POLITECNICO DI TORINO

College of Computer Engineering, Cinema and Mechatronics

Master's Degree in Mechatronic Engineering

Analysis of Visual-Inertial Techniques for UAV Navigation in GPS-denied Urban Environments

Supervisors:

Stefano Primatesta, Academic Supervisor;
Enza Incoronata Trombetta, Academic Tutor;

Emanuele Bezzecchi, Corporate Supervisor;
Marco Scafuro, Corporate Tutor

Candidate:

Maria Grazia Musio

315756

Academic Year 2023/24

Abstract

This thesis investigates the autonomous navigation of Unmanned Aerial Vehicles (UAVs) in Global Positioning System (GPS)-denied urban environments, focusing on a Visual Inertial Odometry (VIO) algorithm developed from scratch for vehicle pose estimation. In contexts where GPS signals are limited or not available, such as urban areas, VIO provides a promising solution by integrating visual information from a monocular camera with inertial data from an Inertial Measurement Unit (IMU). A monocular visual-inertial system acts as the minimal sensor suite for six Degrees of Freedom (DoF) metric state estimation, offering advantages in size, cost and simplicity of hardware setup. The main objective of this research is to design a monocular VIO algorithm using a loosely coupled approach that enables drones to navigate autonomously using only onboard sensors.

A significant challenge in monocular Visual Odometry (VO) is scale ambiguity. This problem occurs because a three-dimensional scene is reduced to a two-dimensional image plane, losing depth information. Despite this limitation, monocular systems are still a cost-effective choice and are often favored over stereo vision, especially when the distance of the scene from the camera is much greater than the stereo baseline. This issue is addressed through the fusion of IMU and camera data using an Error State Extended Kalman Filter (ES-EKF), which combines visual information with inertial measurements to resolve scale ambiguity and achieve reliable pose estimation.

This work also includes a thorough review of key visual localization components, analyzing various design approaches to assess their advantages and limitations. As feature extraction algorithm, the Scale-Invariant Feature Transform (SIFT) method is proposed. A detailed examination of SIFT is also done to understand its structure, feature extraction methods and its characteristics in general. Additionally, a theoretical analysis of the essential matrix and its computation methods is performed, focusing on the five-point algorithm used in 2D-to-2D methods for estimating camera motion in VO.

A synthetic dataset is created using the integration of PX4, Gazebo and QGroundControl frameworks. PX4 provides the flight control software, including necessary drone dynamics and control algorithms. Gazebo is used to develop a realistic 3D simulation environment, featuring various elements like buildings, roads and obstacles that closely replicate real-world conditions. Incorporating these details, the tailored environment provides a valuable testing Ground Truth (GT) for analyzing the effectiveness of the proposed algorithms in urban navigation scenarios. QGroundControl acts as the ground control station, enabling mission planning and monitoring of the UAV during simulations.

Therefore, this simulation setup allows for the design of various trajectories and the generation of useful data for analysis, as well as demonstrating the VIO algorithm's performance.

Contents

List of Figures

List of Tables

Acronyms

ARW Angle Random Walk

B Body Frame

BFMatcher Brute Force Matcher

BRIEF Binary Robust Invariant Scalable Keypoints

CENSURE CENsus-based SUbmap REfinement

DoF Degrees of Freedom

EKF Extended Kalman Filter

ES-EKF Error State Extended Kalman Filter

FAST Features from Accelerated Segment Test

FLU Forward-Left-Up

FRD Forward-Right-Down

GCS Ground Control Station

GPS Global Positioning System

GT Ground Truth

HDR High Dynamic Range

I IMU Frame

IMU Inertial Measurement Unit

INS Inertial Navigation System

KF Kalman Filter

MAE Mean Absolute Error

MEMS Micro Electro-Mechanical Systems

N Navigation Frame

NCC Normalized Cross-Correlation

NED North-East-Down

OF Optical Flow

ORB Oriented FAST and Rotated BRIEF

P3P Perspective-3-Point

PnP Perspective-n-Point

RANSAC Random Sample Consensus

RMS Root Mean Square

RMSE Root Mean Square Error

ROS2 Robot Operating System 2

SIFT Scale-Invariant Feature Transform

SITL Software In The Loop

SLAM Simultaneous Localization and Mapping

SSD Sum of Squared Differences

SURF Speeded Up Robust Features

SVD Singular Value Decomposition

UAV Unmanned Aerial Vehicle

UKF Unscented Kalman Filter

VIO Visual Inertial Odometry

VO Visual Odometry

W World Frame

Chapter 1

Introduction

UAVs are characterized by their small size, lightweight and agility. They are commonly referred as a pilotless aircraft with the capability to fly without requiring any human onboard operator, providing more cost-efficient operations than equivalent manned systems, and performing cost-efficient critical mission without risking human life. UAVs can be remotely piloted, whereby control commands are provided from a Ground Control Station (GCS) through a remote control [28].

