulated in three different steps. The first one consists in the optimization of the camera pose by solving the following nonlinear least squares problem:

$$T_{k,w} = \underset{T_{k,w}}{\operatorname{argmin}} \frac{1}{2} \sum_i \|u_i - \pi(T_{k,w} \ {}_w p_i)\|^2 \tag{3.8}$$

The second one consists in the optimization of the structure that is performed in a similar way to that of camera pose outlined above. The third step is called local bundle adjustment. This process consists in the refinement of both the pose of the latest frames and the 3D position of the points together.

3.1.3 Mapping

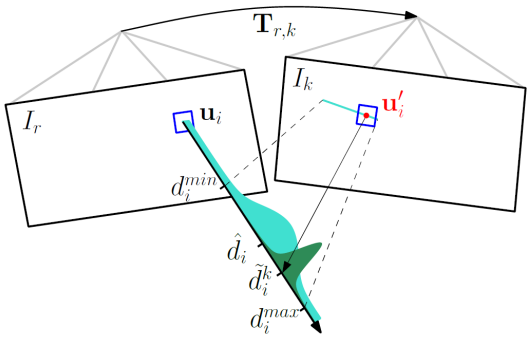

Figure 3.5: The mapping process consists in the estimation of the depth required for the triangulation of the 3D point to insert in the map by using a Bayesian filter [41].

The mapping thread is designated for mapping the environment. The environment is mapped by using a Bayesian filter for depth estimation. The filter is initialized with a high uncertainty in depth in the areas of the keyframe in which there are only a few 3D-to-2D correspondences. The observation of the successive frames is used for updating the depth estimate by using the depth $\tilde{d}_i^k$ computed by triangulation from the point $u_i^{'}$ of highest correlation with the patch of the keyframe. The point of highest correlation is found on the epipolar line that is computed from the transformation $T_{r,k}$ and the line passing through the point u_i. The depth $\tilde{d}_i^k$ is defined by using a Gaussian-uniform mixture model:

$$p(\tilde{d}_i^k|d_i,\rho_i) = \rho_i \mathcal{N}\left(\tilde{d}_i^k|d_i,\tau_i^2\right) + (1-\rho_i)\mathcal{U}(\tilde{d}_i^k|d_i^{min},d_i^{max}) \tag{3.9}$$

where ρ_i is the inlier probability and τ_i^2 is the variance of an inlier measurement. The update of the depth estimate is performed until the filter has converged. The filter convergence occurs when the variance of the distribution falls below the convergence threshold. The 3D point is then triangulated by using the relationship:

$$_kp = \pi^{-1}(u,d_u) \tag{3.10}$$

where π is the projection model, $u = (u,v)^T$ are the image coordinates, and d_u

is the depth estimate. Finally, the 3D point is introduced in the map that is ready to be employed in the motion estimation thread. This method provides a better performance in terms of robustness and accuracy than the standard method because the filter updates the depth estimate several times before the convergence is reached. This results in a map with a smaller number of outliers.

3.1.4 Experimental results

The experimental results obtained by running the algorithm on a laptop set up with ROS Melodic are displayed in Rviz. The left side of the figure shows the input images of the camera in a monocular setup. The right side of the figure shows the output of the algorithm that consists of the odometry data represented by a blue line and the point cloud map of the environment represented by violet sparse points.

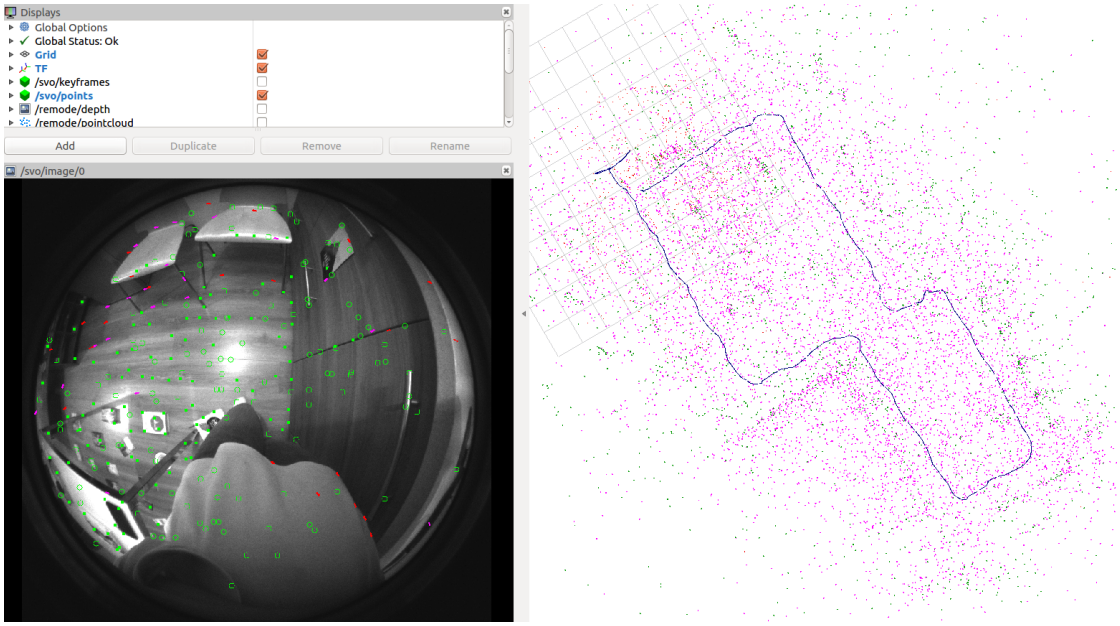

Figure 3.6: Illustration of the algorithm performance in Rviz - The output of the algorithm consists of the odometry data represented by a blue line and the point cloud map of the environment represented by violet sparse points

3.2 VINS-Fusion

VINS-Fusion is an optimization-based multi-sensor visual-inertial odometry algorithm for state estimation of autonomous vehicles. The algorithm performs state estimation by using a tightly-coupled nonlinear optimization-based approach. This approach is used to obtain a high performance in terms of robustness and accuracy by fusing pre-integrated IMU measurements and feature correspondences. The other properties of the algorithm are: efficient IMU pre-integration with bias correction, automatic estimator initialization, online spatial and temporal calibration, visual loop closure, and global pose graph optimization and reuse. The algorithm is considered an extension of VINS-Mono supporting both monocular and stereo cameras in different configurations. The implementation of the algorithm is presented by Qin et al. in their paper "VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator" [42].

3.2.1 Algorithm architecture

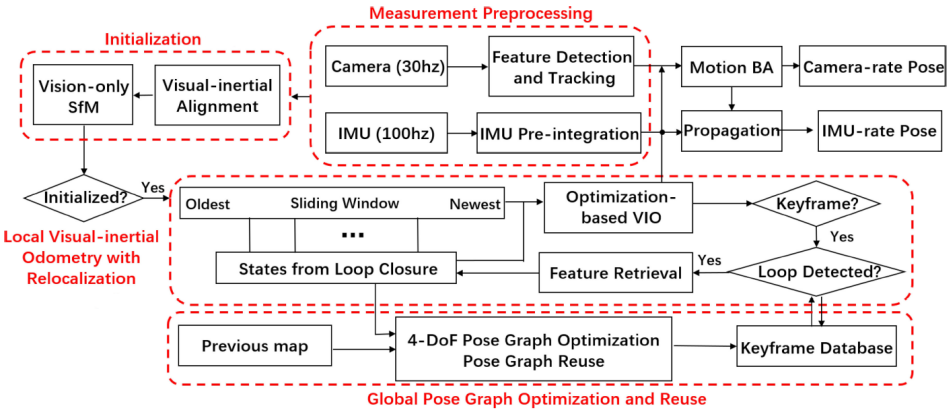

Figure 3.7: The architecture of the algorithm is composed of four different threads: measurement preprocessing, estimator initialization, Visual-Inertial Odometry with relocalization, and global pose graph optimization and reuse [42].

The architecture of the algorithm is organized in four different threads: measurement preprocessing, estimator initialization, Visual-Inertial Odometry with relocalization, and global pose graph optimization and reuse. The measurement preprocessing consists in performing feature detection and tracking and IMU pre-integration.

The estimator initialization consists in computing the necessary variables for the Visual-Inertial Odometry thread. The necessary variables are pose, velocity, gravity acceleration, gyroscope bias and feature positions. The Visual-Inertial Odometry thread with relocalization performs state estimation by fusing IMU pre-integrated measurements and feature correspondences. Finally, the global pose graph optimization is the process of refining the poses by removing the accumulated drift to achieve global consistency of the path. This thread also provides the reuse of the pose graph. The Visual-Inertial Odometry and pose graph optimization threads are executed simultaneously.

3.2.2 Measurement Preprocessing

The measurement preprocessing is performed for both visual and inertial measurements. The preprocessing of visual measurements consists in detecting and tracking features across the frames while the preprocessing of inertial measurements consists in the pre-integration of the measurements between two subsequent frames. In the following, the operations of preprocessing are described in detail.

Visual Preprocessing

The preprocessing of visual measurements is articulated in three different processes. The first one consists in the detection of features in the current frame. The number of features must be maintained constant in each frame. A good rule of thumb is that of detecting around 100-300 features in every frame. In addition, the distribution of the features must be as much as possible uniform in the image. The second one consists in tracking the detected features by using the KLT feature tracker. This process is completed by using RANSAC for outlier removal. Finally, the third one consists in keyframe selection. There are two different methods to perform the keyframe selection: the parallax of features and the tracking quality. The first one consists in selecting a new keyframe when the average parallax of features between the current frame and the last keyframe is greater than a fixed limit. The second one consists in selecting a new keyframe when the number of features involved in the tracking process exceeds a minimum threshold. This method is useful to avert the loss of feature tracks.

IMU Pre-integration

1) IMU Measurements: The accelerometer and gyroscope measurements are respectively the linear acceleration and the angular velocity. These measurements are

expressed by the following relationships:

$$\begin{aligned} \hat{a}_t &= a_t + b_{a_t} + R_w^t g^w + n_a \\ \hat{\omega}_t &= \omega_t + b_{w_t} + n_w \end{aligned} \tag{3.11}$$

where n and b are respectively the additive noise and the bias affecting the measurements. The additive noise can be modeled as a Gaussian white noise and the bias can be modeled as a random walk. The derivative of a random walk process is again a Gaussian white noise.

$$\begin{aligned} n_a \sim \mathcal{N}(0, \sigma_a^2) \qquad & n_w \sim \mathcal{N}(0, \sigma_w^2) \\ \dot{b}_{a_t} = n_{b_a} \sim \mathcal{N}(0, \sigma_{b_a}^2) \qquad & \dot{b}_{w_t} = n_{b_w} \sim \mathcal{N}(0, \sigma_{b_w}^2) \end{aligned} \tag{3.12}$$

2) Pre-integration: There are a lot of inertial measurements between the two subsequent frames b_k and b_{k+1} due to the higher sampling rate of the inertial sensor. The pre-integration terms are computed by integrating the inertial measurements in the time interval $[t_k, t_{k+1}]$ as follows:

$$\begin{aligned} \alpha_{b_{k+1}}^{b_k} &= \iint_{t \in [t_k, t_{k+1}]} R_t^{b_k} \left(\hat{a}_t - b_{a_t} \right) dt^2 \\ \beta_{b_{k+1}}^{b_k} &= \int_{t \in [t_k, t_{k+1}]} R_t^{b_k} \left(\hat{a}_t - b_{a_t} \right) dt \\ \gamma_{b_{k+1}}^{b_k} &= \int_{t \in [t_k, t_{k+1}]} \frac{1}{2} \Omega(\hat{\omega}_t - b_{w_t}) \gamma_t^{b_k} \, dt \end{aligned} \tag{3.13}$$

where:

$$\Omega(\omega) = \begin{bmatrix} -\lfloor \omega \rfloor_\times & \omega \\ -\omega^T & 0 \end{bmatrix}, \qquad \lfloor \omega \rfloor_\times = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \tag{3.14}$$

The pre-integration is then obtained simply using the inertial measurements given a bias estimate. The covariance $P_{b_{k+1}}^{b_k}$ of the pre-integration terms propagates in the same way.

3) Bias Correction: The pre-integration terms can be corrected by using their first-order approximations with respect to the bias if the estimation of the bias has undergone a small variation:

$$\begin{aligned}
\alpha_{b_{k+1}}^{b_k} &\approx \hat{\alpha}_{b_{k+1}}^{b_k} + J_{b_a}^{\alpha}\delta b_{a_k} + J_{b_w}^{\alpha}\delta b_{w_k} \\
\beta_{b_{k+1}}^{b_k} &\approx \hat{\beta}_{b_{k+1}}^{b_k} + J_{b_a}^{\beta}\delta b_{a_k} + J_{b_w}^{\beta}\delta b_{w_k} \\
\gamma_{b_{k+1}}^{b_k} &\approx \hat{\gamma}_{b_{k+1}}^{b_k} \otimes \begin{bmatrix} 1 \\ \frac{1}{2} J_{b_w}^{\gamma}\delta b_{w_k} \end{bmatrix}
\end{aligned} \tag{3.15}$$

Conversely, the repropagation under the new bias estimation is performed if the estimation of the bias has undergone a significant change. The IMU pre-integration eliminates the need to propagate the inertial measurements continuously resulting in a great saving of computational resources.

3.2.3 Estimator Initialization

The tightly-coupled visual-inertial odometry algorithm needs an initial guess to start state estimation because of its highly nonlinear nature. The initial values are computed by aligning the visual structure and the IMU pre-integration.

Structure from Motion (SfM)

The Structure from Motion (SfM) is a process that estimates a graph of camera poses and feature locations. This process is performed in a sliding window in order to reduce the level of computational resources usage. First of all, the feature correspondences between the last frame and the previous frames are searched. The relative rotation and the up to scale translation between two frames is computed by using the five-point algorithm. The computation is performed only when there is a stable feature tracking (at least 30 tracked features) and enough parallax (at least 20 pixels) between the frames. The features in the two frames are then triangulated once the scale has been arbitrarily set. At this point, the perspective-n-point (PnP) method is used to estimate the poses of the other frames in the sliding window. In the end, a global bundle adjustment is performed by minimizing the reprojection error of all feature correspondences. In the Structure from Motion process the first camera frame $(\cdot)^{c_0}$ is considered the reference frame. This means that the camera poses $(\bar{p}_{c_k}^{c_0}, q_{c_k}^{c_0})$ and feature locations are computed with respect to the first camera frame. The poses can be translated from the camera frame to the body frame by

using the extrinsic parameters (p_c^b, q_c^b) between the camera and the inertial sensor as follows:

$$\begin{aligned} q_{b_k}^{c_0} &= q_{c_k}^{c_0} \otimes (q_c^b)^{-1} \\ s\bar{p}_{b_k}^{c_0} &= s\bar{p}_{c_k}^{c_0} - R_{b_k}^{c_0} p_c^b \end{aligned} \tag{3.16}$$

Visual-Inertial Alignment

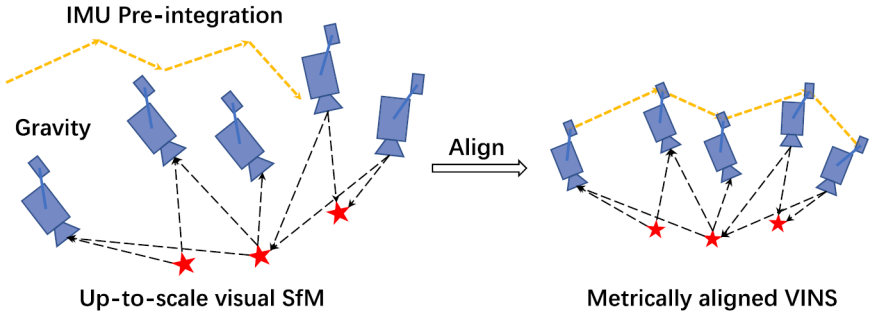

Figure 3.8: Illustration of the visual-inertial alignment - The procedure consists in matching the up to scale vision-only structure and the IMU pre-integration [42].

1) Gyroscope Bias Calibration: The first step of the visual-inertial alignment process is the initial calibration of the gyroscope bias b_w. The gyroscope bias calibration is performed by minimizing the following cost function:

$$\min_{\delta b_w} \sum_{k \in \mathcal{B}} \left\| {q_{b_{k+1}}^{c_0}}^{-1} \otimes q_{b_k}^{c_0} \otimes \gamma_{b_{k+1}}^{b_k} \right\|^2 \tag{3.17}$$

where $q_{b_k}^{c_0}$ and $q_{b_{k+1}}^{c_0}$ are the rotations obtained from the SfM and $\gamma_{b_{k+1}}^{b_k}$ is the relative constraint obtained from IMU pre-integration. Once the calibration of the gyroscope bias is obtained, the pre-integration terms are repropagated by using the new gyroscope bias.

2) Initialization: The second step of the visual-inertial alignment process consists in the initialization of the other states of autonomous navigation, that are velocity, gravity acceleration, and metric scale:

$$\mathcal{X}_I = \left[v_{b_0}^{b_0}, v_{b_1}^{b_1}, ..., v_{b_n}^{b_n}, g^{c_0}, s\right] \tag{3.18}$$

where $v_{b_k}^{b_k}$ is velocity in the body frame while capturing the kth frame, g^{c_0} is the gravity acceleration in the first camera frame, and s is the metric scale. The following relationships are valid for two subsequent frames in the sliding window:

$$\begin{aligned} \alpha_{b_{k+1}}^{b_k} &= R_{c_0}^{b_k}\left[s\left(\bar{p}_{b_{k+1}}^{c_0} - \bar{p}_{b_k}^{c_0}\right) + \frac{1}{2}g^{c_0}\Delta t_k^2 - R_{b_k}^{c_0} v_{b_k}^{b_k}\Delta t_k\right] \\ \beta_{b_{k+1}}^{b_k} &= R_{c_0}^{b_k}\left(R_{b_{k+1}}^{c_0} v_{b_{k+1}}^{b_{k+1}} + g^{c_0}\Delta t_k - R_{b_k}^{c_0} v_{b_k}^{b_k}\right) \end{aligned} \tag{3.19}$$

The equations (3.16) and (3.19) can be combined into the linear measurement model:

$$\hat{z}_{b_{k+1}}^{b_k} = \begin{bmatrix} \alpha_{b_{k+1}}^{b_k} - p_c^b + R_{c_0}^{b_k} R_{b_{k+1}}^{c_0} p_c^b \\ \beta_{b_{k+1}}^{b_k} \end{bmatrix} = H_{b_{k+1}}^{b_k}\mathcal{X}_I + n_{b_{k+1}}^{b_k} \tag{3.20}$$

in which:

$$H_{b_{k+1}}^{b_k} = \begin{bmatrix} -I\Delta t_k & 0 & \frac{1}{2}R_{c_0}^{b_k}\Delta t_k^2 & R_{c_0}^{b_k}(\bar{p}_{c_{k+1}}^{c_0} - \bar{p}_{c_k}^{c_0}) \\ -I & R_{c_0}^{b_k} R_{b_{k+1}}^{c_0} & R_{c_0}^{b_k}\Delta t_k & 0 \end{bmatrix} \tag{3.21}$$

The velocities in the body frame for every image in the sliding window, the gravity acceleration in the camera reference frame, and the scale factor are computed by solving the following linear least-square problem:

$$\min_{\mathcal{X}_I} \sum_{k\in\mathcal{B}} \left\| \hat{z}_{b_{k+1}}^{b_k} - H_{b_{k+1}}^{b_k}\mathcal{X}_I \right\|^2 \tag{3.22}$$

3) Gravity Refinement: The third step of the visual-inertial alignment process consists in the refinement of the gravity vector. The gravity refinement is performed by constraining the magnitude resulting in only two remaining degrees of freedom. The gravity vector is then perturbed by $g(\bar{\hat{g}} + \delta g)$ where g is the magnitude of the gravity, $\bar{\hat{g}}$ is the direction of the gravity, and $\delta g = w_1 b_1 + w_2 b_2$ in which b_1 and b_2 are two orthogonal basis spanning the tangent plane and w_1 and w_2 are the perturbations along b_1 and b_2. The gravity vector in equation (3.19) is substituted by the perturbed configuration. The equation is then solved to find δg and the

other state variables. This iterative process is repeated a lot of times until the convergence is obtained.