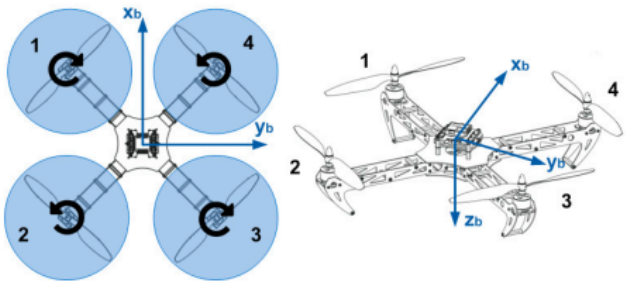


Figure 1.1: Structure and configuration UAV

Nowadays, UAVs are becoming essential in various fields such as search and rescue, environmental monitoring, security surveillance and inspection activities. The advantage of this device is the ability to navigate through outdoor and indoor spaces while minimizing risks to individuals. One of the main challenges of scientific research is the autonomous navigation of drones in GPS - denied environments, like in urban environments or in densely populated contexts. The strength of GPS satellite signals is highly dependent on environmental conditions, it is effective in areas with clear skies and is not suitable for indoor navigation where it is affected by walls and objects. Therefore, they are not a good candidate for precise localization which is one of the main module of autonomous navigation.

UAVs need the ability to independently explore unfamiliar environments, avoid obstacles, and build maps, but overcoming these challenges requires significant progress in helicopter design, perception, actuation, control, and navigation, all of which remain areas of ongoing development [29].

1.1 Background UAV localization

A central problem in aerial vehicle navigation is ensuring stabilization and control along six DoF, which include attitude and position control.

Unlike ground robots, whose mobility is conditioned by interaction with the terrain, UAVs benefit from their full three-dimensional motion capability, allowing them to move freely without being confined to the ground. Although they have a greater number of degrees of freedom than ground robots, three-dimensional navigation is still possible even with platforms equipped with a minimum number of sensors. The goal is to develop techniques that allow UAVs to move with the same dexterity and agility. However, there are three main constraints that affect the performance of UAVs [30]:

1. **Limited payload budget**: In UAVs lightweight sensors are essential because these vehicles have limited payload, which refers to anything the drone is carrying in addition to its own weight, such as sensors, cameras, or other equipment.

   When the payload increases the drone needs more energy to stay airborne. Specifically, every 10 grams of extra weight added to the drone requires approximately 1 Watt of additional power to keep the drone in a fixed position in the air [30]. Hovering requires continuous upward force from the rotors to counteract gravity, so any extra weight means the motors must work harder, consuming more power.

1. **Power consumption**: Minimizing computing power requirements leads to lighter and more energy-efficient processing units. Conversely, higher power demands from sensors and computing units reduce the energy available for propulsion, impacting the vehicle's range.

1. **Availability of sensor signals**: While ground robots can maintain close proximity to their environment, airborne vehicles may operate at significant distances, where distance-measuring sensors or small baseline stereo vision setups can fail. Additionally, GPS signals may be weak or entirely absent in certain conditions.

1.2 Motivation and objectives

In recent years, significant research has focused on odometry techniques for localization, which involves identifying the position of an object in space. In environments where GPS signals are not available, localization can be achieved using external sensors (e.g., motion capture systems) or on-board sensors (e.g., cameras and laser rangefinders). While external sensors enable precise location tracking, they also limit autonomy since the system, which relies on external equipment, is limited to small and confined spaces, making autonomous navigation in unknown and yet unexplored environments impossible. Therefore, fully autonomous navigation for UAVs requires on-board sensors [29].

Addressing key constraints, previously defined, namely limited payload, power consumption, and sensor reliability, research has increasingly focused on vision-based navigation combined with inertial measurements from proprioceptive sensors such as IMUs. Advances in cameras and IMUs have helped to meet payload constraints, while the challenge of low power consumption drives the need for efficient algorithms to process large amounts of visual data. Since cameras are passive sensors that only capture reflected light, their performance can diminish in low-light or visually homogeneous environments. To mitigate these limitations, integrating multiple sensor types and fusing their data during flight can significantly enhance both accuracy and robustness in navigation.

VIO attracts considerable attention in the field of UAV research because the algorithm efficiently integrates the rich representation of a scene captured in an image, along with accurate short-term measurements from the IMU. Therefore, the focus of this thesis is on autonomous local navigation in a GPS-denied environment of a UAV using only on-board sensors, such as an IMU and a monocular camera. A weakly coupled monocular inertial odometry based on the ES-EKF algorithm is proposed. Due to the characteristics of the weakly coupled architecture, the algorithm has been divided into two main sections: VO part and ES-EKF part.

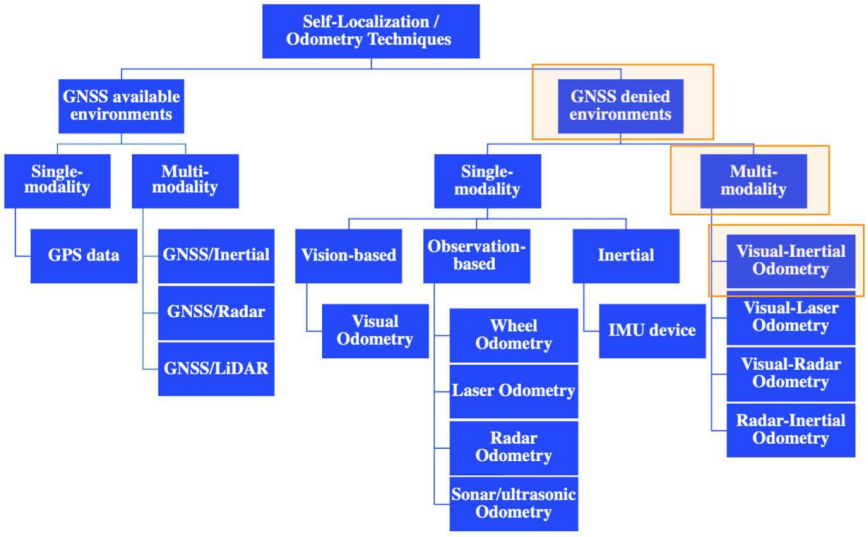


Figure 1.2: Self-localization odometry techniques [1]

1.3 Structure of the Thesis

The thesis is organized as follows:

- **Chapter 1: Introduction** - This chapter provides an overview of the research, detailing the background of UAV localization, the motivation behind the study, and the objectives pursued in this work. It lays the foundation for the technical aspects discussed in the following chapters.

- **Chapter 2: Sensors** - This chapter introduces the sensors used in the system, focusing on the camera and the IMU. The different types of sensors are presented, including their mathematical models and specific characteristics.

- **Chapter 3: Visual Odometry** - In this chapter, VO as a technique for UAV localization is explained. The advantages, challenges, and various approaches to VO, such as appearance-based, feature-based, and hybrid methods, are discussed. Special attention is given to monocular VO and the challenges it faces, especially scale ambiguity.

- **Chapter 4: Feature-based approach** - Here, a feature-based approach to VO is explored in detail. The chapter covers the key elements of VO systems, including keypoint detectors, descriptors, and the performance of different feature detection algorithms. Motion estimation methods and optimization techniques such as bundle adjustment are also explored.

- **Chapter 5: Visual inertial odometry** - This chapter extends VO by incorporating inertial data to improve accuracy and robustness. Various VIO approaches, including loosely- and tightly-coupled methods, are discussed. Data fusion techniques with their advantages and challenges are also explained in this context.

- **Chapter 6: Simulation Environment and Synthetic Datasets** - A detailed overview of the simulation environment used to test the algorithms is provided. This chapter includes the tools and frameworks used, such as Robot Operating System 2 (ROS2), PX4, Gazebo, and QGroundControl, as well as the reference frames for the simulation.

- **Chapter 7: Synchronization** - This chapter discusses time and data synchronization between simulation components, such as PX4 and Gazebo. Proper synchronization is essential for accurate processing of sensor data.

- **Chapter 8: Implementation** - This chapter presents the implementation of the VIO system. It describes the design of the ES-EKF, detailing the kinematics, prediction, and correction phases of the filter.

- **Chapter 9: Results** - In this chapter, the results of the system implementation are presented. This includes the performance of the VO system, the integration of the IMU, and the evaluation of the ES-EKF. Several cases are examined and attention is also paid to scale estimation.

- **Chapter 10: Enhanced VIO with a Depth Camera: Preliminary Implementation** - In this chapter, an alternative method to VO is explored. The depth details of this approach are explained, followed by an implementation and analysis of the results using the ES-EKF.