4) Completion of the Initialization: The last step of the visual-inertial alignment consists in the computation of the rotation $q_{c_0}^w$ between the world frame and the camera reference frame by rotating the gravity to the z axis. The variables are then rotated to the world frame. In addition, the translational components obtained from the Structure from Motion are scaled to metric units by using the scale factor. The values obtained from the estimator initialization are then ready to feed the Visual-Inertial Odometry thread.

3.2.4 Visual-Inertial Odometry

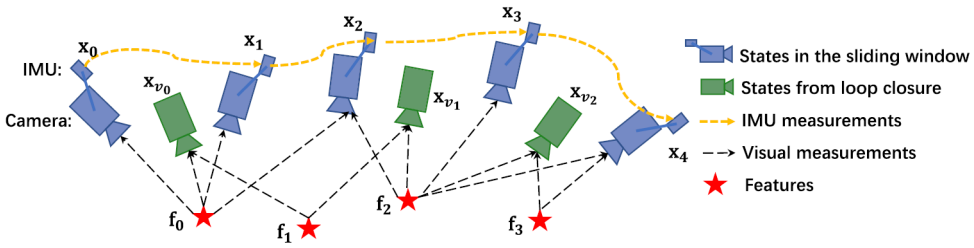

Figure 3.9: Illustration of the Visual-Inertial Odometry process - The Visual-Inertial Odometry module uses a tightly-coupled formulation that fuses the inertial and visual raw measurements for high accuracy and robust pose estimation [42].

The state vector in the sliding window is denoted by the expression:

$$\begin{aligned} \mathcal{X} &= [x_0, x_1, ..., x_n, x_c^b, \lambda_0, \lambda_1, ..., \lambda_m] \\ x_k &= [p_{b_k}^w, v_{b_k}^w, q_{b_k}^w, b_a, b_g], \quad k \in [0, n] \\ x_c^b &= [p_c^b, q_c^b] \end{aligned} \tag{3.23}$$

where x_k is the IMU state that includes pose and velocity in the world frame and accelerometer and gyroscope biases in the body frame, n and m are respectively the number of keyframes and features in the sliding window and λ is the inverse distance from the first observation of a feature. The statement of the problem is performed by using a visual-inertial bundle adjustment formulation. The maximum a posteriori

estimation is obtained by minimizing the sum of prior and the Mahalanobis norm of the IMU and visual measurement residuals as:

$$\min_{\mathcal{X}} \left\{ \|r_p - H_p\mathcal{X}\|^2 + \sum_{k\in\mathcal{B}} \left\| r_{\mathcal{B}}\left(\hat{z}^{b_k}_{b_{k+1}}, \mathcal{X}\right) \right\|^2_{P^{b_k}_{b_{k+1}}} + \sum_{(l,j)\in\mathcal{C}} \rho \left\| r_{\mathcal{C}}\left(\hat{z}^{c_j}_{l}, \mathcal{X}\right) \right\|^2_{P^{c_j}_{l}} \right\} \tag{3.24}$$

IMU Measurement Residual

The residual of the IMU measurements between two consecutive frames b_k and b_{k+1} in the sliding window is defined as:

$$r_{\mathcal{B}}(\hat{z}^{b_k}_{b_{k+1}}, \mathcal{X}) = \begin{bmatrix} \delta\alpha^{b_k}_{b_{k+1}} \\ \delta\beta^{b_k}_{b_{k+1}} \\ \delta\theta^{b_k}_{b_{k+1}} \\ \delta b_a \\ \delta b_g \end{bmatrix} = \begin{bmatrix} R^{b_k}_w \left(p^w_{b_{k+1}} - p^w_{b_k} + \frac{1}{2} g^w \Delta t_k^2 - v^w_{b_k} \Delta t_k\right) - \hat{\alpha}^{b_k}_{b_{k+1}} \\ R^{b_k}_w \left(v^w_{b_{k+1}} + g^w \Delta t_k - v^w_{b_k}\right) - \hat{\beta}^{b_k}_{b_{k+1}} \\ 2\left[{q^w_{b_k}}^{-1} \otimes q^w_{b_{k+1}} \otimes {\hat{\gamma}^{b_k}_{b_{k+1}}}^{-1}\right]_{xyz} \\ b_{a_{b_{k+1}}} - b_{a_{b_k}} \\ b_{w_{b_{k+1}}} - b_{w_{b_k}} \end{bmatrix} \tag{3.25}$$

where $\alpha^{b_k}_{b_{k+1}}$, $\beta^{b_k}_{b_{k+1}}$ and $\gamma^{b_k}_{b_{k+1}}$ are the IMU pre-integration terms and $\delta\theta^{b_k}_{b_{k+1}}$ is the 3D error state representation of a quaternion.

Visual Measurement Residual

The majority of cameras, from wide angle to omnidirectional cameras, are modeled by using a unit ray connecting the surface of a unit sphere. The residual of visual measurements is then defined on a unit sphere as:

$$\begin{aligned} r_{\mathcal{C}}\left(\hat{z}^{c_j}_{l}, \mathcal{X}\right) &= [b_1\ b_2]^T \cdot \left(\hat{\bar{\mathcal{P}}}^{c_j}_{l} - \frac{\mathcal{P}^{c_j}_{l}}{\|\mathcal{P}^{c_j}_{l}\|} \right) \\ \hat{\bar{\mathcal{P}}}^{c_j}_{l} &= {\pi_c}^{-1} \begin{bmatrix} \hat{u}^{c_j}_{l} \\ \hat{v}^{c_j}_{l} \end{bmatrix} \\ P^{c_j}_{l} &= R^c_b \left\{ R^{b_j}_w \left[R^w_{b_i} \left(R^b_c \frac{1}{\lambda_l} {\pi_c}^{-1} \begin{bmatrix} \hat{u}^{c_i}_{l} \\ \hat{v}^{c_i}_{l} \end{bmatrix} + p^b_c \right) + p^w_{b_i} - p^w_{b_j} \right] - p^b_c \right\} \end{aligned} \tag{3.26}$$

where $[\hat{u}^{c_i}_l, \hat{v}^{c_i}_l]$ and $[\hat{u}^{c_j}_l, \hat{v}^{c_j}_l]$ are respectively the first observation of the lth feature in the ith image and the observation of the same feature in the jth image, ${\pi_c}^{-1}$ is the camera back projection model and b_1 and b_2 are two orthogonal bases spanning the tangent plane of $\hat{\bar{\mathcal{P}}}^{c_j}_{l}$.

Marginalization

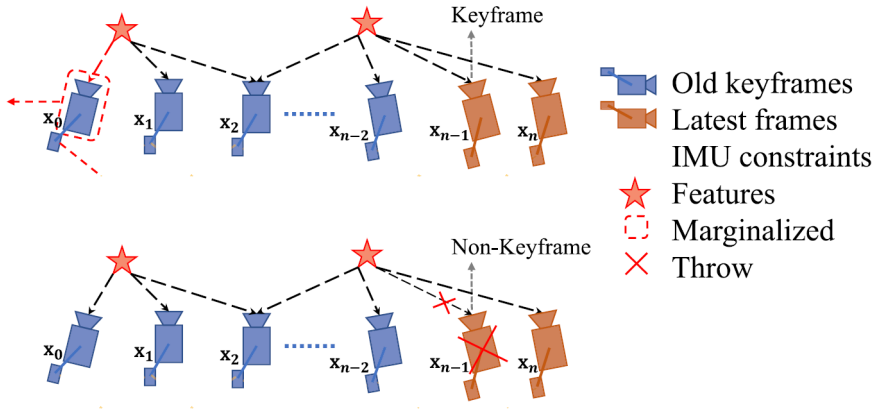

Figure 3.10: Illustration of the marginalization strategy - When the penultimate frame in the sliding window is a keyframe, the oldest frame is marginalized out along with its visual and inertial measurements. Conversely, when the penultimate frame in the sliding window is a non-keyframe, the marginalization of the oldest frame is not performed because the new frame is removed by storing only its inertial measurements [42].

The strategy of marginalization is adopted to reduce the computational burden of the algorithm. The marginalization process is now briefly described. When the penultimate frame in the sliding window is a keyframe, the oldest frame is marginalized out along with its visual and inertial measurements. The marginalized measurements are converted into a prior. Conversely, when the penultimate frame in the sliding window is a non-keyframe, the marginalization of the oldest frame is not performed because the new frame is removed by storing only its inertial measurements. The inertial measurements of non-keyframes are not marginalized out to preserve the sparsity of the system. The marginalization strategy is used to obtain sufficiently separated keyframes in the sliding window. The separation of the keyframes is required to have a sufficient amount of parallax for feature triangulation. The marginalization is performed by using the Schur complement.

3.2.5 Relocalization

The marginalization strategy causes a significant accumulated drift in the state estimation. The integration of a relocalization process in the Visual-Inertial

Odometry module is performed to remove the drifting. The relocalization process is based on loop closure detection. Loop closing is the process of detecting whether the robot has returned to a previously visited location. The feature correspondences between the last frame and the loop closure candidates are integrated in the Visual-Inertial Odometry module resulting in a highly accurate and robust state estimation. The relocalization procedure is now described in detail.

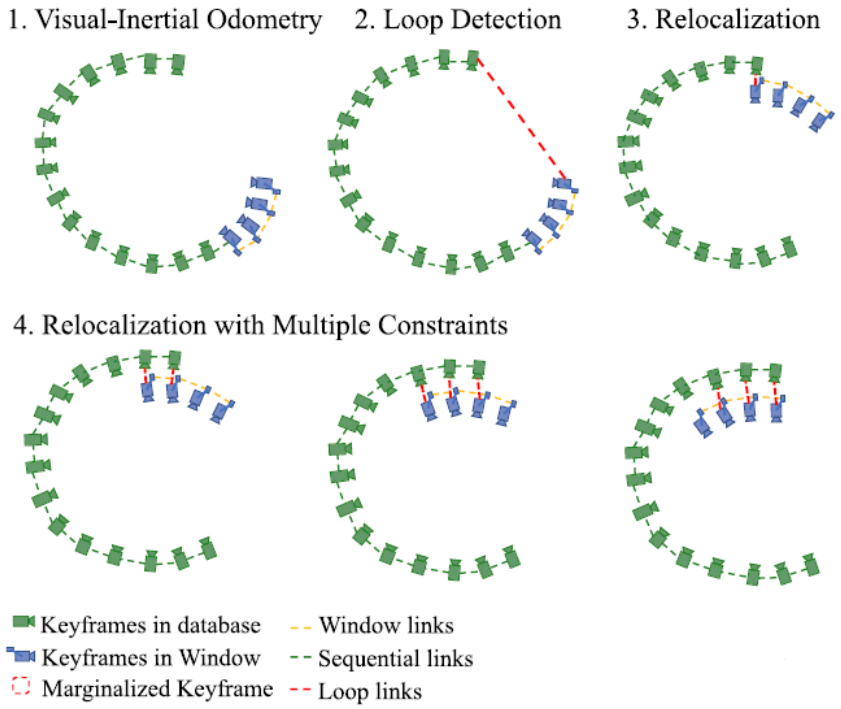

Figure 3.11: Illustration of the relocalization procedure - The first three plots show that relocalization is performed whenever a loop is detected for the last keyframe in the window. The last three plots show that the loop constraints can be incorporated for multiple keyframes in the database because of the use of feature correspondences for relocalization [42].

Loop Detection

The loop detection is performed by using a place recognition approach called DBoW2. This approach uses the ORB feature detector and descriptor to detect and describe 500 more features in addition to the features used in the Visual-Inertial Odometry process. The additional features are required to obtain high precision

in loop detection. The output of the algorithm are the loop closure candidates. During loop detection, the descriptors are stored for feature retrieval while the raw images are removed to bound the memory load.

Feature Retrieval

The feature retrieval step consists in finding the feature correspondences between the newest frame in the sliding window and the loop closure candidate. The process of feature matching generally results in false correspondences. The outlier removal is then performed in two steps by using the RANSAC method:

1) 2-D–2-D: The first step is the fundamental matrix test performed by using the 2-D observations of retrieved features in the newest frame and loop closure candidate.

2) 3-D–2-D: The second step is the PnP test performed by using the 3-D locations of features in the sliding window and the 2-D observations of features in the loop closure candidate.

After the process of outlier removal, the loop closure candidate is considered a correct loop detection so the relocalization is performed.

Relocalization

The relocalization process consists in the alignment of the sliding window to the oldest poses. During relocalization, the optimization of the sliding window is performed by using inertial and visual measurements and retrieved feature correspondences. The visual measurement residual of the retrieved features observed in a loop closure frame v is identical to that defined in the Visual-Inertial Odometry thread. The only difference is that the pose of the loop closure frame $(\hat{q}_v^w, \hat{p}_v^w)$ in this case is considered constant. Therefore, the nonlinear cost function in the equation (3.24) is modified by adding a loop term representing the reprojection error in the loop closure frame as follow:

$$\min_{\mathcal{X}} \left\{ \dots + \sum_{(l,v)\in\mathcal{L}} \rho \left\| r_{\mathcal{C}}\left(\hat{z}_l^v, \mathcal{X}, \hat{q}_v^w, \hat{p}_v^w\right) \right\|_{P_l}^{2\ c_v} \right\} \tag{3.27}$$

where $\mathcal{L}$ is the set of the features in the loop closure frames and (l, v) indicates the lth feature observed in the loop closure frame v. Note that the dimension of the states to be solved remains unchanged because the poses of loop closure frames are considered constant. The optimization is performed by using the feature correspondences from multiple loop closure frames simultaneously. The

establishment of loop constraints in multiple frames of the sliding window results in a high accuracy and robust relocalization.

3.2.6 Global Pose Graph Optimization

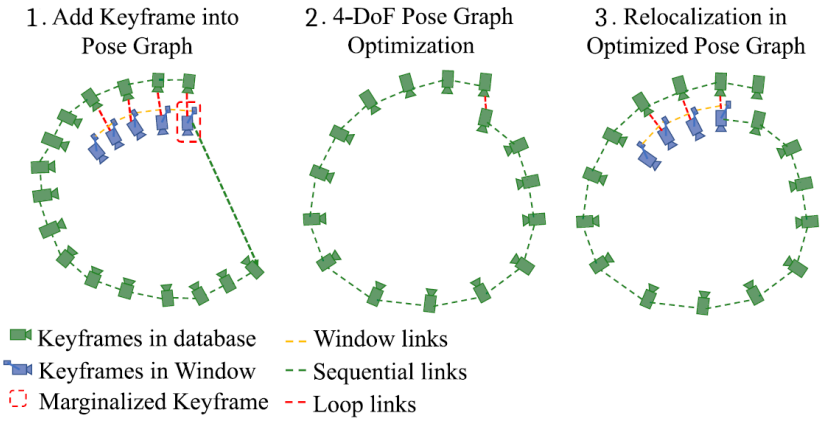

Figure 3.12: The global pose graph optimization - First of all, the pose graph is updated whenever a keyframe is marginalized out from the sliding window. The loop closure constraints between the new keyframe and any other past keyframe in the pose graph are also recorded. The global pose graph optimization is performed by using all relative constraints between the poses. The relocalization is performed by always using the latest optimized pose graph [42].

The global pose graph optimization is performed to obtain global consistency of the path. The roll and pitch angles are absolute states in the world frame that can be determined by the horizontal plane from the gravity vector. Conversely, the x,y,z and yaw angle change relatively with respect to the reference frame. This means that the drift is accumulated only in x,y,z, and yaw states. The global pose graph optimization is then only performed in 4 DOF.

Keyframe Addition in the Pose Graph

The pose graph is articulated in a large number of keyframes represented by vertexes connected with one another by two different types of edges:

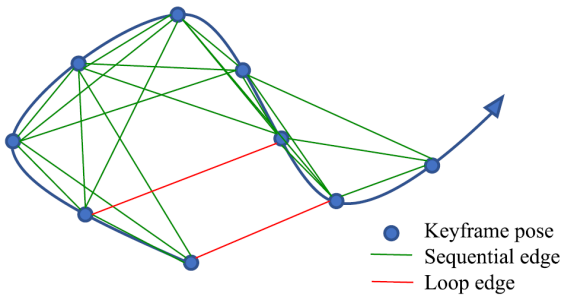

Figure 3.13: Illustration of the pose graph - A large number of keyframes represented by vertexes are connected with one another by sequential edges and loop edges [42].

1) Sequential Edge: A sequential edge consists in the relative transformation between two keyframes computed during the Visual-Inertial Odometry process. The sequential edge between a keyframe i and a keyframe j is composed by the relative position $\hat{p}_{ij}^{i}$ and the yaw angle $\hat{\psi}_{ij}$ defined as:

$$\begin{aligned} \hat{p}_{ij}^{i} &= \hat{R}_{i}^{w-1}\left(\hat{p}_{j}^{w} - \hat{p}_{i}^{w}\right) \\ \hat{\psi}_{ij} &= \hat{\psi}_{j} - \hat{\psi}_{i} \end{aligned} \tag{3.28}$$

2) Loop Edge: A loop edge consists in the relative transformation between the keyframe and the loop closure frame. The loop edge is composed by the relative position and the yaw angle between two keyframes as the sequential edge. The value of the loop edge is determined in the relocalization process.

Pose Graph Optimization

The residual of the edge connecting the keyframes i and the keyframe j is defined as:

$$r_{i,j}(p_{i}^{w}, \psi_{i}, p_{j}^{w}, \psi_{j}) = \begin{bmatrix} R(\hat{\phi}_{i}, \hat{\theta}_{i}, \hat{\psi}_{i})^{-1}(p_{j}^{w} - p_{i}^{w}) - \hat{p}_{ij}^{i} \\ \psi_{j} - \psi_{i} - \hat{\psi}_{ij} \end{bmatrix} \tag{3.29}$$

where $\hat{\phi}_{i}$ and $\hat{\theta}_{i}$ are the drift-free estimates of roll and pitch angles obtained in the Visual-Inertial Odometry process. The pose graph optimization of sequential edges and loop closure edges is performed by minimizing the following cost function:

$$\min_{p,\psi}\left\{\sum_{(i,j)\in\mathcal{S}}\|r_{i,j}\|^2+\sum_{(i,j)\in\mathcal{L}}\rho\|r_{i,j}\|^2\right\} \tag{3.30}$$

where $\mathcal{S}$ is the set of the sequential edges and $\mathcal{L}$ is the set of the loop closure edges. The Huber norm $\rho(\cdot)$ for loop edges is used to limit the amount of wrong loop closures. Conversely, the use of the Huber norm is not necessary for sequential edges that are obtained from the Visual-Inertial Odometry process which is in itself equipped with robust processes of outlier removal. The pose graph optimization and the relocalization are performed separately in two different modules. This means that the relocalization is carried out by always using the newest optimized pose graph.

Pose Graph Merging

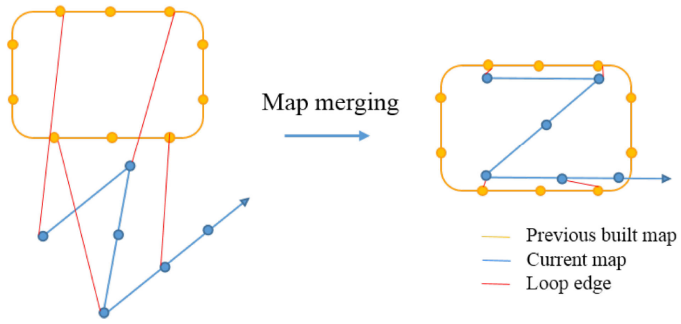

Figure 3.14: Map merging - The yellow graph represents the map previously built and the blue graph represents the newest map. The map merging is performed on the basis of the loop closure edges. [42].

The pose graph merging consists in the fusion of the newest map and a map previously built. The map merging is performed on the basis of the loop closure edges available between the two maps. As depicted in Figure 3.14, the newest map is pulled into the previous map by using loop connections. Since the vertexes and edges are relative variables, the pose graph merging is performed by only fixing the first vertex of the pose graph.

3.2.7 Experimental results

The experimental results obtained by running the algorithm on a laptop set up with ROS Melodic are displayed in Rviz. The left side of the figure shows the left and right input images of the camera in a stereo setup. The right side of the figure shows the output of the algorithm that consists of the odometry data represented by a green line and the point cloud map of the environment represented by green and white sparse points. In addition, it is possible to observe the left and right camera lenses represented by two red rectangles and the system reference frame represented by the triad of red, green and blue axes.

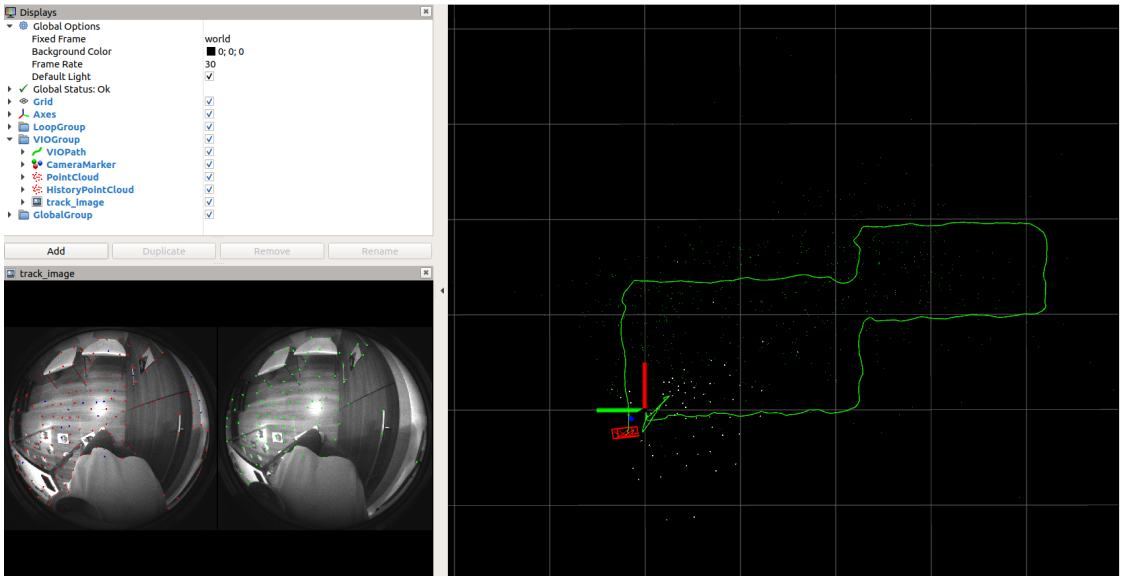

Figure 3.15: Illustration of the algorithm performance in Rviz - The output of the algorithm consists of the odometry data represented by a green line and the point cloud map of the environment represented by green and white sparse points

3.3 ORB-SLAM2

This section presents ORB-SLAM2 that is the first open-source SLAM solution for monocular, stereo and RGB-D cameras. The system is able to compute in real-time the camera trajectory and the map of the environment in a wide variety of scenarios. The most important features of ORB-SLAM2 are an automatic and robust initialization, loop closing, relocalization and map reuse. The system is also equipped with a localization mode that is used for a lightweight and long-term localization with the local mapping and loop closing threads disabled. The

implementation of the algorithm is presented by Mur-Artal et al. in their paper "ORB-SLAM: a versatile and accurate monocular SLAM system" [43].

3.3.1 System Overview

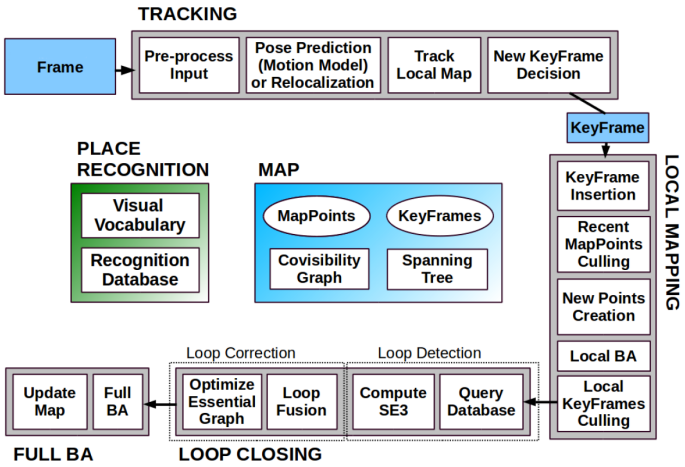

Figure 3.16: The architecture of the algorithm is composed of three different threads: tracking, local mapping, loop closing [44].

Architecture of the algorithm

The architecture of the algorithm is composed of three parallel threads: tracking, local mapping and loop closing. The loop closing thread is meant to launch a full bundle adjustment after a loop closure to compute a robust and accurate solution. The system uses ORB features because they are extremely fast to extract and match allowing for real time applications. In addition, they are robust to rotation and scale and invariant to viewpoint and illumination changes. The goals of the tracking are the localization of the camera and the insertion of new keyframes. First of all, an initial estimation of the camera pose is computed by matching features with the previous frame. The pose is then optimized by using motion-only BA. It may happen to loose the tracking because of occlusions or aggressive motion. The place recognition module is then activated to perform a global relocalization. Secondly, a local map is constructed using the covisibility graph. The correspondences between the features in the frame and the local map points are searched by minimizing the reprojection error. The optimization of the camera pose is then performed once more by using all matches. Finally, the tracking thread is responsible for the insertion of new keyframes. The goals of the

local mapping are the processing of new keyframes and the optimization of the local map by using a local BA. The correspondences between the unmatched features in the new keyframe and the features in the other keyframes of the covisibility graph are searched for the triangulation of new points. The local mapping thread consists also in the application of a point and keyframe culling policy to maintain only high-quality points and remove redundant keyframes. The goal of the loop closing is the detection of loop closures. A similarity transformation carrying the information about the drift accumulated in the loop is computed after the detection of a loop closure. The alignment of the two sides of the loop is then performed by fusing doubled points in the map. Eventually a pose graph optimization is performed using the essential graph to obtain global consistency of the path.

Covisibility Graph and Essential Graph

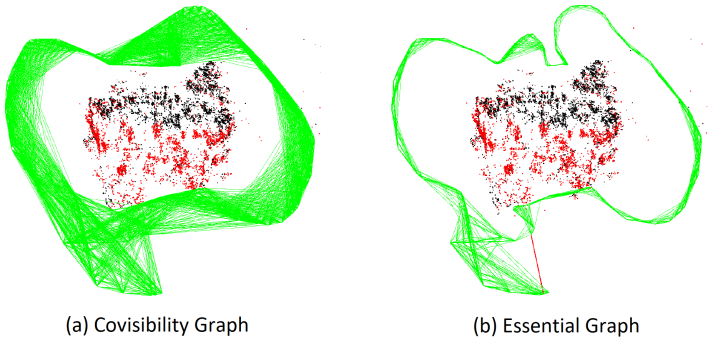

Figure 3.17: Illustration of the covisibility graph (a) and the essential graph (b) - The essential graph is a sparse subgraph of the covisibility graph in which only the subset of edges with higher covisibility is maintained [43].

The covisibility graph and the essential graph are graph structures that are useful in different parts of the system. The covisibility graph is an undirected weighted graph in which the nodes represent the keyframes and the edges between the nodes are established only if the keyframes observe a certain number of the same map points. The weight of the edge is the number of common map points. The essential graph is a sparse subgraph of the covisibility graph that is composed of the same number of nodes and a minimal number of edges. In fact, only the subset of edges of the covisibility graph with higher covisibility is maintained. This graph structure is still a strong network of nodes that brings high accuracy and robust results. The structure of both these graphs is realized from a spanning tree of keyframes.

3.3.2 Tracking

In this section, the steps of the tracking thread are described in detail. They are: input pre-processing, map initialization, initial pose estimation, local map tracking, and new keyframe decision. The map initialization is the only step to be performed once at the beginning of the process. The other steps of the tracking thread are performed with every frame captured from the camera.

Input Pre-processing

The first step of the tracking thread is the input pre-processing. The system pre-processes the input images by extracting features at salient keypoint locations. The process of feature extraction is performed at 8 different scale levels obtained by scaling the image with a scale factor of 1.2. The main requirements for robust and stable motion estimation are a large number of features and a uniform distribution of the features in the image. The number of features ranges from 1000 to 2000 depending on the image resolution. The uniform distribution of the features in the image is obtained by extracting at least 5 features in every cell of the grid in which the image is partitioned. The minimum threshold of features fixed for every cell of the image can be adapted if not enough features are found. The features are then associated with a ORB descriptor that improves the accuracy of the feature matching. The input images are then eliminated so that the rest of the system operates independently of the type of camera.

Map Initialization

The map initialization is a procedure for the triangulation of an initial set of map points. The procedure is only required in the monocular case because in the stereo or RGB-D case the depth information is obtained from just one frame so that the map can be immediately created. The goal of the initialization is the computation of the pose between two frames. The main novelty of the process is the computation of two geometrical models in parallel threads to ensure that the procedure is independent of the scene. The geometrical models are a homography assuming a planar scene and a fundamental matrix assuming a non planar scene. The selection of the model is performed by using a heuristic. The relative pose between the frames is computed by using a specific method according to the choice of the model. The triangulation of the points is then performed by using the relative pose. The method is able to detect the inconsistencies of the models such as the twofold planar ambiguity and low-parallax configurations in order to avoid the initialization of a corrupted map.

Initial Pose Estimation

The initial pose estimation is performed by using a constant velocity motion model. The process of feature matching is accomplished by searching for the map points observed in the previous frame. The optimization of the pose is then performed with all the found correspondences by using the motion-only BA. The motion-only BA performs the optimization of the camera pose by minimizing the reprojection error between the map points $X^i \in \mathbb{R}^3$ and keypoints $x^i_{(.)} \in \mathbb{R}^2$:

$$\{R,t\} = \underset{R,t}{\operatorname{argmin}} \sum_{i \in \mathcal{X}} \rho \left(\left\| x^i_{(.)} - \pi_{(.)}(RX^i + t) \right\|^2_{\Sigma} \right) \tag{3.31}$$

where ρ is the Huber function and Σ is the covariance matrix associated to the scale of the keypoint. The projection function $\pi_{(.)}$ is defined as:

$$\pi \left(\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \right) = \begin{bmatrix} f_x \frac{X}{Z} + c_x \\ f_y \frac{Y}{Z} + c_y \end{bmatrix} \tag{3.32}$$

where f_x and f_y are the focal lenghts and c_x and c_y are the principal point coordinates. In case of tracking failure the place recognition module performs a relocalization by using DBoW2. The frame is converted in a bag of words. The query of the database results in different keyframe candidates. The process is characterized by a high computational efficiency because the database includes an index that informs in which keyframe every visual word of the vocabulary has been observed. In addition the database returns all the keyframe matches that have a score being over the 75% of the best score. The correspondences are then refined by using an orientation consistency test for outlier removal. The camera pose is then computed by using the PnP algorithm for every keyframe in a RANSAC scheme. A guided search of the map points observed in the candidate keyframe is performed to increase the number of matches. Finally, the optimization of the camera pose is performed by using all the matches found.

Local Map Tracking

Once the computation of the pose and the initial set of feature matches is obtained, the projection of the local map into the frame is performed in order to search for new map point correspondences. The local map is composed of the set of keyframes $\mathcal{K}_1$ that have common map points with the current frame and the set of keyframes $\mathcal{K}_2$ that are neighbors to the keyframes of the previous set in the covisibility graph.

In the first set of keyframes it is possible to identify a reference keyframe $\mathrm{K_{ref}}$ that has the highest number of common map points with the frame. The algorithm that searches in the frame every map point observed in $\mathcal{K}_1$ is articulated in the following steps. First of all, the projection of the map points in the frame is computed. The points that lay out of the image bounds are discarded. Secondly, the computation of the angle between the viewing ray of the frame v and the map point viewing direction n is performed. The points for which $\mathrm{v} \cdot \mathrm{n} < \cos(60°)$ are removed. Thirdly, the distance d from the map point to the camera center is computed. The points whose distance is out of the scale invariance region $d \notin [d_{min}, d_{max}]$ are discarded. The bounds of the scale invariance region are the minimum and the maximum distances at which the point can be observed. Fourthly, the computation of the scale in the frame is performed by the ratio d/d_{min}. The descriptor of the map point is compared with the descriptor of the unmatched features in the frame at the estimated scale. Finally, the map point is associated with the best match. The optimization of the camera pose is then performed by using all the map points found in the image.

New Keyframe Decision

The last step of the tracking thread consists in deciding if the current frame can be considered a keyframe. The keyframe insertion is performed as fast as possible to obtain a robust and accurate tracking against the challenging movements of the camera. This process is possible only because in the local mapping thread a culling policy is performed to remove redundant keyframes. There are four conditions that must be satisfied to insert a new keyframe. First of all, the local mapping is inactive or at least 20 keyframes must have passed from the last keyframe insertion. Secondly, the last global relocalization has been performed more than 20 frames earlier. Thirdly, the points tracked in the current frame are at least 50. Finally, the points tracked in the current frame are less than the 90% of the points tracked in the reference frame $\mathrm{K_{ref}}$. The main novelty of the ORB-SLAM system with respect to the other state of the art systems is the usage of a minimum visual change criterion that is represented by the last condition.

3.3.3 Local Mapping

In this section, the steps of the local mapping thread are described in detail. They are: keyframe insertion, recent map points culling, new points creation, local bundle adjustment and local keyframes culling. These steps are performed by the local mapping with every new keyframe K_i.

Keyframe Insertion

The insertion of a new keyframe requires the accomplishment of different tasks. The covisibility graph and the essential graph have to be updated every time a new keyframe is inserted. The update of these graphs is performed by adding a node representing the new keyframe and updating the edges connecting the new keyframe with the older keyframes. The computation of the bag of words representation of the keyframe is also performed to achieve better accuracy in the data association for the triangulation of new points in the map. The map points are then ready to undergo a point culling policy that maintains only high quality points in the map.

Recent Map Points Culling

The point culling policy consists in the submission of a test to the map points during the first three keyframes after the triangulation. The test ensures that the map points are high quality points that are easily trackable. There are two conditions that have to be satisfied to retain a point in the map. The first condition states that the tracking must find the point at least in the 25% of the frames in which the point is supposed to be observed. The second condition states that the point must be in the field of view of at least three keyframes if more than one keyframe has passed from the triangulation of the point. The point that at any time is observed from not more than two keyframes can be discarded even if it has passed the test. This situation could happen when the local bundle adjustment removes the outliers or when the keyframe culling policy is applied.

New Map Point Creation

The triangulation of new map points is performed from the features of the keyframes $\mathcal{K}_c$ that are connected in the covisibility graph. The process of feature matching is performed by using a bag of words approach as described in the Initial Pose Estimation paragraph. For every unmatched feature in the new keyframe K_i a match is searched among the unmatched features of the other keyframes. The matches that do not satisfy the epipolar constraint are clearly discarded. The triangulation of the points is then performed from the feature pairs found. The new map points are accepted only after a parallax, reprojection error and scale consistency check. The projection of the map point is performed in the rest of the keyframes of the covisibility graph. The correspondences are then searched by using the method described in the Local Map Tracking paragraph.

Local Bundle Adjustment

The local bundle adjustment is a process that consists in the optimization of the new keyframe K_i and the other covisible keyframes $\mathcal{K}_c$. The optimization is performed also on the map points observed by those keyframes $\mathcal{P}_c$. The other keyframes $\mathcal{K}_F$ that observe the same map points but are not connected to the new keyframe in the covisibility graph are fixed in the optimization. The optimization problem is defined as:

$$\{X^i, R_l, t_l | \, i \in \mathcal{P}_c, l \in \mathcal{K}_c\} = \underset{X^i, R_l, t_l}{\operatorname{argmin}} \sum_{K_i \in \mathcal{K}_c \cup \mathcal{K}_F} \sum_{j \in \mathcal{X}_k} \rho(E_{K_i j}) \tag{3.33}$$

with:

$$E_{K_i j} = \left\| x^j_{(.)} - \pi_{(.)}(R_{K_i} X^j + t_{K_i}) \right\|^2_{\Sigma} \tag{3.34}$$

where $\mathcal{X}_k$ is the set of matches between the features in the new keyframe K_i and the map points observed by the covisible keyframes $\mathcal{P}_c$. The outlier observations are removed at the middle and at the end of the refinement process.

Local Keyframe Culling

The keyframe culling policy consists in the removal of redundant keyframes. This process is advantageous for several reasons: for example, it bounds the bundle adjustment complexity that grows with the number of keyframes and allows a long-lasting performance in the same environment because there is not a uncontrolled growth of the number of keyframes. The keyframes in which the 90% of the map points have been observed in at least other three keyframes at the same scale or a smaller one are removed. The scale condition is necessary to preserve those keyframes from which the map points are computed with the best accuracy.

3.3.4 Loop Closing

The loop closing thread is responsible for the detection and closure of loops for the last keyframe K_i. The steps in which the loop closing thread is articulated are: detection of loop candidates, computation of the similarity transformation, loop fusion, and essential graph optimization. In addition, the loop closing thread is meant to launch a full bundle adjustment after the pose graph optimization to compute a robust and accurate solution. The steps of the loop closing are detailed in this section.

Loop Candidates Detection

The first step of the loop closing thread is the detection of loop candidates. The computation of the similarity between the bags of words of the current keyframe and the covisible keyframes that share with it at least 30 observations of map points is performed. The lowest score of similarity is then stored. The database is queried for loop candidates. The keyframes that have a score lower than the stored one are discarded. Moreover the keyframes that are directly conneted to the current keyframe in the covisibility graph are removed. The validation of a loop candidate occurs only if three loop candidates that are consistent are sequentially detected. The covisibility information is so used to obtain robustness and accuracy in the process of loop closing.

Computation of the Similarity Transformation

The second step of the loop closing thread is the computation of the similarity transformation. The similarity transformation between the current keyframe K_i and the loop candidate K_l contains the information about the drift that is accumulated in the loop. The process of feature matching is performed between the current keyframe and the loop candidate following the method used in case of tracking failure in the Initial Pose Estimation paragraph. The computation of the transformation is then performed by using the found 3-D to 3-D correspondences. Alternatively the similarity transformation is estimated by using the method of Horn in a RANSAC scheme. The optimization of the pose is then performed after a guided search of more correspondences. The loop candidate K_j is accepted if the similarity transformation contains a sufficient number of inliers.

Loop Fusion

The third step of the loop closing thread is loop correction. The pose of the current keyframe T_{iw} is corrected by using the similarity transformation S_{ij}. The propagation of the correction is then performed to the neighbors of the current keyframe in order to perfectly align the two sides of the loop. The map points observed by the loop candidate and its neighbors are projected into the current keyframe and its neighbors. The search of the matches is performed in the area around the projection of the points as explained in the Local Map Tracking paragraph. Finally, the fusion of the duplicated map points and the update of the covisibility graph is performed. The covisibility graph is updated by adding loop closure edges to the keyframes involved in the loop correction.

Essential Graph Optimization

The final step of the loop closing thread is the essential graph optimization. The pose graph optimization is performed over the essential graph to distribute the loop closing error along the graph. The error in an edge of the pose graph is defined as:

$$e_{i,j} = \log_{\mathrm{Sim}(3)}(S_{ij}\, S_{jw}\, S_{iw}^{-1}) \tag{3.35}$$

The $\log_{\mathrm{Sim}(3)}$ transforms the pose in the tangent space to obtain the error in seven degrees of freedom. The seven degrees of freedom are three translations, three rotation and a scale factor. This is valid for the monocular case in which the scale is unknown. The optimization of the keyframe poses is performed by minimizing the following cost function:

$$C = \sum_{i,j}(e_{i,j}^{T}\, \Lambda_{i,j}\, e_{i,j}) \tag{3.36}$$

where $\Lambda_{i,j}$ is the information matrix of the edge that is set to the identity. This method is an approximation of a full bundle adjustment, but it demonstrates to converge faster and better than it. After the optimization, the map points are transformed on the basis of the correction of one of the keyframes observing those points.