- **Chapter 11: Conclusions and Future Developments** - The final chapter summarizes the main results of the thesis, discusses the implications of the results, and proposes potential avenues for future research and improvements in the system.

Each chapter builds on the previous one, contributing to a comprehensive understanding of the system design, implementation and evaluation. Through this structured approach, the thesis aims to provide both theoretical insights and practical solutions to improve UAV localization using VIO.

Chapter 2

Sensors

This thesis focuses on the use of a monocular camera and a IMU. Our primary goal is to estimate the pose of a UAV using these sensors but first we analyze in detail the characteristics of the sensors we worked with.

2.1 Camera

One of the most crucial tools in computer vision is camera, that allows us to capture the world around us and utilize the images it produces for a wide range of applications.

A camera is an optical device that captures light through an optical system and stores this information digitally using a light-sensitive electronic sensor. The digital image produced is then processed by computer vision algorithms for tasks such as motion estimation.

The fundamental principle of camera technology involves measuring light intensities — quantifying the amount of light reaching the camera from various directions. At the core of this process is the light-sensitive chip which consists of numerous small regions, each corresponding to a pixel in the final image. Each of them act as sensors, measuring light intensity and essentially turning the chip into a photon counter.

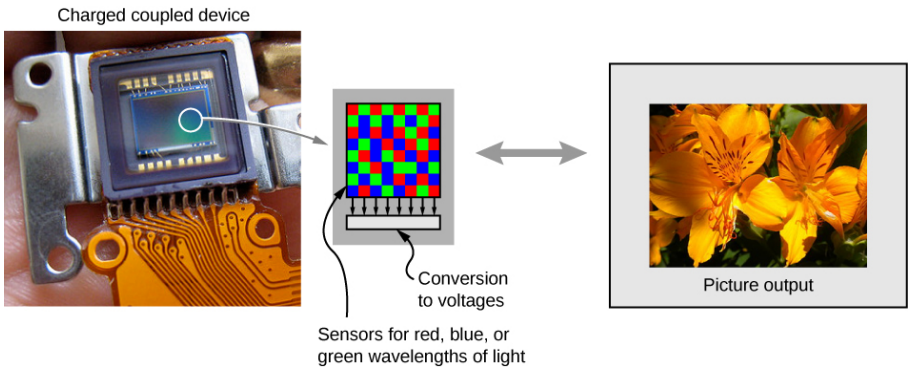


Figure 2.1: Electronic processing and storage of visual images. This is the basis for electronic imaging in all digital cameras.

When an image is captured, the chip records multiple measurements across its surface since the camera lens directs light rays toward these photon-counting pixels and measures the intensity of light arriving from a specific direction in space. Thus, the camera generates an image by measuring the light intensities from various directions.

2.1.1 Camera for geometric measurements

To use cameras for geometric measurements, it's important to consider how light interacts with the scene: a light source illuminates an object and the object reflects light to the camera's sensor, where specific pixels register higher intensity. To identify which 3D points in the environment correspond to which pixels in the image, extraction of keypoints and features, using methods such as SIFT or Speeded Up Robust Features (SURF), is required. These feature extractors analyze the image to detect locally distinct patterns, such as corners or high-contrast gradients and these distinct points, identified through feature descriptors are presumed to map 3D world points onto the 2D image plane.

By focusing on these key points, the task shifts to estimating their positions on the 2D image plane. Ultimately, the goal is to reconstruct the location of these points in the 3D environment by combining multiple images of the same object taken from different angles [31].

2.1.2 Camera Model

A camera model is a mathematical framework used to describe how a camera captures and represents visual information. It defines the relationship between the 3D world coordinates and the 2D image coordinates.

This mapping is described as:

$$x = PX$$

x: 2D Image point, P: Projection matrix, X: 3D world point

The projection of 3D points into the image plane does not directly correspond to what we see in actual digital images, for several reasons. First, points in the digital images are, in general, in a different reference system than those in the image plane and this reference system is the world reference frame. Second, digital images are divided into discrete pixels, whereas points in the image plane are continuous. Finally, the physical sensors can introduce non-linearity such as distortion to the mapping. To account for these differences, we will introduce a number of additional transformations that allow us to map any point from the 3D world to pixel coordinates [3].

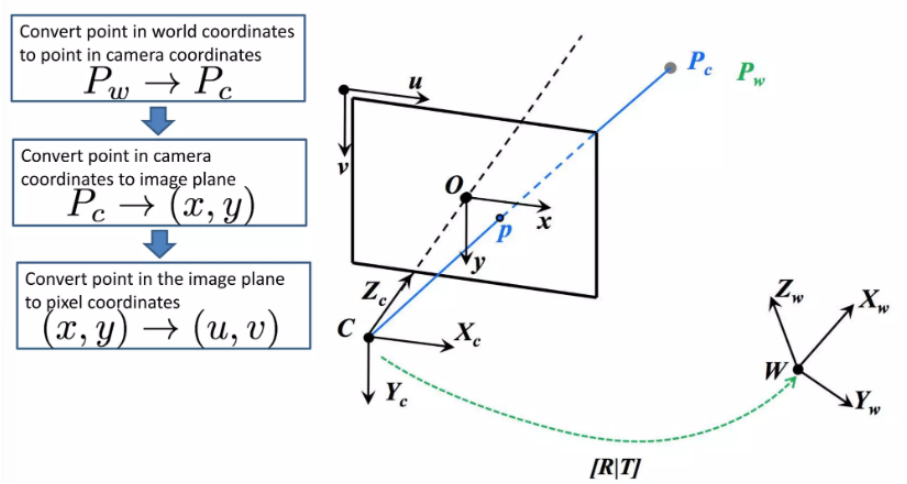


Figure 2.2: Steps of camera model [2]

World-to-Camera transformation: The world-to-camera transformation involves projecting 3D coordinates in the world coordinate system to a 3D camera coordinate system. This transformation includes rotations, scaling and translation. Mathematically, this can be represented by a transformation matrix:

$$\mathbf{T}_{\text{world-to-camera}} = \mathbf{R} \cdot \mathbf{s}\,\mathbf{t}$$

where $\mathbf{R}$ is the rotation matrix, $\mathbf{s}$ is the scaling matrix, and $\mathbf{t}$ is the translation vector.

Camera-to-Image transformation: The camera-to-image transformation maps 3D coordinates from the camera's coordinate system onto a 2D image plane. This process depends on the camera model being used and results in a loss of depth information, as the transformation reduces the 3D scene into a 2D representation.

Image-to-Pixel transformation: The image-to-pixel transformation involves converting continuous image coordinates to discrete pixel coordinates. Mathematically, if (x, y) are the continuous image coordinates, and (u, v) are the pixel coordinates, this can be represented as:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \left\lfloor \frac{x}{\Delta x} \right\rfloor \\ \left\lfloor \frac{y}{\Delta y} \right\rfloor \end{bmatrix}$$

where Δx and Δy are the sizes of the pixels in the x and y directions, respectively.

2.1.3 Pinhole Camera model

A simple camera system can be created by placing a barrier with a small aperture between a 3D object and a photographic film or sensor. This setup allows only specific light rays from the object to pass through the aperture, creating a direct mapping of points from the 3D object to the film, resulting in a clear image of the object [3].

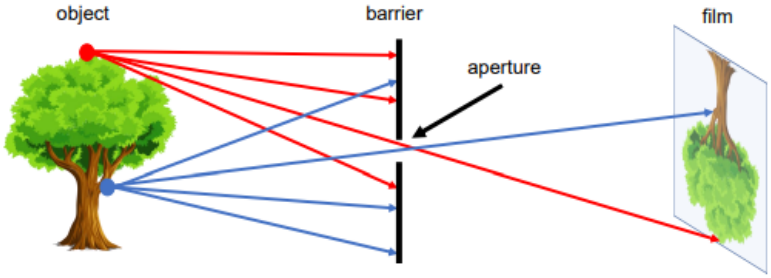


Figure 2.3: A simple working camera model: the pinhole camera model [3]

The pinhole camera model is the simplest camera model. It places the center of projection, through which all light rays pass before reaching the image plane, at the origin of a Euclidean coordinate system. The image plane, also known as the focal plane, is situated at $Z = f$, where f represents the focal length, as it is shown in 2.4.

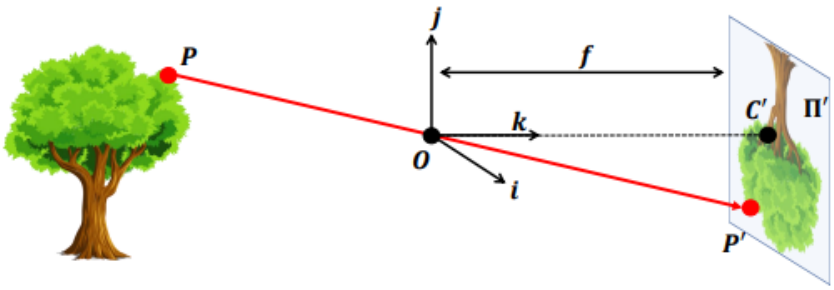


Figure 2.4: A formal construction of the pinhole camera model [3]