Full Bundle Adjustment

The loop closing thread launches a full bundle adjustment after a loop closure to compute a robust and accurate solution. In the full bundle adjustment, all points and keyframes in the map are optimized with the exception of the first keyframe that remains fixed. This optimization is performed in a separate thread to allow the system to continue map creation and loop detection. The decision to perform the optimization in a separate thread requires the capability to merge the output of the optimization with the latest state of the map. The detection of a new loop implies the abortion of the optimization. The full bundle adjustment is then launched again after the loop closure. Once the optimization is completed, the updated set of keyframes and points is fused with the non-updated set of keyframes and points that were added during the optimization. This process is performed by propagating the correction to the non-updated set of keyframes in the spanning tree. The non-updated set of points is then corrected on the basis of the transformation applied to the reference frame.

3.3.5 Experimental results

Unfortunately bad experimental results are obtained by running the algorithm in a monocular setup. The algorithm suffers from severe scale drift and easily gets lost in areas poor of features. These problems are mainly due to the lack of the inertial sensor. The adoption of an inertial sensor can definitely solve the above problems as it is scene-independent and provides metric information.

3.4 ORB-SLAM3

This section presents ORB-SLAM3 that is the first open-source SLAM library capable of performing Visual, Visual-Inertial, and Multiple Map SLAM with different types of sensors, including monocular, stereo, and RGB-D cameras. The system is inspired by ORB-SLAM2 [44] and ORB-SLAM-VI [45]. There are several novelties with respect to the previous approaches. The first novelty is a monocular and stereo tightly-coupled visual-inertial SLAM solution that performs accurate and robust pose estimation in a wide variety of environments. The second novelty is the creation of ORB-SLAM Atlas that is the first multi-map SLAM system with a high-recall place recognition that allows the system to survive to long periods of poor visual information. The third novelty is the abstraction of the camera model that lets the system use any type of camera by simply providing the corresponding camera module. The implementation of the algorithm is presented by Campos and Elvira in their paper "ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM" [46].

3.4.1 Algorithm architecture

The architecture of the algorithm introduces a lot of novelties with respect to the previous approach. The main novelty is the introduction of the Atlas multi-map representation in the system. The set of disconnected maps represented includes the so-called active map. The active map is the map in which the last frames are localized. This map is constantly enlarged and improved by the local mapping thread. The other maps in the Atlas are called non-active maps. The Atlas includes also a database of keyframes that is used in different operations such as relocalization, loop closure, and map merging. The main goal of the tracking thread is the pose estimation. The computation of the pose is performed by minimizing the reprojection error of the map points observed in the current frame. The other tasks of the tracking thread are the processing of visual and inertial information and the new keyframe decision. The tracking thread performs relocalization by using all the maps in the Atlas in case of tracking failure. The relocalization is performed

by using the Maximum Likelihood Perspective-n-Point algorithm (MLPnP) that works independently of the camera model used as it uses projective rays as input. The algorithm only requires the back-projection function that passes from pixels to projection rays to perform relocalization. The active map is switched if the frame is relocated in a non-active map. Conversely, if the frame hasn't been relocated in any of the maps in the Atlas, the initialization of a new active map is performed by storing the last active map as non-active. The goals of the local mapping thread are the insertion of new keyframes and points in the active map and the optimization of the map by using a visual or visual-inertial local bundle adjustment. The other tasks of the local mapping thread are the IMU initialization and the application of a point and keyframe culling policy. The main goal of the loop and map merging thread is the detection of common areas between the maps in the Atlas. The loop correction is immediately performed if the detection of a common area involves the active map. Otherwise, the loop correction is performed after the non-active maps involved in the detection of the common area are merged into a single map that becomes the active map. The loop and map merging thread launches a full bundle adjustment in a separate thread after a loop correction to compute a robust and accurate solution.

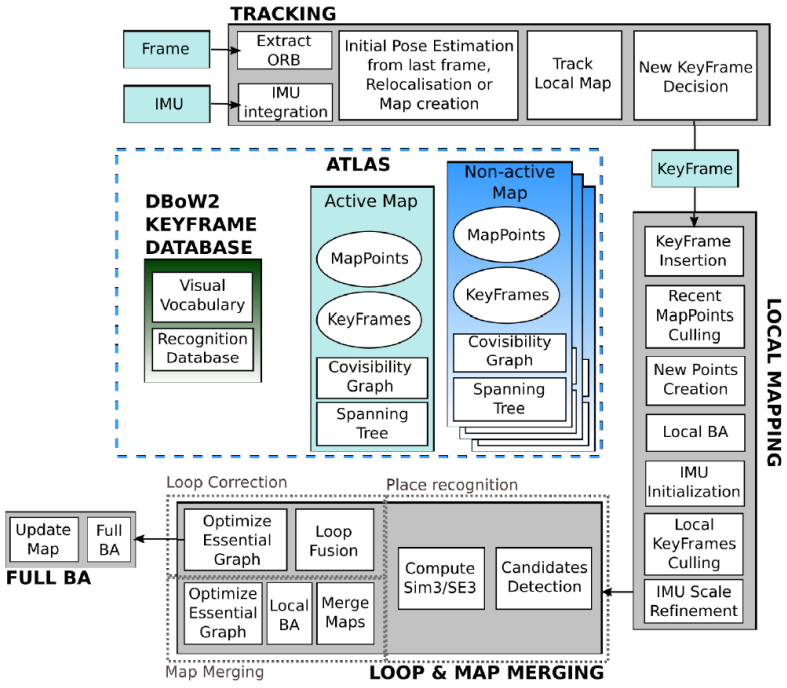

Figure 3.18: The architecture of the algorithm is composed of the Atlas multi-map representation and three parallel threads: tracking, local mapping, and loop and map merging

3.4.2 Visual-Inertial SLAM

This section describes in detail the main features of the first open-source tightly-coupled visual-inertial SLAM system capable of performing accurate and robust pose estimation with a wide variety of cameras.

IMU Initialization

The visual-inertial SLAM requires a fast and accurate initial estimation of the visual and inertial parameters to obtain robustness and accuracy in the solution. The goal of this step is the computation of the initial values of the inertial parameters such as body velocities, gravity direction, and IMU biases. The IMU initialization is articulated in the following steps:

1) Vision-only MAP Estimation: The first step of the initialization is solving the vision-only problem to improve the performance of the initialization. A pure monocular SLAM is run for about 2 seconds to obtain a robust and accurate initial map. The only problem of this map is the scale ambiguity that will be solved in the next step. The result of this process is an up-to-scale map that consists of about ten camera poses and hundreds of points. The optimization of the map is performed by using a visual-only BA. The transformation of the poses to the body reference is then performed obtaining the trajectory $\overline{T}_{0:k} = [R, \bar{p}]_{0:k}$.

2) Inertial-only MAP Estimation: The second step of the initialization is the estimation of the inertial parameters that is performed in the form of an inertial-only MAP estimation by using only the trajectory $\overline{T}_{0:k}$ and the inertial measurements between the keyframes. The inertial parameters are represented by the inertial-only state vector:

$$\mathcal{Y}_k = \{s, R_{wg}, b, \bar{v}_{0:k}\} \tag{3.37}$$

where s is the scale factor, R_{wg} is the gravity direction, $b = (b^a, b^g)$ are the accelerometer and gyroscope biases and $\bar{v}_{0:k}$ are the body velocities estimated from the corresponding trajectory $\overline{T}_{0:k}$. The inertial-only MAP estimation problem results in the optimization problem:

$$\mathcal{Y}_k^* = \underset{\mathcal{Y}_k}{\operatorname{argmin}} \left(\|r_p\|^2_{\Sigma_p} + \sum_{i=1}^{k} \|r_{\mathcal{I}_{i-1,i}}\|^2_{\Sigma_{\mathcal{I}_{i-1,i}}} \right) \tag{3.38}$$

where $r_{\mathcal{I}_{i-1,i}}$ is the inertial residual and r_p is a prior residual imposing that IMU biases are close to zero with a covariance specified by the IMU characteristics. The update of the gravity direction and the scale factor during the optimization is performed by using the following expressions:

$$\begin{aligned} R_{wg}^{\text{new}} &= R_{wg}^{\text{old}} \operatorname{Exp}(\delta\alpha_g, \delta\beta_g, 0) \\ s^{\text{new}} &= s^{\text{old}} \exp(\delta s) \end{aligned} \tag{3.39}$$

At the end of the inertial-only optimization, the camera poses and the map points are scaled with the estimated scale and rotated to align the z axis with the estimated gravity direction and the IMU preintegration is performed again as biases are updated to decrease linearization errors. The initial estimation of the visual and inertial parameters is so obtained.

3) Visual-Inertial MAP Estimation: The third step of the initialization is the refinement of the visual and inertial parameters. The refinement of these variables is performed by using a visual-inertial optimization. The visual-inertial optimization uses common biases for all keyframes and includes the prior residual used in the inertial-only step.

Visual-Inertial SLAM

In visual-inertial SLAM, the state estimation includes the body pose and velocity in the world frame and the gyroscope and accelerometer biases. The state vector is then defined as:

$$\mathcal{S}_i \doteq \{T_i, v_i, b_i^g, b_i^a\} \tag{3.40}$$

The IMU preintegration is performed between the frames i and $i+1$ to save a significant amount of computational resources since it eliminates the necessity to propagate repeatedly the IMU measurements. This process results in the rotation $\Delta R_{i,i+1}$, velocity $\Delta v_{i,i+1}$ and position $\Delta p_{i,i+1}$ preintegration terms. The inertial residual is defined as:

$$r_{\mathcal{I}_{i,i+1}} = [r_{\Delta R_{i,i+1}}, r_{\Delta v_{i,i+1}}, r_{\Delta p_{i,i+1}}] \tag{3.41}$$

where:

$$\begin{aligned} r_{\Delta R_{i,i+1}} &= \mathrm{Log}(\Delta R_{i,i+1}^T \, R_i^T \, R_{i+1}) \\ r_{\Delta v_{i,i+1}} &= R_i^T (v_{i+1} - v_i - g\Delta t_{i,i+1}) - \Delta v_{i,i+1} \\ r_{\Delta p_{i,i+1}} &= R_i^T \Big(p_j - p_i - v_i \Delta t - \frac{1}{2} g \Delta t^2 \Big) - \Delta p_{i,i+1} \end{aligned} \tag{3.42}$$

The visual residual is instead represented by the reprojection error r_{ij} between the frame i and the map point j defined as:

$$r_{ij} = u_{ij} - \Pi \left(T_{\mathrm{CB}} T_i^{-1} \otimes x_j \right) \tag{3.43}$$

where $\Pi : \mathbb{R}^3 \rightarrow \mathbb{R}^n$ is the projection function of the specific camera model, u_{ij} is the observation of point j in the frame i and T_{CB} is the rigid body transformation from the IMU to the camera frame. The visual-inertial SLAM can be considered a keyframe-based minimization problem defined as:

$$\min_{\bar{\mathcal{S}}_k, \mathcal{X}} \left(\sum_{i=1}^{k} \| r_{\mathcal{I}_{i,i+1}} \|^2_{\Sigma_{\mathcal{I}_{i,i+1}}} + \sum_{j=0}^{l-1} \sum_{i \in \mathcal{K}^j} \rho_{Hub} \Big(\| r_{ij} \|_{\Sigma_{ij}} \Big) \right) \tag{3.44}$$

where $\bar{\mathcal{S}}_k$ is the set of states associated with the set of keyframes and $\mathcal{X}$ is the set of states associated with the set of map points involved in the optimization. The Huber norm is only used for the visual residual to reduce the impact of the outliers.

Tracking and Mapping

The visual-inertial optimization requires to be adapted for efficiency in the tracking and local mapping thread. In the tracking thread, the optimization is adapted by involving only the states of the latest frames maintaining fixed map points. In the local mapping thread, the optimization is adapted by using a sliding window of keyframes in which the covisible keyframes are fixed. The simplification of the visual-inertial optimization is required by the local mapping thread to handle large maps. In addition, the local mapping thread performs an IMU scale refinement every ten seconds until a large number of keyframes is inserted in the map to obtain a robust and accurate initialization of the inertial parameters even in presence of slow motion. The scale refinement consists in an inertial-only optimization in which the scale and the gravity direction are the only parameters to compute. All the keyframes of the map and the IMU biases remain fixed in the optimization.

3.4.3 Map Merging and Loop Closing

One of the main novelties in the ORB-SLAM3 library is a new place recognition algorithm with a high recall for loop closing. The place recognition is launched every time a new keyframe is inserted to detect a loop candidate among all the keyframes in the Atlas. This approach is considered innovative for several reasons. For example, the loop closing is immediately performed only when the candidate keyframe is part of the active map. Conversely, when a multi-map data association is found, the active map and the matching map are merged before closing the loop. The innovations of the new place recognition algorithm are the reasons for the better accuracy of the ORB-SLAM3 library with respect to the previous implementations. This section provides a detailed description of the loop closing process.

Place Recognition

Every time a new keyframe is inserted in the active map, the place recognition is performed by querying the DBoW2 keyframe database for the most similar keyframes in the Atlas among the non-covisible ones. The new keyframe in the active map is denoted as K_a and each of the candidates for place recognition is denoted as K_m. The definition of a local window that includes the candidate and its covisible keyframes and the map points observed by them is performed. The candidate then undergoes a consistent geometrical verification. The geometrical verification consists in the search of a set of matches between the keypoints in the new keyframe and the keypoints in the local window of keyframes. The matches are searched by comparing the ORB descriptors of the keypoints in a search window. The computation of the transformation T_{am} that aligns the map points of the local window and those of the active keyframe is then performed by using the Horn algorithm in a RANSAC scheme. The hypotheses for the transformation are determined by using a minimal set of three matches. The matches for which the reprojection error obtained by applying the hypothesis to the map point in the new keyframe is below a certain threshold assign a positive vote to the hypothesis. The transformation T_{am} is chosen as the hypothesis with the highest number of votes. The refinement of the transformation is performed by solving a non-linear optimization problem in which all the matches obtained in a guided matching refinement are used. The goal function of the optimization is the bidirectional reprojection error. The refinement can be performed for a second time by using a smaller search window if the number of inliers after the optimization is over a threshold. Finally, the validation of the place recognition is performed in three covisible keyframes. The candidate for place recognition is accepted if there are at

least two keyframes covisible with the new keyframe in the active map in which the number of matches with the points in the local window is above a threshold. In the visual-inertial case, the validation of the candidate goes also through a gravity direction verification. The verification is performed by checking if the pitch and roll angles are below a defined threshold.

Map Merging

In case the place recognition finds a multi-map data association, the active map and the matching map are merged into a single map before closing the loop. The map merging is performed by bringing the active map into the matching map reference so that the information in the matching map can be immediately reused to prevent map duplication. The first step of the map merging is the assembly of a welding window in which the merge is performed before the correction is propagated to the rest of the map. The welding window is composed of the new keyframe and the matching keyframe along with their covisible keyframes and the map points observed by all of them. The active map and the matching map are merged in the welding window to become the new active map. The removal of duplicated points is performed by searching for the points of the active map in the matching map. For every match, only the point of the matching map is retained. The update of the covisibility and the essential graph is also performed by adding the new edges between the keyframes in the active map and those in the matching map. The optimization of the keyframes in the welding window is then performed by a local BA. The poses, velocities, and biases of the new keyframe and the matching keyframe along with those of their five last temporal keyframes are involved in the optimization. The keyframe in the matching map that is immediately outside the welding window is fixed in the optimization. The corresponding keyframe in the active map is also included in the optimization but its pose remains optimizable. The map points observed by all the keyframes involved in the optimization are also optimized. Finally, a global pose graph optimization is performed to propagate the correction from the welding window to the rest of the map.

Loop Closing

The process of loop closing is performed in a similar way to that of map merging. The only difference is that the new keyframe and the loop candidate now belong to the same map. The first step of the loop closing is the assembly of a welding window. The fusion of the duplicated points is performed adding new edges in the covisibility graph. The pose graph optimization is then run to propagate the loop correction to the rest of the map. The final step of the loop closing is a global BA that performs the pose estimation by using all the loop closure matches. This step

is not always performed in the visual-inertial case to reduce the computational burden of the process.

3.4.4 Experimental results

The experimental results obtained by running the algorithm on a laptop set up with ROS Melodic are displayed in ORB-SLAM3 Map Viewer. The right side of the figure shows the input image of the camera in a monocular setup. The left side of the figure shows the output of the algorithm that consists of the odometry data represented by the green line and the point cloud map of the environment represented by black and red sparse points. The most interesting thing is the observation of the essential graph of the whole map that is composed of the spanning tree of keyframes represented by the blue rectangles and the subset of edges of the covisibility graph with higher covisibility represented by the tangle of green lines. The loop closure is represented by the link connecting the first and the last keyframe.

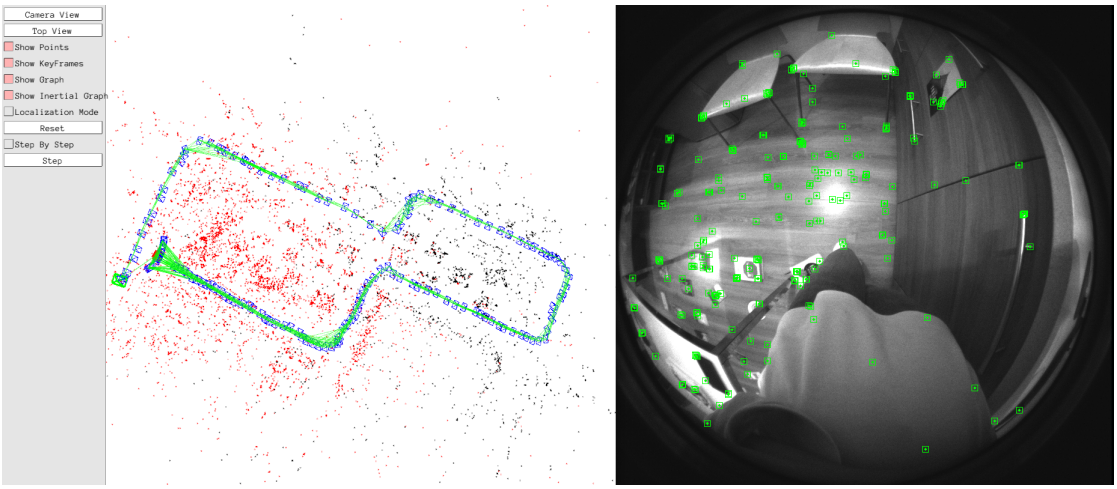

Figure 3.19: Illustration of the algorithm performance in ORB-SLAM3 Map Viewer - The output of the algorithm consists of the odometry data represented by the green line and the point cloud map of the environment represented by black and red sparse points

Chapter 4

Intel® RealSense™ T265 Tracking Camera

The Intel® RealSense™ T265 Tracking Camera is a tracking sensor that uses visual and inertial sensor fusion to perform the pose estimation of the host system. The hardware design includes two fisheye cameras (OV9282) with a 163-degree FOV, an IMU module (Bosch BMI055), and an Intel® Movidius™ Myriad™ 2.0 Vision Processing Unit (VPU). There are a lot of reasons because this sensor is considered ideal for aerial robots. First of all, the camera is 108 x 25 x 13 mm in size and weighs around 60 g. The small size and lightness of the sensor mean that this is the first device capable of performing the pose estimation completely with embedded computers. In addition, this sensor provides high performance with low latency and low power consumption. The low power consumption is a crucial characteristic because, as well as long battery life, it enables the use of small batteries thus reducing the overall size of the system. Finally, the wide field of view of the fisheye lenses allows for feature tracking even in presence of high-speed motion. The tracking camera features a highly optimized proprietary VI-SLAM algorithm running on the Intel® Movidius™ Myriad™ 2.0 VPU for robust and accurate odometry estimation. The interface between the camera and the host system is realized by the Intel® RealSense™ Software Development Kit 2.0. The RealSense™ T265 tracking camera can be jointly used with the RealSense™ D400 depth camera for advanced applications such as occupancy mapping construction and obstacle sensing and avoidance. In Table 4.1 the Intel® RealSense™ T265 Tracking Camera datasheet is presented. This chapter provides a discussion of the camera modeling and calibation and the IMU noise modeling using the Allan variance. At the end of the chapter, the results of the camera calibration and the IMU performance analysis are presented [47].