Relation between image coordinate system and the camera coordinate system

In the pinhole camera model, using similar triangles, it is possible to derive the mathematical relationship between the 3D coordinates (X, Y, Z) and the 2D coordinates (x, y) on the image plane as follows:

$$(x, y) = \left(\frac{fX}{Z}, \frac{fY}{Z}\right)$$

where f is the focal length of the camera. Thus, a point in 3D space with coordinates $(X, Y, Z)^T$ is mapped to the image plane point $(fX/Z, fY/Z)^T$, as illustrated in Figure 2.5:

$$(X, Y, Z)^T \rightarrow \left(\frac{fX}{Z}, \frac{fY}{Z}\right)^T \tag{1}$$

Assuming that world and image points are represented in homogeneous coordinates, central projection can be described as a linear transformation between these coordinates through matrix multiplication:

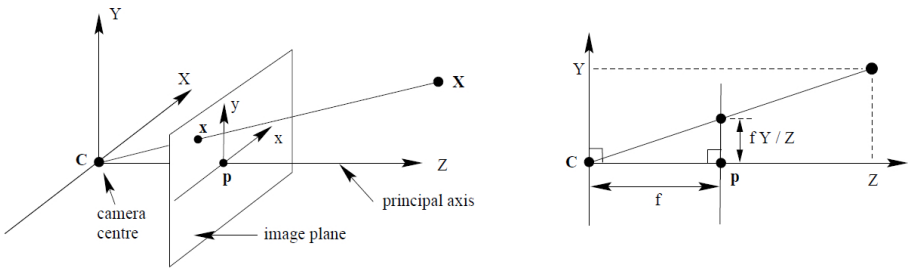


Figure 2.5: Pinhole camera geometry. The center of projection is called the camera center or optical center. The line from the camera center perpendicular to the image plane is the principal axis or principal ray. The point where the principal axis meets the image plane is the principal point. The plane through the camera center parallel to the image plane is the principal plane of the camera. The camera center is placed at the coordinate origin [4].

$$\begin{bmatrix} fX \\ fY \\ Z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_{\text{cam}} \\ Y_{\text{cam}} \\ Z_{\text{cam}} \\ 1 \end{bmatrix}$$

Principal point offset

In an ideal camera model, the origin of the coordinate system on the image plane (where $X = 0$ and $Y = 0$) coincides with the principal point of the camera.

Principal Point

The principal point is defined as the location where the optical axis intersects the image plane.

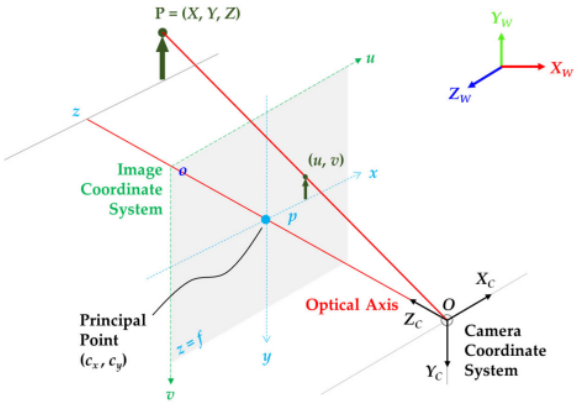


Figure 2.6: Principal point [5]

However, in practical scenarios, the optical axis might not perfectly align with the center of the image plane due to imperfections in the camera or lens. Consequently, the principal point may not be exactly at the origin of the coordinate system used for the image plane. To account for this offset in the principal point, we modify the projection equations. The adjusted projection can be expressed as:

$$\begin{bmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & p_x \\ 0 & f & 0 & p_y \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_{\text{cam}} \\ Y_{\text{cam}} \\ Z_{\text{cam}} \\ 1 \end{bmatrix} \tag{3}$$

where p_x and p_y are the offsets of the principal point from the origin of the image coordinate system, and f is the focal length of the camera.

The matrix on the right side of Equation (3) is called the camera calibration matrix, typically denoted as K. For generality, the calibration matrix can be expressed as:

$$K = \begin{bmatrix} f_x & s & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

where s is the skew parameter, which is zero for most cameras, f_x and f_y are the camera's focal length in pixel dimensions along the x-axis and y-axis, while the coordinates (p_x, p_y) represent the principal point.