Figure 4.1: Illustration of the Intel® RealSense™ T265 Tracking Camera - The coordinate system is composed of x direction towards the right imager, y direction upwards toward the top of the device, and z direction inwards toward the back of the device

Intel® RealSense™ T265 Tracking Camera Datasheet			
Model	T265 Tracking Camera	Depth FOV	D:163
Dimensions	108 x 25 x 13 mm	Color	Monochrome
Weight	60 g	Resolution	848 x 800
Voltage	5 V	Baseline	64 mm
Current	300 mA	Video Format	8bit, 10-bit RAW
Temperature	0 - 35 °C	Shutter type	Global Shutter
Power	1.5 W	Filter type	IR Cut Filter
Interface Type	USB 3.0	IMU model	Bosch BMI055
Imagers	OV9283 Fisheye	IMU Sample Rate	200 Hz

Table 4.1: Intel® RealSense™ T265 Tracking Camera datasheet [48]

4.1 Camera Modeling and Calibration

This section provides a detailed description of the camera modeling and calibration. First of all, the simplest projection model, called the pinhole camera model, is discussed. In addition, the main distortions, called radial and tangential distortion, are presented. Finally, the camera calibration procedure used to obtain the internal and external camera parameters is described. The main goal of this dissertation is that of recovering the 3D geometry of the world from the 2D images captured by the camera.

4.1.1 Pinhole Camera Model

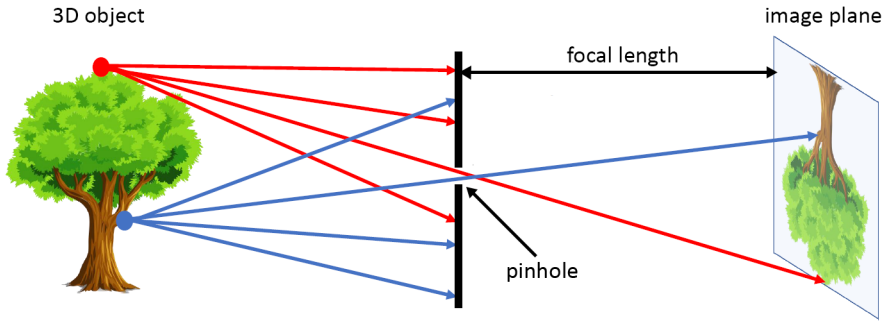

Figure 4.2: The pinhole camera model is the simplest projection model that describes the projection of a 3-D point into the image plane of a pinhole camera [49]

The pinhole camera model is the simplest projection model that describes the projection of a point of the three-dimensional space into the image plane of a pinhole camera. The model assumes that the rays of light emitted by the point pass without deflections through a pinhole placed between the scene and the image plane. Sometimes the image plane is placed in front of the pinhole at a distance equal to the focal length in order to obtain the unrotated image of the scene. In this particular case, the image plane is called the virtual image plane. The pinhole camera model uses the extrinsic and intrinsic parameters to define the transformation between the scene and the image plane.

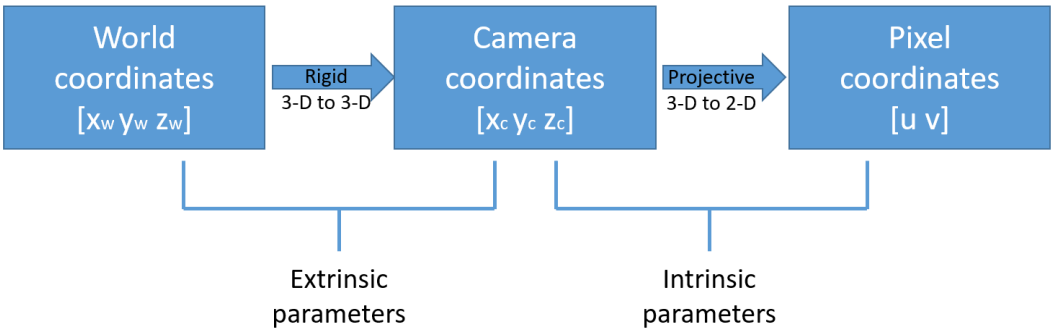

Figure 4.3: The extrinsic parameters are used to define the rigid body transformation between the 3-D world reference frame and the 3-D camera coordinate frame and the intrinsic parameters are used to define the projective transformation between the 3-D camera coordinate frame and the 2-D pixel coordinate frame.

Extrinsic parameters

The first step consists in mapping the point from the world reference frame to the camera coordinate frame using the extrinsic parameters of the camera. The extrinsic parameters are the camera position and orientation with respect to the world reference frame. These parameters are external parameters that change as the camera moves in space. Using the homogeneous coordinates, the rigid body transformation between the world reference frame and the camera coordinate frame is defined as:

$$P_c = \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} P_w \tag{4.1}$$

where the extrinsic parameters are the rotation matrix R and the translation vector t. There are different methods for obtaining the camera orientation angles around the pitch, roll and yaw axes from the rotation matrix and vice versa. The goal of the localization is that of performing the pose estimation of the drone after recovering the extrinsic parameters of the camera [50].

Intrinsic parameters

The second step consists in mapping the point from the camera reference frame to the image plane using the intrinsic parameters of the camera. The intrinsic parameters are the focal lengths in pixels f_x and f_y, the principal point coordinates c_x and c_y and the skew coefficient, s, which is non-zero if the pixel axes are not perpendicular. Most cameras have identical focal lengths and zero-skew. These parameters are intrinsic properties of the camera that don't change during the pose estimation. This means that they can be computed once for all during the calibration procedure. Using the homogeneous coordinates, the projective transformation between the camera reference frame and the image plane is defined as:

$$P' = z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} P_c = MP_c \tag{4.2}$$

The transformation matrix can be decomposed into:

$$M = \begin{bmatrix} f_x & s & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = K \begin{bmatrix} I & 0 \end{bmatrix} \tag{4.3}$$

where:

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.4}$$

is called the intrinsic matrix. It is possible to notice that the camera provides information up to a scale factor. This means that multiplying the camera coordinates by a scalar results in the same image coordinates [50].

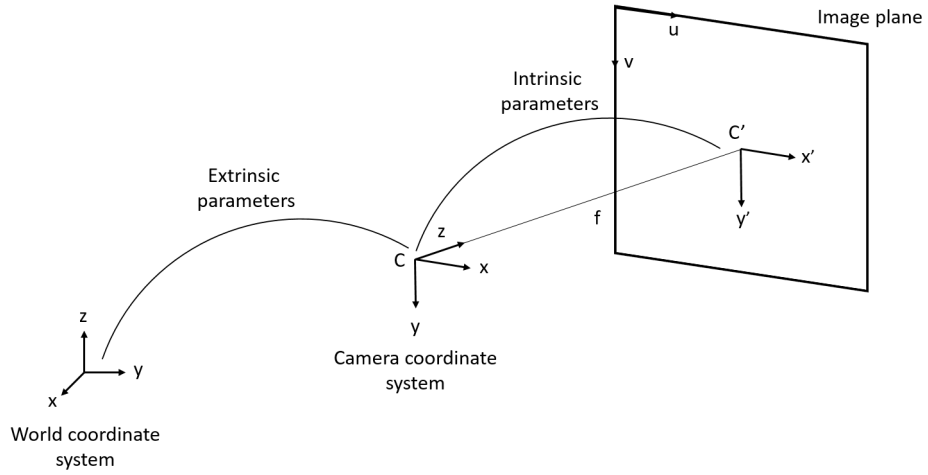

Figure 4.4: Illustration of the pinhole camera model - The extrinsic parameters define the transformation between the world coordinate system and the camera coordinate system and the intrinsic parameters define the transformation between the camera coordinate system and the pixel coordinate system

In conclusion, it can be said that the pinhole camera model uses the extrinsic and intrinsic parameters of the camera to describe the transformation between the world reference frame and the pixel coordinate frame. This transformation is defined as:

$$P' = z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = K \begin{bmatrix} R & t \end{bmatrix} P_w \tag{4.5}$$

where $P = K[R\ \ t]$ is called the projection matrix. This matrix provides the pixel coordinates up to a scale factor, as mentioned before. This problem can be solved immediately by normalizing the obtained pixel coordinates in such a manner that the final coordinate equals one [49].

4.1.2 Lens Distortions

Real cameras use lenses with defects in design and manufacturing that cause distortion in the image. Lens distortion is "an optical aberration that causes the lens a deviation from the rectilinear projection" [51]. The main types of distortion are radial and tangential distortion. The radial distortion is divided into barrel distortion and pincushion distortion depending on whether the image magnification decreases or increases as a function of the distance from the principal point. The fisheye cameras are heavily affected by barrel distortion. The tangential distortion is an asymmetrical effect that consists in non-parallelism between the lens and the image plane [49].

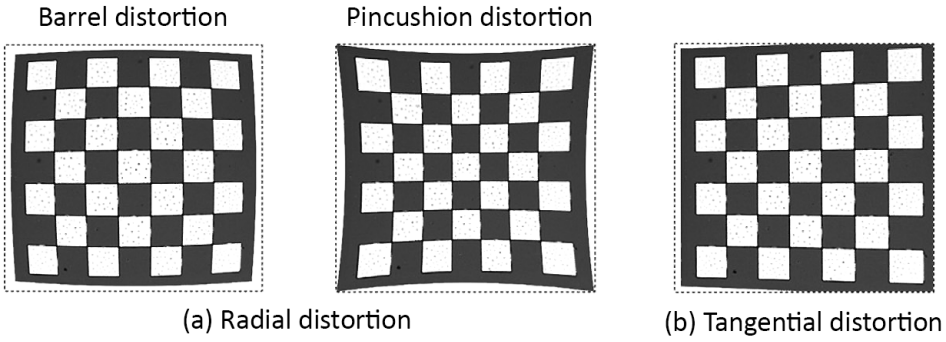

Figure 4.5: Illustration of the radial (a) and tangential (b) distortion - The radial distortion can be classified as barrel distortion when the image magnification decreases and pincushion distortion when the image magnification increases [49]

The radial and tangential distortions are modeled with the Brown–Conrady distortion model. The Brown–Conrady distortion model was first presented in 1971 by Brown in his paper "Close-Range Camera Calibration" [52]. This model uses a third order approximation for the first one and a second order approximation for the second one resulting in:

$$x_d = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 xy + p_2(r^2 + 2x) \tag{4.6}$$
$$y_d = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_2 xy + p_1(r^2 + 2y) \tag{4.7}$$

where x_d and y_d are the coordinates of the distorted point, x and y are the coordinates of the undistorted point, k_i and p_i are the radial and tangential distortion coefficients of the lens and r is the distance between the undistorted point and the image center [50].

4.1.3 Camera calibration

Camera calibration is the process of computing the intrinsic and extrinsic parameters of the camera. There are different types of calibration procedures. The calibration of the Intel RealSense T265 Tracking Camera has been performed using the checkerboard-based method. The checkerboard calibration consists in using a collection of pictures of a checkerboard pattern captured with the camera from different points of view. The checkerboard pattern contains a set of points located at the corners of the squares whose world coordinates and pixel location in the images are used to calculate the camera parameters. In the following, the checkerboard-based method is described step-by-step.

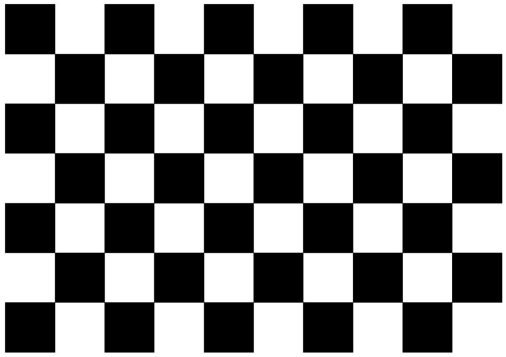

Figure 4.6: Illustration of the calibration checkerboard - The camera parameters are calculated using the world coordinates and pixel location of a set of points located at the corners of the squares in the checkerboard pattern

The first step consists in placing a checkerboard pattern on a planar surface such as a wall or a hardcover book. The 9x6 checkerboard illustrated in Figure 4.6 has been attached to a white wall in the room. The points of the checkerboard lie on the same plane so that it can be arbitrarily chosen Z=0 for every point. The (X, Y) coordinates of the points are obtained by defining the location of the corners with respect to a reference point (0,0). The procedure is easy because the points are equally spaced on the checkerboard. The coordinates obtained need to be multiplied by the square size as they are in the scale of the size of the checkerboard square. The square size of the checkerboard used for the calibration was 26 mm.

The second step consists in capturing a collection of pictures of the checkerboard under different points of view by moving either the camera or the checkerboard. Several pictures of the checkerboard were taken at different orientations by moving the camera in the room while keeping the checkerboard static on the wall. The pictures were captured under different light conditions in order to obtain a robust and accurate calibration.

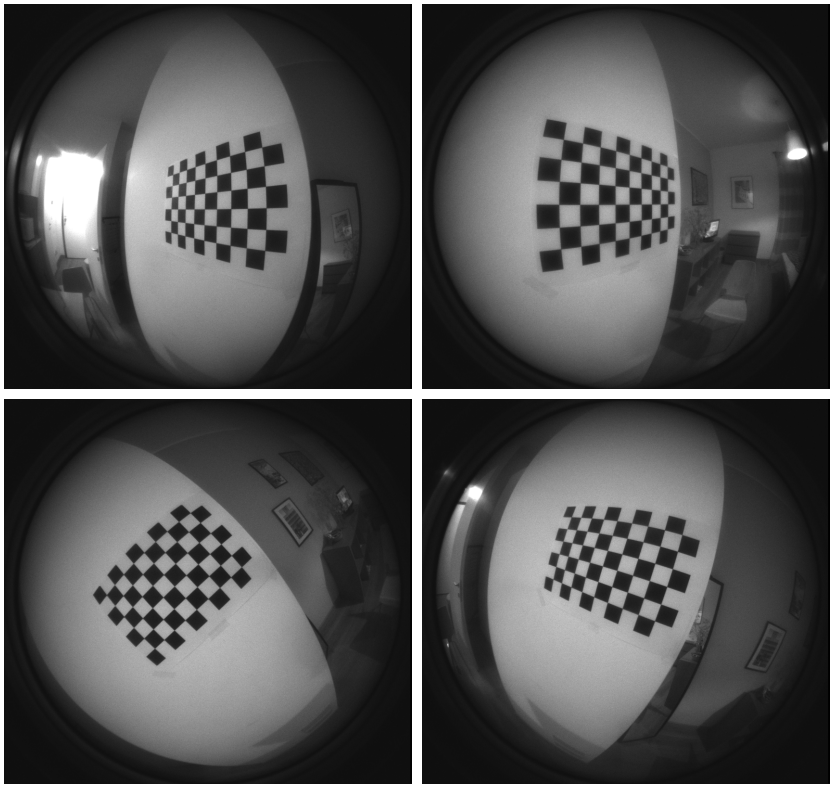

Figure 4.7: Samples of pictures used for calibration - The pictures were taken at different orientations by moving the camera in the room while keeping the checkerboard static on the wall

The third step is the process of computing the pixel location of the checkerboard corners in the images. This process is performed using the builtin function cv2.findChessboardCorners() provided by OpenCV. The input data required are a picture of the checkerboard and the pattern size. The function locates the checkerboard corners in the image after having determined whether the image is a view of the checkerboard pattern. In fact, the output data are the array of the corner locations and the retval value which is true or false depending on whether the

pattern has been found or not in the image. At this point, it is possible to perform the refinement of the chessboard corners using the function cv2.cornerSubPix() that achieves a sub-pixel level of accuracy. The input data required are the image of the checkerboard and the location of corners in the image. The algorithm calculates iteratively the best location of the points in the neighborhood of the original location. Consequently, the termination criteria for the process of corner refinement (e.g. the number of iterations or the accuracy of the location) must be specified. In conclusion, there is the possibility to draw the pattern on the images using the function cv2.drawChessboardCorners() provided by OpenCV. The input data required are the image of the checkerboard, the pattern size, and the location of corners in the image. The function draws the corners on the checkerboard as colored circles connected with lines. The pictures obtained from the third step of camera calibration are shown in Figure 4.8.

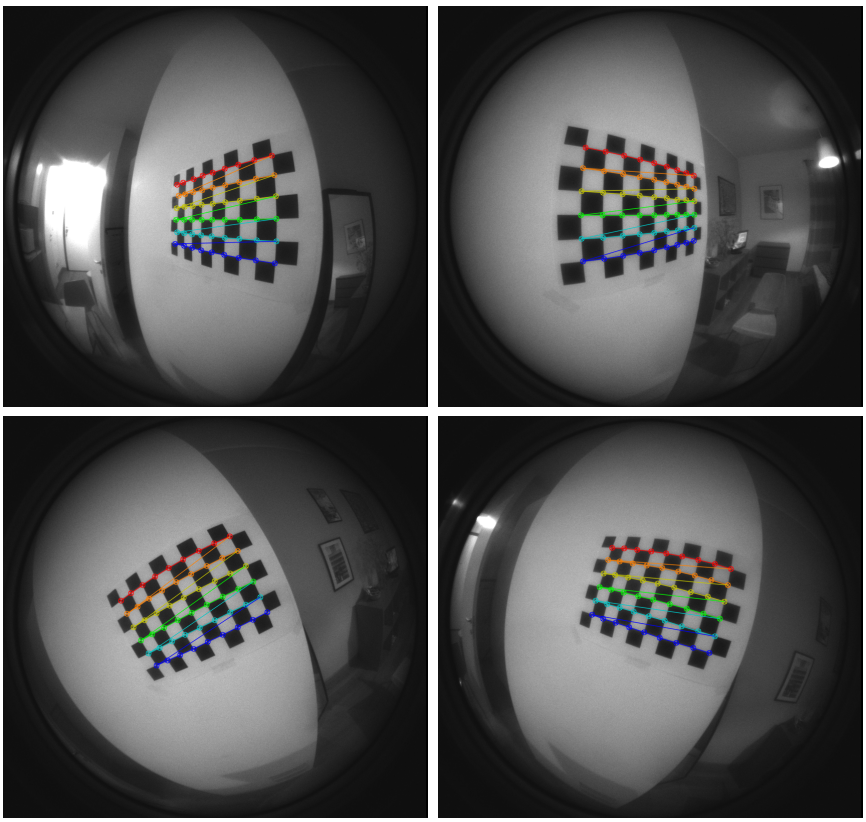

Figure 4.8: Samples of pictures obtained from the third step of camera calibration - The chessboard corners detected in the pattern are drawn on the pictures as colored circles connected with lines

The final step consists in the calibration of the camera. This process is performed by using the function cv2.calibrateCamera() provided by OpenCV. The function calculates the intrinsic and extrinsic parameters of the camera for different views of the checkerboard using the world coordinates and the pixel locations of the corners obtained in the previous steps. The implementation that is described in this section is based on the paper "A Flexible New Technique for Camera Calibration" by Zhengyou Zhang [53]. The relationship between a chessboard corner and its image correspondence is described by:

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} r_1 & r_2 & r_3 & t \end{bmatrix} \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} = K \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{4.8}$$

where the homography $H = [h_1 \;\; h_2 \;\; h_3] = K[r_1 \;\; r_2 \;\; t]$ is a 3×3 matrix defined up to a scale factor. The homography can be estimated from an image of the model plane with a technique based on the maximum likelihood criterion. Once the homography has been computed, it is possible to define the two fundamental constraints on the intrinsic parameters. The definition of the constraints is based on the observation that r_1 and r_2 are orthonormal. This results in the equations:

$$h_1^T K^{-T} K^{-1} h_2 = 0 \tag{4.9}$$

$$h_1^T K^{-T} K^{-1} h_1 = h_2^T K^{-T} K^{-1} h_2 \tag{4.10}$$

That said, the closed-form solution to the problem of camera calibration is now presented. Define:

$$B = K^{-T} K^{-1} = \begin{bmatrix} \frac{1}{f_x^2} & -\frac{s}{f_x^2 f_y} & \frac{c_y s - c_x f_y}{f_x^2 f_y} \\ -\frac{s}{f_x^2 f_y} & \frac{s^2}{f_x^2 f_y^2} + \frac{1}{f_y^2} & -\frac{s(c_y s - c_x f_y)}{f_x^2 f_y^2} - \frac{c_y}{f_y^2} \\ \frac{c_y s - c_x f_y}{f_x^2 f_y} & -\frac{s(c_y s - c_x f_y)}{f_x^2 f_y^2} - \frac{c_y}{f_y^2} & \frac{(c_y s - c_x f_y)^2}{f_x^2 f_y^2} + \frac{c_y^2}{f_y^2} + 1 \end{bmatrix} \tag{4.11}$$

The symmetric matrix B is characterized by the vector:

$$b = \begin{bmatrix} B_{11}, B_{12}, B_{22}, B_{13}, B_{23}, B_{33} \end{bmatrix}^T \tag{4.12}$$

In this case, the following equation applies:

$$h_i^T B h_j = v_{ij}^T b \tag{4.13}$$

with

$$v_{ij} = \left[h_{i1}h_{j1}, h_{i1}h_{j2} + h_{i2}h_{j1}, h_{i2}h_{j2}, h_{i3}h_{j1} + h_{i1}h_{j3}, h_{i3}h_{j12} + h_{i2}h_{j3}, h_{i3}h_{j3}\right]^T \tag{4.14}$$

Based on the above, the two fundamental constraints on the intrinsic parameters can be rewritten as two homogeneous equations in b:

$$\begin{bmatrix} v_{12}^T \\ (v_{11} - v22)^T \end{bmatrix} b = 0 \tag{4.15}$$

At this point, assuming that n is the number of images observed, it is possible to stack the n above equations obtaining the following system:

$$Vb = 0 \tag{4.16}$$

where V is a $2n \times 6$ matrix. The solution to the system of equations is the eigenvector of $V^T V$ associated with the smallest eigenvalue. The estimate of the vector b is used to compute the intrinsic parameters of the camera as follows:

$$c_y = B_{12}B_{13} - B_{11}B_{23}/B_{11}B_{22} - B_{12}^2 \tag{4.17}$$

$$\lambda = B_{33} - B_{13}^2 + c_y(B_{12}B_{13} - B_{11}B_{23})/B_{11} \tag{4.18}$$

$$f_x = \sqrt{\lambda/B_{11}} \tag{4.19}$$

$$f_y = \sqrt{\lambda B_{11}/B_{11}B_{22} - B_{12}^2} \tag{4.20}$$

$$s = -B_{12}f_x^2 f_y/\lambda \tag{4.21}$$

$$c_x = sc_y/f_y - B_{13}f_x^2/\lambda \tag{4.22}$$

Finally, the extrinsic parameters are easily calculated for each image from the intrinsic matrix K:

$$r_1 = \lambda K^{-1} h_1 \tag{4.23}$$
$$r_2 = \lambda K^{-1} h_2 \tag{4.24}$$
$$r_3 = r_1 \times r_2 \tag{4.25}$$
$$t = \lambda K^{-1} h_3 \tag{4.26}$$

where $\lambda = 1/\|K^{-1}h_1\| = 1/\|K^{-1}h_2\|$. The results obtained from the camera calibration are presented in Table 4.2.

f_x	f_y	c_x	c_y
276.15397757	276.08295983	417.72146588	406.29712124

Table 4.2: Intel® RealSense™ T265 Tracking Camera calibration results

4.2 IMU Allan Variance Analysis

The Allan variance is a time-domain analysis technique that was first presented in the 1960s by the physicist David W. Allan to study the frequency stability of precision oscillators. This technique can also be used to determine the characteristics of the noise processes that affect the inertial sensor measurements. This section provides a detailed description of the inertial sensor noise modeling and analysis using the Allan variance technique based on the paper "Analysis and Modeling of Inertial Sensors Using Allan Variance" by El-Sheimy et al. [54].

4.2.1 Allan Variance

Assume that the inertial sensor provides N samples of data with a sampling time t_0. The sequence of data can be divided into clusters of n consecutive data points with $n < (N-1)/2$. The data points contained in each cluster can be averaged over the length of the cluster. The Allan variance is defined as the two-sample variance of the data cluster averages:

$$\sigma^2(T) = \frac{1}{2T^2(N-2n)} \sum_{k=1}^{N-2n} [\overline{\Omega}_{next}(T) - \overline{\Omega}_k(T)]^2 \tag{4.27}$$

where $\overline{\Omega}_k(T)$ and $\overline{\Omega}_{next}(T)$ are the cluster averages of two consecutive clusters defined as:

$$\overline{\Omega}_k(T) = \frac{\theta_{k+n} - \theta_k}{T} \tag{4.28}$$

$$\overline{\Omega}_{next}(T) = \frac{\theta_{k+2n} - \theta_{k+n}}{T} \tag{4.29}$$

The noise parameters of the inertial sensor are determined by extrapolation from the Allan standard deviation plot that is the log-log plot of the square root of the Allan variance as a function of the time T.

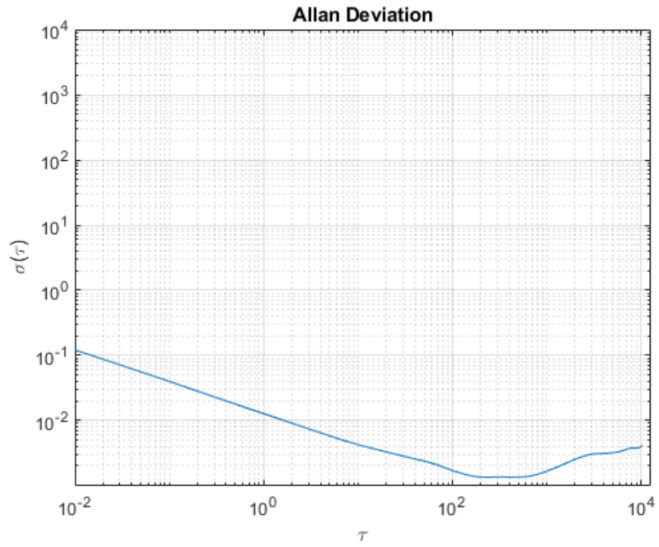

Figure 4.9: The Allan standard deviation plot is the log-log plot of the square root of the Allan variance as a function of the time T

4.2.2 Representation of Noise Terms

The different types of random processes are identified in different areas of the plot by using the relationship between the Allan variance and the two-sided power spectral density (PSD) of the random process defined as:

$$\sigma^2(T) = 4 \int_0^\infty S_\Omega(f) \, \frac{\sin^4(\pi f T)}{(\pi f T)^2} \, df \tag{4.30}$$

Angle (Velocity) Random Walk

The thermo-mechanical noise fluctuating at a higher rate than the sampling rate of the sensor results in a white noise sequence that characterize the gyroscope angle (or accelerometer velocity) random walk. The power spectral density (PSD) associated with this noise is:

$$S_\Omega(f) = N^2 \tag{4.31}$$

where N is the angle (or velocity) random walk coefficient. The substitution of this expression in the original equation results in:

$$\sigma^2(T) = \frac{N^2}{T} \tag{4.32}$$

This equation is a line with a slope of -1/2 on the Allan standard deviation plot. The value of the angle (or velocity) random walk coefficient can be determined by reading the slope line at T=1.

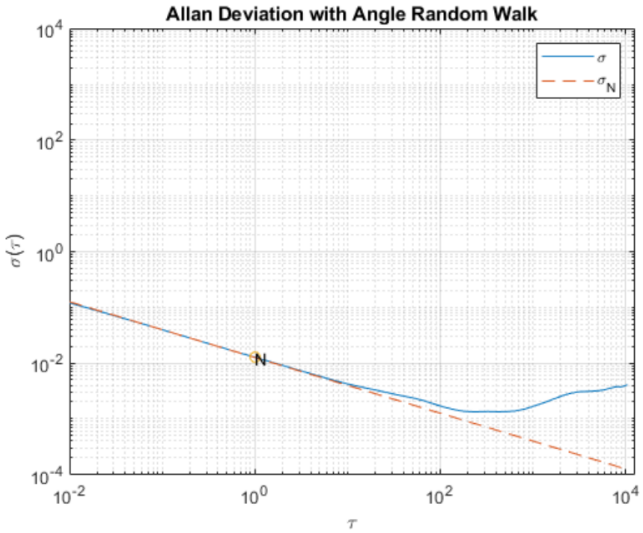

Figure 4.10: The value of the angle (or velocity) random walk coefficient can be determined by reading the line with slope of -1/2 at T=1 [55]

Bias instability

The flicker noise that arises in electronic components causes bias fluctuations that are appreciable at low frequencies. The power spectral density (PSD) associated with this noise is:

$$S_\Omega(f) = \begin{cases} \left(\frac{B^2}{2\pi}\right)\frac{1}{f} & f \leq f_0 \\ 0 & f > f_0 \end{cases} \tag{4.33}$$

where B is the bias instability coefficient and f_0 is the cutoff frequency. The substitution of this expression in the original equation results in:

$$\sigma^2(T) = \frac{2B^2}{\pi}\left[\ln 2 - \frac{\sin^3 x}{2x^2}(\sin x + 4x\cos x) + C_i(2x) - C_i(4x)\right] \tag{4.34}$$

where $x = \pi f_0 T$ and C_i is the cosine-integral function. This equation is a line with a slope of 0 on the Allan standard deviation plot. The value of the bias instability coefficient can be determined by reading the value at the intersection of the line with slope 0 and the Allan standard deviation curve. A scale factor of 0.664 must be considered.

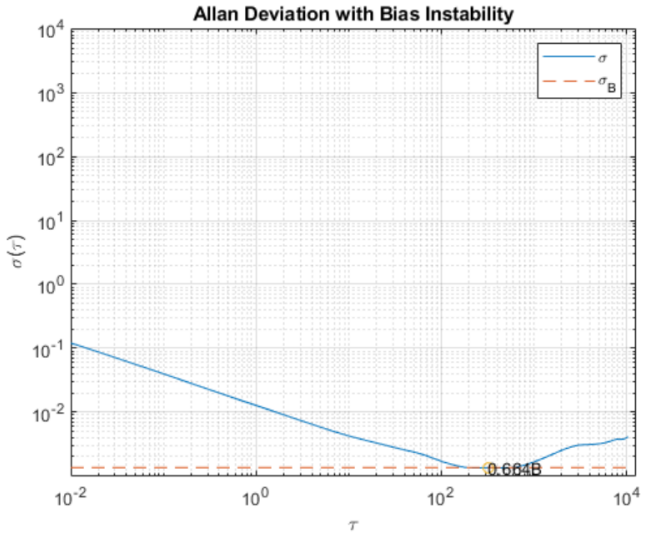

Figure 4.11: The value of the bias instability coefficient can be determined by reading the minimum value of the Allan standard deviation curve with a scale factor of 0.664 [55]

Rate random walk

The rate random walk is a random process of uncertain origin characterized by the red noise spectrum of the sensor output. The power spectral density (PSD) associated with this process is:

$$S_\Omega(f) = \left(\frac{K}{2\pi}\right)^2 \frac{1}{f^2} \tag{4.35}$$

where K is the rate random walk coefficient. The substitution of this expression in the original equation results in:

$$\sigma^2(T) = \frac{K^2 T}{3} \tag{4.36}$$

This equation is a line with a slope of +1/2 on the Allan standard deviation plot. The value of the rate random walk coefficient can be determined by reading the slope line at T=3.

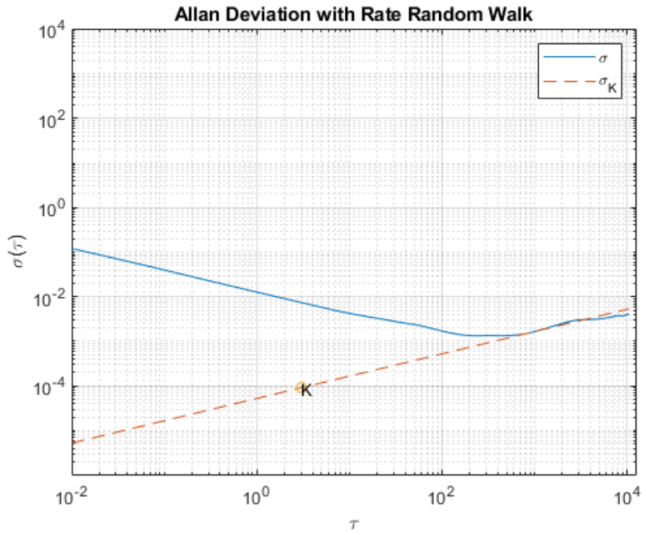

Figure 4.12: The value of the rate random walk coefficient can be determined by reading the slope line at T=3 [55]

The Allan standard deviation plot with the noise parameters obtained with the Allan variance analysis is illustrated in the following figure:

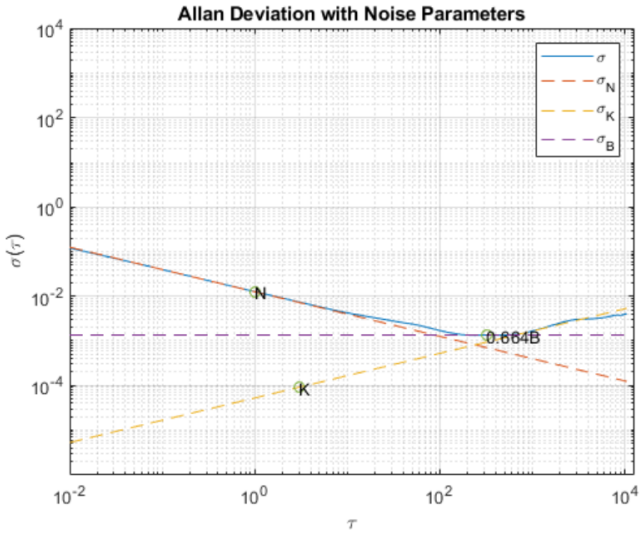

Figure 4.13: Illustration of the Allan standard deviation plot with the noise parameters obtained with the Allan variance analysis [55]

4.2.3 Noise Analysis Results

The IMU performance of the Intel RealSense T265 Tracking Camera has been analyzed with the ROS package tool imu_utils based on the Allan variance technique. The results of the Allan variance analysis are presented in Table 4.3.

gyro_noise	gyro_walk	acc_noise	acc_walk
1.8491e-03	2.5482e-05	1.09387e-02	5.8973e-04

Table 4.3: Intel® RealSense™ T265 Tracking Camera IMU parameters

Chapter 5

Results and Discussion

This chapter starts with a brief description of the methodology used for data collection and analysis. After the data collection and analysis overview, in which the testing environment is described in detail, the results of the experimentation are presented and discussed. The comparison between the different visual-inertial odometry state of the art algorithms is introduced. The algorithm that shows the best performance in terms of accuracy is then tested in both Mono+IMU and Stereo+IMU modes. Finally, the experimentation ends by testing the algorithm on the EuRoC MAV MH 01 dataset to verify the results obtained in a realistic situation.

5.1 Data collection and analysis

The experimental phase was performed in a home indoor environment with artificial lighting using an Intel T265 Tracking Camera. The camera was mounted on an HP Envy 15 Notebook PC in which Ubuntu 18.04 operating system with ROS Melodic was installed. The testing environment was a one-room apartment in which a red line, representing the ground truth reference, was drawn on the floor. In the room, a large number of furniture and objects were placed on the floor and different markers were fixed on the walls to help the camera not getting lost in areas poor of features. An illustration of the testing environment is provided in Figure 5.1. The methodology used for data collection and analysis is now briefly described. First of all, the ground truth reference was measured manually with a measuring tape. Subsequently, the testing was performed. The test consisted in walking along the ground truth path holding the laptop with the camera looking downward. The data were recorded once in a rosbag file that was then used to run the different visual-inertial odometry algorithms offline. For every algorithm, the

odometry data were saved in a text file that was then imported in MATLAB for data processing. The analysis of data was performed by computing the position error with respect to the ground truth reference.

Figure 5.1: The testing environment was a room in which a large number of furniture and objects were placed on the floor and different markers were fixed on the walls to help the camera not getting lost in areas poor of features.

5.2 Comparison of the algorithms' performances

The goal of the experimentation is to provide a comparison between different visual-inertial odometry state of the art algorithms to assess which one is the best solution in terms of accuracy for navigation in GPS-denied environments. The following algorithms were tested:

- Intel T265 Proprietary Algorithm
- SVO
- VINS-Fusion
- ORB-SLAM2
- ORB-SLAM3

The comparison of the algorithms with respect to the ground truth reference is presented in Figure 5.2.

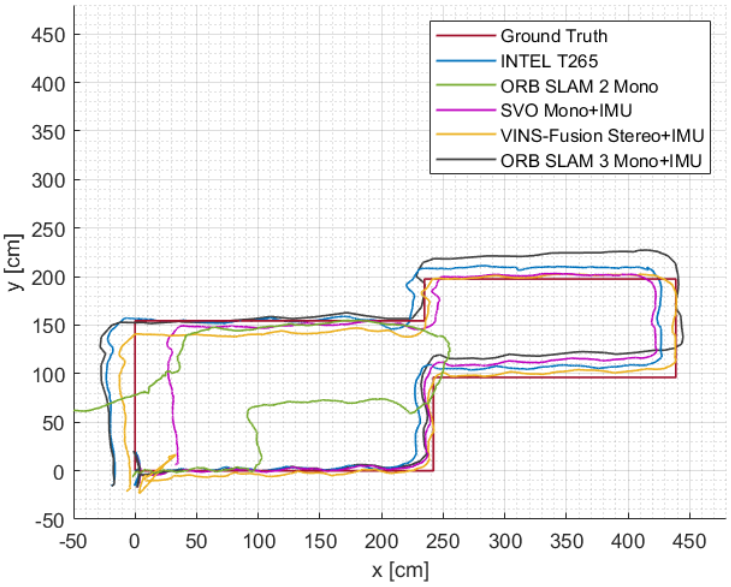

Figure 5.2: Comparison of the state of the art VIO algorithms - The odometry data of the different algorithms were compared to each other with respect to the ground truth reference

The performance of the algorithms was evaluated by computing the position error with respect to the ground truth reference. In this particular case, the points of the ground truth path were not known except for the corners because the measurements had been performed manually with a measuring tape. For this reason, the process followed for the computation of the position error is now briefly described. First of all, for each point of the trajectory, the code looked for the closest corner of the ground truth path. Secondly, the errors along the x and y axes were computed by subtraction. Finally, the position error was chosen as the minimum value between the obtained errors. This process, partially illustrated in Figure 5.3, is described in detail in the MATLAB code presented on the following page. The choice of the best algorithm was made by considering the mean error and the standard deviation of the odometry data with respect to the ground truth reference. The results of the error analysis are presented in Table 5.1.