Relation between camera coordinate system and the world coordinate system

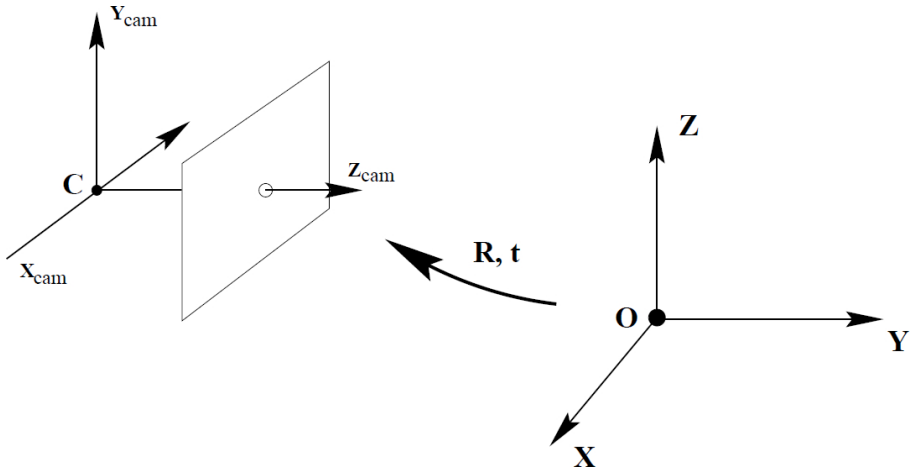


Figure 2.7: The Euclidean transformation between the world and camera coordinate frames [4].

The relationship between the camera coordinate frame and the world coordinate frame is defined by rotation and translation. As shown in Figure 2.7, if $X = (X, Y, Z, 1)^T$ represents a point in world coordinates, then X_{cam} is transformed by:

$$X_{\text{cam}} = [R \mid t]X \tag{6}$$

where R is a 3×3 rotation matrix and t is a 3×1 translation vector.

The general mapping of a pinhole camera in the world coordinate frame x is given by:

$$x = K[R \mid t]X \tag{7}$$

The internal camera parameters K define the camera's intrinsic properties which describe the mapping of the scene in front of the camera to the pixels in the final image (sensor). Since these parameters are intrinsic properties of the camera that don't change during the pose estimation they can be computed once for all during the calibration procedure. While the external parameters R and t describe the camera's orientation and position relative to the world coordinate system. These parameters change as the camera moves in space. [4].

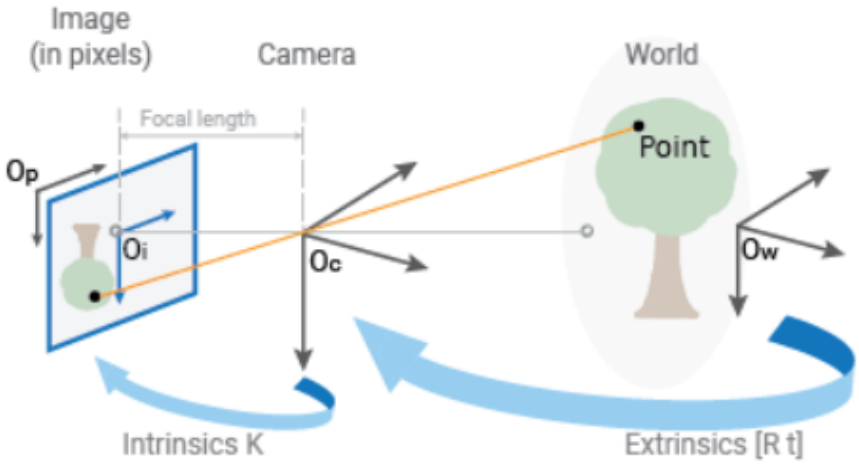


Figure 2.8: Intrinsic and extrinsic parameters

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:

$$P = K[R \mid t] = \begin{bmatrix} f & 0 & 0 & p_x \\ 0 & f & 0 & p_y \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_1 & r_4 & r_7 & t_1 \\ r_2 & r_5 & r_8 & t_2 \\ r_3 & r_6 & r_9 & t_3 \end{bmatrix} \tag{8}$$

where P = K[R t] is called the projection matrix. This matrix has 9 degrees of freedom: three for K (i.e., f, p_x, p_y), three for the rotation matrix R, and three for the translation vector t [3].

The pixel coordinates provided by P are scaled, but this problem can be resolved by normalizing the pixel coordinates so that the final coordinate equals one.