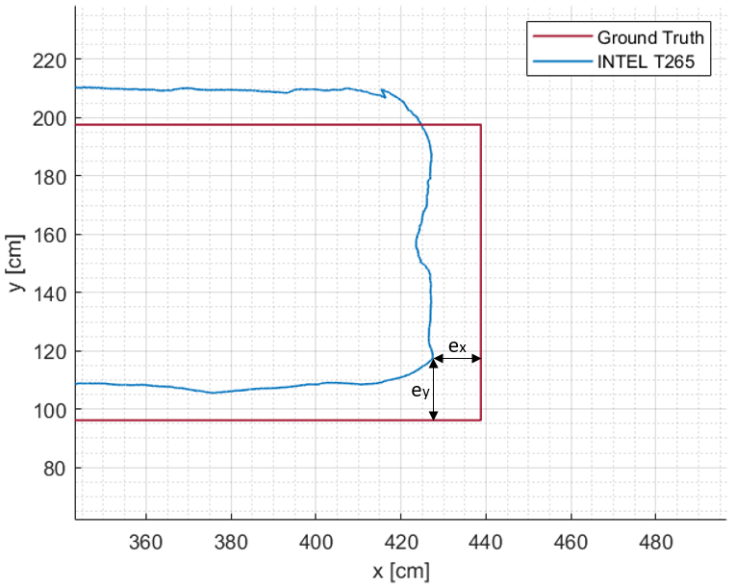

Figure 5.3: Position error estimation - The plot shows the errors along the x and y axes of a given point of the trajectory with respect to the ground truth reference.

```
%% Error analysis of the state of the art VIO algorithms

figure(2)
sgtitle('Error analysis of the state of the art VIO
    algorithms')
subplot(1,4,1)
for i=1:100:length(t1)-1
    err1(i,:) = [min([abs(yt1(i)-b) abs(xt1(i)-a)]) min([abs
    (yt1(i+1)-b) abs(xt1(i+1)-a)])];
    plot([t1(i) t1(i+1)], [min([abs(yt1(i)-b) abs(xt1(i)-a)
    ]) min([abs(yt1(i+1)-b) abs(xt1(i+1)-a)])],'ko','
    MarkerSize',5);
    xlabel('time [s]')
    ylabel('|E| [cm]')
    xlim([0 60])
    ylim([0 35])
    title('INTEL T265')
    hold on
    pause(0.02)
end
```

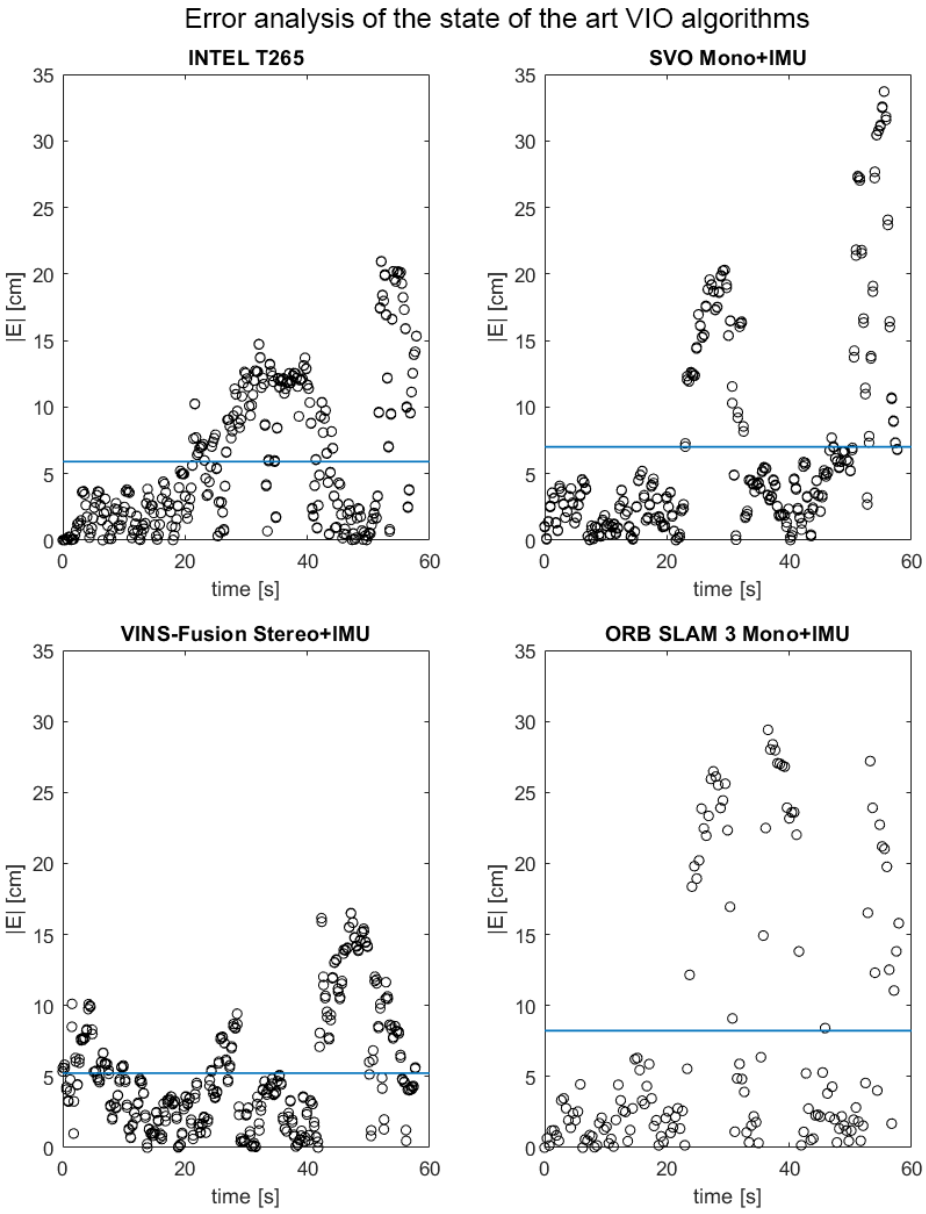

Figure 5.4: Error analysis of the state of the art VIO algorithms - The choice of the best algorithm was made by considering the mean error and the standard deviation of the odometry data with respect to the ground truth reference.

	Mean Error [cm]	STD [cm]
Intel T265	5.9076	5.4471
SVO	7.0145	7.8013
VINS-Fusion	5.2388	3.6595
ORB SLAM 3	8.2290	9.6044

Table 5.1: Error analysis results of the different VIO algorithms

The best performance in terms of accuracy is provided by VINS-Fusion which presents the smallest average error and standard deviation with respect to the ground truth reference. The performances of Intel T265 and SVO provide also reasonable results so these algorithms can be considered in case VINS-Fusion does not satisfy the computational constraints of the aerial platform. The worst performance is provided by ORB SLAM 2 that has shown to easily get lost despite the obstacles and markers placed around the room.

5.3 The choice of VINS-Fusion

A comparison of the performances of VINS-Fusion in Mono+IMU and Stereo+IMU modes was performed with the aim of verifying the impact of the advantages of a stereo camera on the accuracy of the performance. The results of the comparison are presented in Figure 5.4

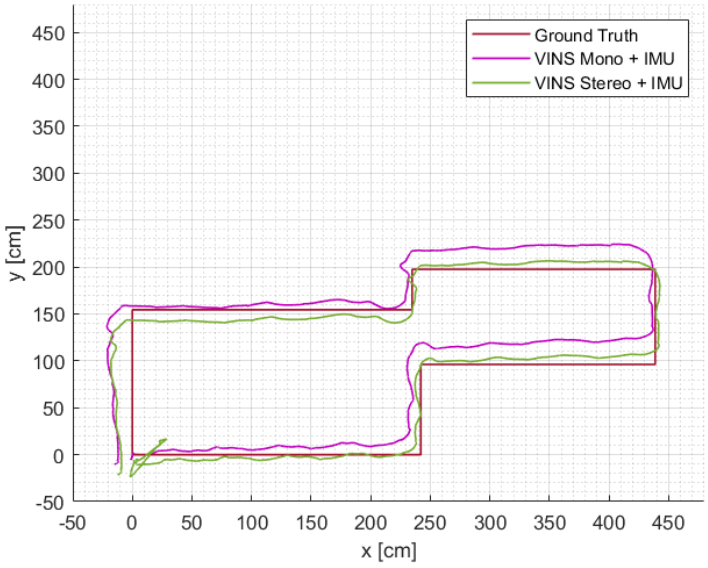

Figure 5.5: Comparison of the performances of VINS-Fusion in Mono+IMU and Stereo+IMU modes

The performances of the algorithm in Mono+IMU and Stereo+IMU modes were evaluated by computing the position error with respect to the ground truth reference. The choice of the best performance was made by considering the mean error and the standard deviation of the odometry data with respect to the ground truth reference. The error analysis of the algorithm's performances is illustrated in Figure 5.5.

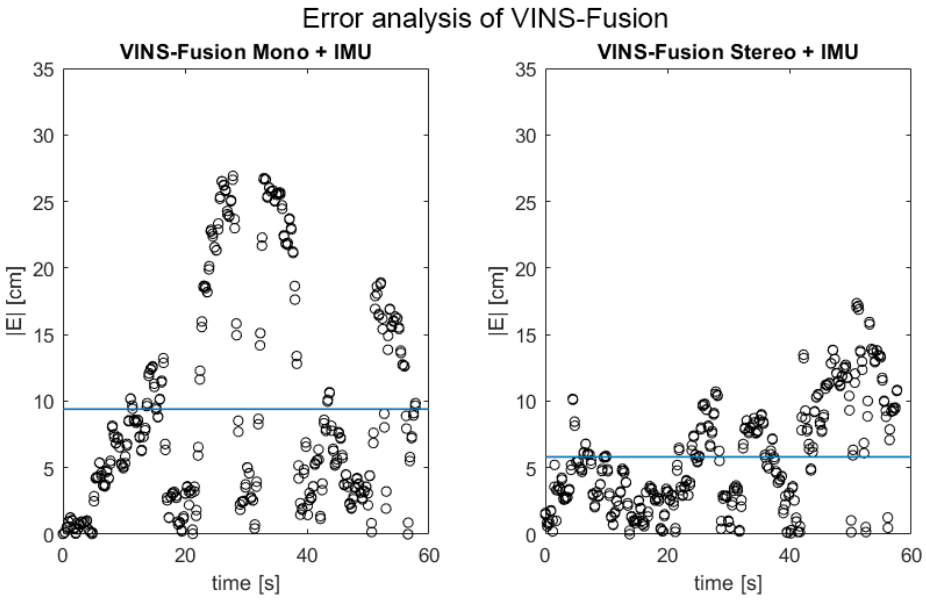

Figure 5.6: Error analysis of VINS-Fusion in Mono+IMU and Stereo+IMU modes - The choice of the best mode was made by considering the mean error and the standard deviation of the odometry data with respect to the ground truth reference.

The results of the error analysis in terms of mean error and standard deviation are summarised in Table 5.2.

VINS-Fusion	Mean Error [cm]	STD [cm]
Mono + IMU	9.3987	8.1544
Stereo + IMU	5.2388	3.6595

Table 5.2: Error analysis results of VINS-Fusion in Mono+IMU and Stereo+IMU modes

The algorithm has proven to perform best in Stereo+IMU mode. In fact, the performance of the algorithm in Stereo+IMU mode presents a mean error that is almost halved compared to that of the performance of the algorithm in Mono+IMU

mode. This means that the advantages of using a stereo camera have a great impact on the accuracy of the performance. There are different advantages of using a stereo camera. First of all, unlike monocular cameras in which the features are triangulated from time separated frames, in stereo cameras, the features are triangulated from a stereo pair in a single time step. For this reason, the stereo performance presents less drift than the monocular performance in case of small motions. Secondly, in the stereo case, the motion is estimated by observing only two time separated frames. Contrarily, in the monocular case, the motion needs to be computed from at least three different frames. Finally, the stereo camera is not affected by the scaling problem computing the transformation directly in the absolute scale. The scale ambiguity problem is one of the main problems of a monocular camera. In fact, in monocular vision, the scale of the transformation between the first two frames is unknown. Therefore, setting the scale to an arbitrary value means that the scale of the following transformations will be relative to the initial one. The scale ambiguity problem can be solved by using information from other sensors such as an IMU. The inertial sensor provides metric measurements. This means that the absolute scale can be recovered by simply integrating the measurements of the inertial sensor.

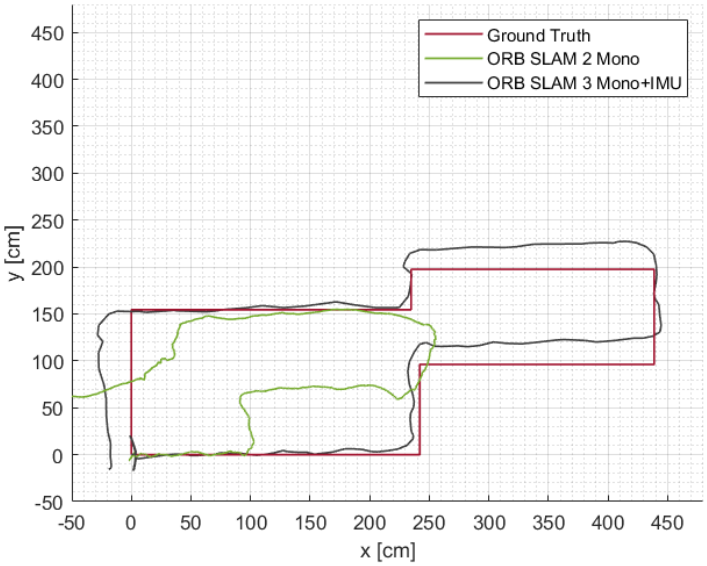

Figure 5.7: Comparison of ORB SLAM 2 Mono and ORB SLAM 3 Mono+IMU - The integration of the IMU in the implementation solves the scale ambiguity problem providing good results in terms of accuracy

5.4 Results validation on the EuRoC datasets

The experimentation ended by testing the algorithm on the EuRoC MAV datasets with the purpose of verifying the results obtained in a realistic situation. The EuRoC MAV datasets, that were recorded in the context of the European Robotics Challenge (EuRoC), were first published by Burri et al. in their paper "The EuRoC Micro Aerial Vehicle datasets" [56]. These datasets include eleven visual-inertial sequences captured on-board an AscTec Firefly hex-rotor helicopter by a visual-inertial sensor mounted in a front-down looking position. The visual-inertial sensor is composed of a stereo camera capturing monochrome images at 20 Hz and an IMU providing angular velocity and linear acceleration measurements at 200 Hz. There are two different types of datasets. The first type of dataset includes five sequences recorded in a machine hall at ETH Zurich in which the ground truth reference was measured by a Leica MS50 laser tracker. The second type of dataset includes six sequences recorded in a Vicon room in which the ground truth reference was measured by a Vicon motion capture system. The number of the sequences indicates the level of difficulty in terms of flight dynamics and illumination conditions. The sequence used for the validation of the results obtained is Machine Hall 01 which allows testing the algorithm in a realistic industrial scenario. The algorithm was tested on the dataset in both Mono+IMU and Stereo+IMU modalities. The results of the comparison are presented in Figure 5.8.

Figure 5.8: Illustration of the machine hall at ETH Zurich used to record the Machine Hall sequences of the EuRoC MAV datasets

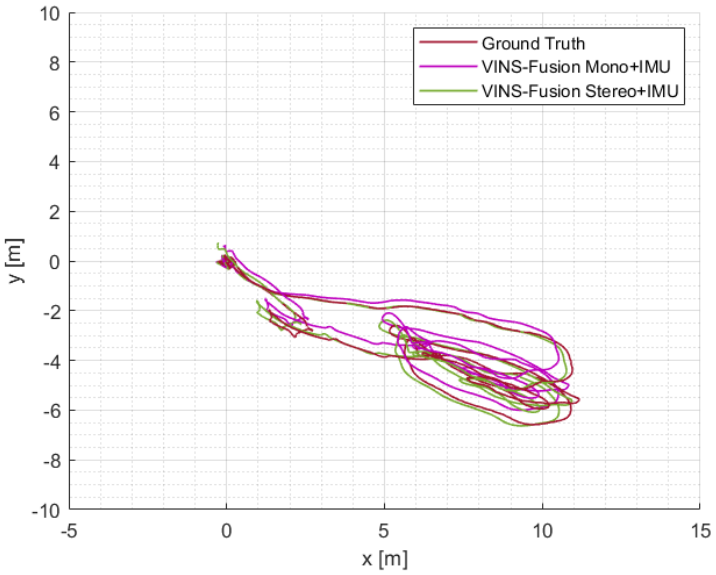

Figure 5.9: Comparison of the performances of VINS-Fusion in Mono+IMU and Stereo+IMU modes on the EuRoC MH 01 dataset

The performances of the visual-inertial algorithm in Mono+IMU and Stereo+IMU modes were evaluated by computing the mean square error with respect to the ground truth reference. The mean square error is defined as:

$$MSE = \sqrt{e_x^2 + e_y^2 + e_z^2} \tag{5.1}$$

where:

$$e_x = x - x_g \tag{5.2}$$

$$e_y = y - y_g \tag{5.3}$$

$$e_z = z - z_g \tag{5.4}$$

The choice of the best modality was made by considering the mean error and the standard deviation of the performances. The error analysis of the algorithm performances is illustrated in Figure 5.9.

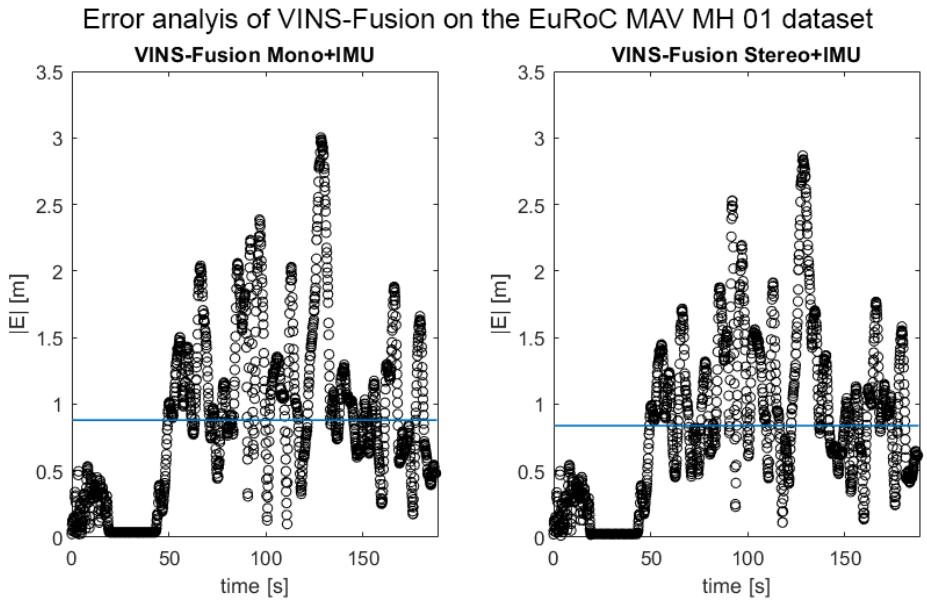

Figure 5.10: Error analysis of VINS-Fusion on the EuRoC MAV MH 01 dataset in Mono+IMU and Stereo+IMU modes - The choice of the best performance was made by considering the mean error and the standard deviation of the odometry data with respect to the ground truth reference.

The results of the error analysis in terms of mean error and standard deviation are summarised in Table 5.3.

VINS-Fusion	Mean Error [cm]	STD [cm]
Mono + IMU	88.0425	64.1824
Stereo + IMU	83.8275	61.1559

Table 5.3: Error analysis results of VINS-Fusion on the EuRoC MAV MH 01 dataset

The findings of the experiments have been validated on the EuRoC MAV MH 01 dataset. The best performance of the algorithm happens in Stereo+IMU mode providing smaller values of mean error and standard deviation. Besides, the error analysis shows that VINS-Fusion Stereo+IMU provides also a smaller maximum error compared to VINS-Fusion Mono+IMU. Nevertheless, unlike the previous tests, in this case, the difference between the two modalities is not as clear. This mismatch depends on several factors such as the number of features in the environment and the type of inertial sensor used to collect data.

Chapter 6

Conclusions and Future Developments

In the previous chapter, the comparison of the performances of different visual-inertial odometry algorithms showed that the best algorithm in terms of accuracy is VINS-Fusion. The performances of the algorithms were evaluated when running on a laptop with high computational resources. Unfortunately, the power and payload constraints of flying robots do not allow the deployment of a laptop onboard. For this reason, the evaluation of the algorithms requires to be performed on a hardware platform that is typical of a flying robot. This chapter provides the results of the research conducted by Delmerico and Scaramuzza in their paper "A Benchmark Comparison of Monocular Visual-Inertial Odometry Algorithms for Flying Robots" [57]. In this research, the most promising open-source visual-inertial odometry algorithms were tested on the EuRoC Micro Aerial Vehicle datasets when running on different hardware configurations that are typical of aerial platforms. The algorithms of our interest are VINS-Fusion and SVO that showed the best performances in terms of accuracy among the others. The hardware platform considered is the Up Board. The Up Board is a 64-bit single-board embedded computer with a small form factor (8.5 x 5.6 cm in size and 78 g in weight) and low power requirements (12 W) that includes a quad-core Intel Atom x5-Z8350 CPU running at 1.44 GHz. The results show the performances of VINS-Fusion and SVO on a Up Board in terms of CPU load, memory usage, and per-frame processing time while performing the state estimation. In the box-and-whisker plots illustrated in Figure 6.1 the whiskers represent the upper and lower quartiles and the box represents the two quartiles in the middle. The results regarding the performances of the algorithms on the laptop are included as a reference.

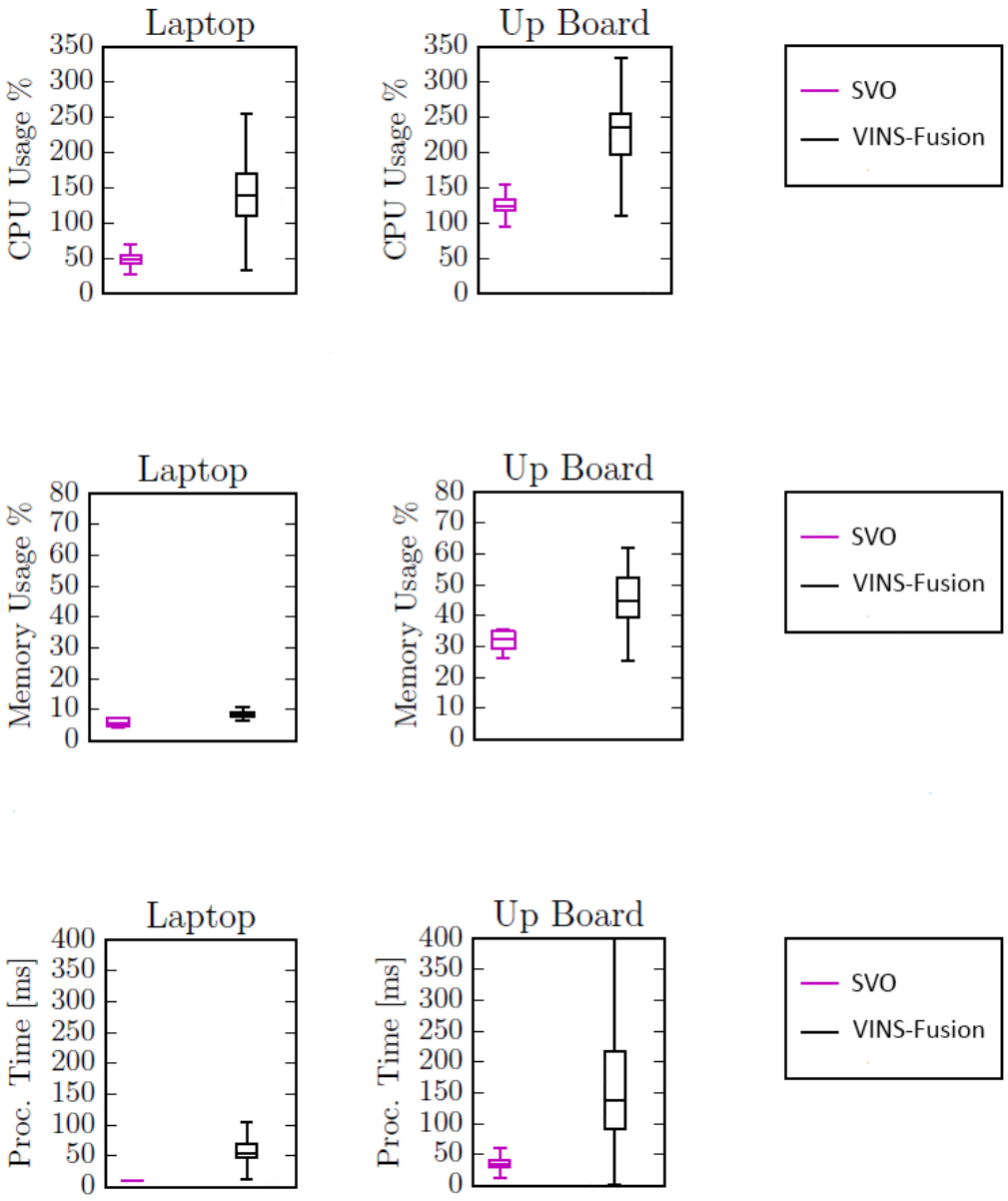

Figure 6.1: Computational requirements of the algorithms in terms of CPU load, memory usage and per-frame processing time while running on a laptop and on a Up Board [57]

The results of the research are also presented in terms of a trade-off between accuracy and computational efficiency. In the scatter plots illustrated in Figure 6.2 the computational requirements are plotted as a function of the Root Mean Square Error (RMSE). Once again the results concerning the laptop are included as a reference.

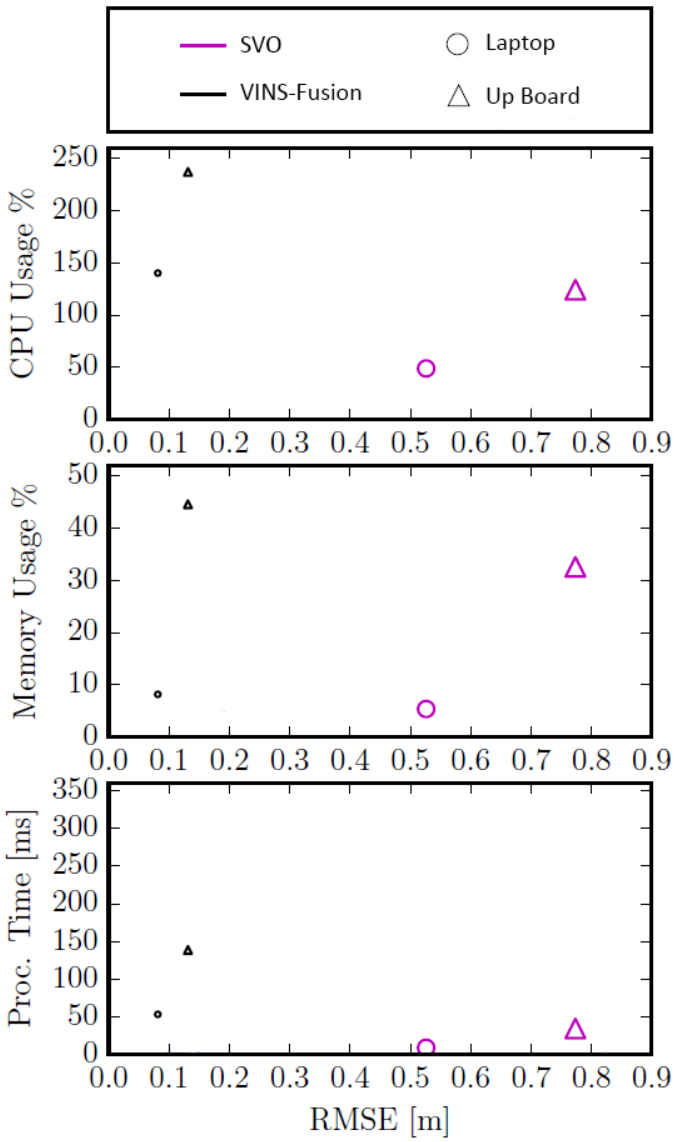

Figure 6.2: Computational requirements in terms of CPU load, memory usage and per-frame processing time plotted as a function of the Root Mean Square Error (RMSE) [57]

There are some conclusions that can be drawn from the results concerning the choice of the visual-inertial odometry algorithm. The performance of VINS-Fusion is the best one in terms of accuracy and robustness at the cost of a high computational burden. Consequently, the usage of this algorithm onboard for state estimation needs a preliminary verification on how many computational resources would remain for the other tasks of autonomous navigation. The performance of SVO can be considered a trade-off between accuracy and computational efficiency. For this reason, SVO can be taken into account in case VINS-Fusion does not satisfy the computational constraints of the platform. A future research direction could be testing VINS-Fusion onboard the flying robot used by the Polytechnic University of Turin in the Leonardo Drone Contest to verify if the remaining computational resources are sufficient for the other activities of autonomous navigation.

Bibliography

[1] K. Vyas. *A Brief History of Drones: The Remote Controlled Unmanned Aerial Vehicles*. 2020. URL: https://interestingengineering.com/a-brief-history-of-drones-the-remote-controlled-unmanned-aerial-vehicles-uavs (cit. on pp. 1–4).

[2] N. Budanovic. *The Early Days Of Drones – Unmanned Aircraft From World War One And World War Two*. 2017. URL: https://www.warhistoryonline.com/military-vehicle-news/short-history-drones-part-1.html (cit. on pp. 1, 2).

[3] D. Daly. *A Not-So-Short History of Unmanned Aerial Vehicles*. 2020. URL: https://consortiq.com/short-history-unmanned-aerial-vehicles-uavs (cit. on p. 2).

[4] J. Alkobi. *The Evolution of Drones: From Military to Hobby Commercial*. 2019. URL: https://percepto.co/the-evolution-of-drones-from-military-to-hobby-commercial/ (cit. on pp. 3, 4).

[5] M. Rouse. *The history of drones*. 2019. URL: https://internetofthingsagenda.techtarget.com/definition/drone (cit. on p. 4).

[6] J. Shahmoradi, E. Talebi, P. Roghanchi, and M. Hassanalian. «A Comprehensive Review of Applications of Drone Technology in the Mining Industry». In: *Drones* 4.3 (July 2020) (cit. on p. 5).

[7] V. Becerra. «Autonomous Control of Unmanned Aerial Vehicles». In: *Electronics* 8 (Apr. 2019), p. 452 (cit. on p. 5).

[8] L. Galtarossa, L.F. Navilli, and M. Chiaberge. «Visual-Inertial Indoor Navigation Systems and Algorithms for UAV Inspection Vehicles». In: *Industrial Robotics - New Paradigms*. Ed. by Antoni Grau and Zhuping Wang. IntechOpen, 2020. Chap. 9 (cit. on pp. 5, 6).

[9] G. Balamurugan, J. Valarmathi, and V.P.S. Naidu. «Survey on UAV Navigation in GPS Denied Environments». In: *2016 International Conference on Signal Processing, Communication, Power and Embedded System*. Paralakhemundi, India, Oct. 2016, pp. 198–204 (cit. on p. 5).

[10] B. Liu and N. Paquin. «Viconmavlink: A software tool for indoor positioning using a motion capture system». In: (Nov. 2018) (cit. on p. 6).

[11] R. Munguía, I. Urzua, Y. Bolea, and A. Grau. «Vision-Based SLAM System for Unmanned Aerial Vehicles». In: *Sensors* 16 (Mar. 2016), p. 372 (cit. on p. 6).

[12] *Leonardo Drone Contest: an open innovation challenge by Leonardo*. URL: `https://www.leonardocompany.com/it/innovation/open-innovation/drone-contest` (cit. on p. 7).

[13] D. Scaramuzza and Z. Zhang. «Visual-Inertial Odometry of Aerial Robots». In: *Encyclopedia of Robotics*. Ed. by M. H. Ang, O. Khatib, and B. Siciliano. Berlin, Heidelberg: Springer Berlin Heidelberg, 2020, pp. 1–9 (cit. on pp. 10, 50).

[14] J. Gui, D. Gu, S. Wang, and H. Hu. «A review of visual inertial odometry from filtering and optimisation perspectives». In: *Advanced Robotics* 29 (Sept. 2015), pp. 1–13 (cit. on p. 10).

[15] D. Scaramuzza and F. Fraundorfer. «Visual Odometry: Part I - The First 30 Years and Fundamentals». In: *IEEE Robot. Automat. Mag.* 18 (Dec. 2011), pp. 80–92 (cit. on pp. 11–13, 16, 30–33).

[16] K. Yousif, A. Bab-Hadiashar, and R. Hoseinnezhad. «An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics». In: *Intelligent Industrial Systems* 1 (Nov. 2015) (cit. on pp. 11, 12, 19, 23, 26, 37).

[17] S. Poddar, R. Kottath, and V. Karar. «Evolution of Visual Odometry Techniques». In: *CoRR* abs/1804.11142 (Aug. 2018) (cit. on p. 11).

[18] L. Matthies and S. Shafer. «Error modeling in stereo navigation». In: *IEEE Journal on Robotics and Automation* 3.3 (1987), pp. 239–248 (cit. on p. 12).

[19] D. Nister, 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. 2004, pp. I–I (cit. on pp. 12, 29).

[20] Y. Cheng, M. Maimone, and L. Matthies. «Visual odometry on the Mars Exploration Rovers». In: *2005 IEEE International Conference on Systems, Man and Cybernetics*. Vol. 1. 2005, 903–910 Vol. 1 (cit. on p. 12).

[21] D. Scaramuzza and R. Siegwart. «Appearance-Guided Monocular Omnidirectional Visual Odometry for Outdoor Ground Vehicles». In: *IEEE Transactions on Robotics* 24.5 (2008), pp. 1015–1026 (cit. on p. 12).

[22] M. Kaess, K. Ni, and F. Dellaert. «Flow separation for fast and robust stereo odometry». In: *2009 IEEE International Conference on Robotics and Automation*. 2009, pp. 3539–3544 (cit. on p. 12).

[23] P. Alcantarilla, L. Bergasa, and F. Dellaert. «Visual odometry priors for robust EKF-SLAM». In: June 2010, pp. 3501–3506 (cit. on p. 12).

[24] C. Harris and M. Stephens. «A combined corner and edge detector». In: *In Proc. of Fourth Alvey Vision Conference*. 1988, pp. 147–151 (cit. on p. 17).

[25] D. Tyagi. *Introduction to Harris Corner Detector*. Mar. 2019. URL: `https://medium.com/data-breach/introduction-to-harris-corner-detector-32a88850b3f6` (cit. on p. 17).

[26] E. Rosten and T. Drummond. «Machine Learning for High-Speed Corner Detection». In: *Computer Vision - ECCV 2006*. Ed. by A. Leonardis, H. Bischof, and A. Pinz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 430–443 (cit. on p. 19).

[27] S. Smith and M. Brady. «SUSAN - A New Approach to Low Level Image Processing». In: *International Journal of Computer Vision* 23 (2004), pp. 45–78 (cit. on p. 19).

[28] D. Lowe. «Distinctive Image Features from Scale-Invariant Keypoints». In: *International Journal of Computer Vision* 60 (Nov. 2004), pp. 91–110 (cit. on p. 20).

[29] E. Karami, S. Prasad, and M. Shehata. «Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images». In: Nov. 2015 (cit. on p. 20).

[30] D. Scaramuzza and F. Fraundorfer. «Visual odometry: Part II - Matching, robustness, optimization, and applications». In: *IEEE Robot. Automat. Mag.* 19 (June 2012), pp. 78–90 (cit. on pp. 21, 22, 28, 29, 34, 37).

[31] H. Bay, T. Tuytelaars, and L. Van Gool. «SURF: Speeded Up Robust Features». In: *Computer Vision - ECCV 2006*. Ed. by A. Leonardis, H. Bischof, and A. Pinz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 404–417 (cit. on p. 23).

[32] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. «BRIEF: Binary Robust Independent Elementary Features». In: vol. 6314. Sept. 2010, pp. 778–792 (cit. on p. 24).

[33] D. Tyagi. *Introduction to BRIEF(Binary Robust Independent Elementary Features)*. Mar. 2019. URL: `https://medium.com/data-breach/introduction-to-brief-binary-robust-independent-elementary-features-436f4a31a0e6` (cit. on p. 24).

[34] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. «ORB: An efficient alternative to SIFT or SURF». In: *2011 International Conference on Computer Vision*. 2011, pp. 2564–2571 (cit. on p. 26).

[35] K. Mikolajczyk and C. Schmid. «A performance evaluation of local descriptors». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27.10 (2005), pp. 1615–1630 (cit. on p. 26).

[36] B. Lucas and T. Kanade. «An Iterative Image Registration Technique with an Application to Stereo Vision». In: *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI'81. Vancouver, BC, Canada: Morgan Kaufmann Publishers Inc., 1981, pp. 674–679 (cit. on p. 28).

[37] C. Tomasi and T. Kanade. *Detection and Tracking of Point Features*. Tech. rep. International Journal of Computer Vision, 1991 (cit. on p. 28).

[38] G. Gallego, E. Mueggler, and P. Sturm. «Translation of "Zur Ermittlung eines Objektes aus zwei Perspektiven mit innerer Orientierung" by Erwin Kruppa (1913)». In: (Dec. 2017) (cit. on p. 32).

[39] D. Scaramuzza. «Tutorial on Visual Odometry». In: *Aerial and Service Robotics Summer School*. ETH Zürich, July 2012 (cit. on pp. 34–36).

[40] Oliver J. Woodman. *An introduction to inertial navigation*. Research report 696. University of Cambridge, Aug. 2007 (cit. on pp. 38, 39, 42).

[41] C. Forster, M. Pizzoli, and D. Scaramuzza. «SVO: Fast semi-direct monocular visual odometry». In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014, pp. 15–22 (cit. on pp. 52, 53, 55–57).

[42] T. Qin, P. Li, and S. Shen. «VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator». In: *IEEE Transactions on Robotics* 34.4 (2018), pp. 1004–1020 (cit. on pp. 59, 63, 65, 67, 68, 70–72).

[43] R. Mur-Artal, J. Montiel, and J. D. Tardos. «ORB-SLAM: a versatile and accurate monocular SLAM system». In: *IEEE Transactions on Robotics* 31 (Oct. 2015), pp. 1147–1163 (cit. on pp. 74, 75).

[44] R. Mur-Artal and J. D. Tardós. «ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras». In: *IEEE Transactions on Robotics* 33.5 (2017), pp. 1255–1262 (cit. on pp. 74, 83).

[45] R. Mur-Artal and J. D. Tardós. «Visual-Inertial Monocular SLAM With Map Reuse». In: *IEEE Robotics and Automation Letters* 2.2 (2017), pp. 796–803 (cit. on p. 83).

[46] C. Campos, R. Elvira, J. Rodríguez, J. Montiel, and J. Tardós. «ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM». In: (July 2020) (cit. on p. 83).

[47] Intel RealSense. *Case study - Robust Visual-Inertial Tracking from a Camera that Knows Where it's Going*. 2019. URL: `https://www.intelrealsense.com/visual-inertial-tracking-case-study` (cit. on p. 91).

[48] *Intel RealSense Tracking Camera T265 Datasheet*. 572522-002. Rev. 002. Intel RealSense. Mar. 2019 (cit. on p. 92).

[49] K. Hata and S. Savarese. «CS231A Course Notes 1: Camera Models». In: *Stanford University* (), pp. 1–17 (cit. on pp. 93, 96, 97).

[50] B. Dehem. «Three Dimensional Monocular SLAM for Autonomous Drone Navigation». Master Thesis. École polytechnique de Louvain, 2016/2017 (cit. on pp. 94, 95, 97).

[51] V. Chari and A. Veeraraghavan. «Lens Distortion, Radial Distortion». In: *Computer Vision: A Reference Guide*. Ed. by K. Ikeuchi. Boston, MA: Springer US, 2014, pp. 443–445 (cit. on p. 97).

[52] Duane C. Brown. «Close-range camera calibration». In: *Photogrammetric Engineering* 37.8 (1971), pp. 855–866 (cit. on p. 97).

[53] Z. Zhang. «A flexible new technique for camera calibration». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11 (2000), pp. 1330–1334 (cit. on p. 101).

[54] N. El-Sheimy, H. Hou, and X. Niu. «Analysis and Modeling of Inertial Sensors Using Allan Variance». In: *IEEE Transactions on Instrumentation and Measurement* 57.1 (2008), pp. 140–149 (cit. on p. 103).

[55] MATLAB Documentation. *Inertial Sensor Noise Analysis Using Allan Variance*. URL: `https://it.mathworks.com/help/nav/ug/inertial-sensor-noise-analysis-using-allan-variance.html` (cit. on pp. 105–108).

[56] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. Achtelik, and R. Siegwart. «The EuRoC micro aerial vehicle datasets». In: *The International Journal of Robotics Research* 35 (Jan. 2016) (cit. on p. 117).

[57] 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)*. 2018, pp. 2502–2509 (cit. on pp. 120–122).