2.2 Inertial Measurement Unit

IMU, standing for Inertial Measurement Unit, is an electromechanical or solid-state device that measures and provides acceleration, orientation, angular rates, and also gravitational forces. It is composed of 3 accelerometers, 3 gyroscopes, and depending

on the heading requirement, 3 magnetometers — each corresponding to one of the vehicle's three axes: roll, pitch, and yaw [32].

The IMU measures an object's motion by detecting the forces acting on it due to its resistance to changes in direction, then it converts these detected forces into data that describes how the object is moving. The output of an IMU is typically the raw sensor data from:

- **Accelerometers**: These sensors measure the rate of change in the object's speed along each of the three axes (x, y, and z), which is known as linear acceleration.
- **Gyroscopes**: They measure how quickly the object is rotating around each of the three axes, which is known as angular velocity.

There are two main categories of IMUs, distinguished by the frame of reference in which their gyroscopes and accelerometers operate: Stable platform IMU and Strap-down IMU. Their configurations are shown in Figures 2.9 and 2.10.

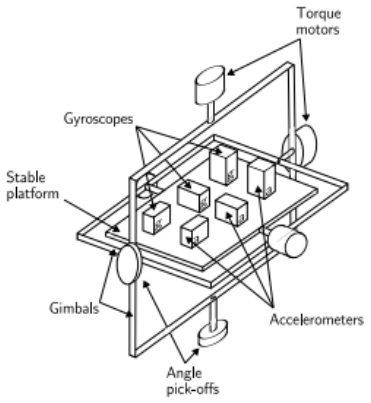


Figure 2.9: Stable platform IMU [6]

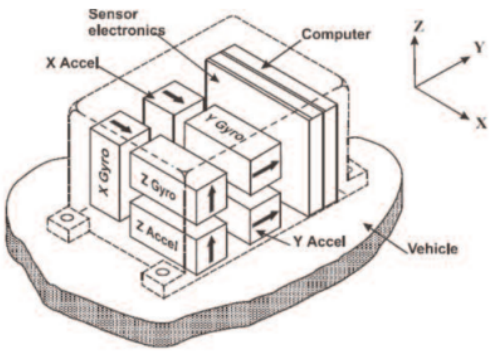


Figure 2.10: Strap-down IMU

In a **stable platform IMU** the sensors (gyroscopes and accelerometers) are mounted on a mechanically stabilized platform that maintains a fixed orientation relative to the inertial space. This setup allows the platform to stay level, minimizing errors due to platform motion but requiring complex gimbal systems to achieve stabilization.

In a **strap-down 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 strap-down system is the most widely used inertial system for aerial robots.

Among the various types of sensors available, strap-down systems frequently utilize MEMS sensors. These sensors are preferred for their benefits, which include

compact size, light weight, low cost, low power consumption, and rapid start-up time. Detailed descriptions and error characteristics of MEMS gyroscopes and accelerometers are discussed in the following sections [33].

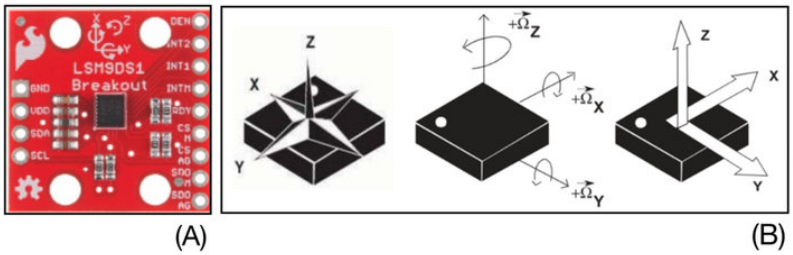


Figure 2.11: (A) MEMS. (B) 3-axis accelerometer, a 3-axis gyroscope, a 3-axis magnetometer and a temperature sensor [7]

2.2.1 MEMS Gyroscopes

A MEMS gyroscope is a sensor produced using silicon micro-machining techniques. Its operation is based on the Coriolis effect and according to this principle, when a mass m moves with velocity v within a rotating frame of reference with angular velocity ω, it experiences a force given by:

$$\mathbf{F}_c = -2m(\boldsymbol{\omega} \times \mathbf{v}) \tag{2.1}$$

The Coriolis effect is detected using various vibrating elements, including designs like vibrating wheels and tuning forks. In the most basic setup, a mass is made to vibrate along a primary drive axis and when the gyroscope rotates, this motion causes an additional vibration along a perpendicular sense axis, resulting from the Coriolis effect. By measuring this secondary vibration, the angular velocity can be determined. The vibrating mass gyroscope is illustrated in Figure 2.12.

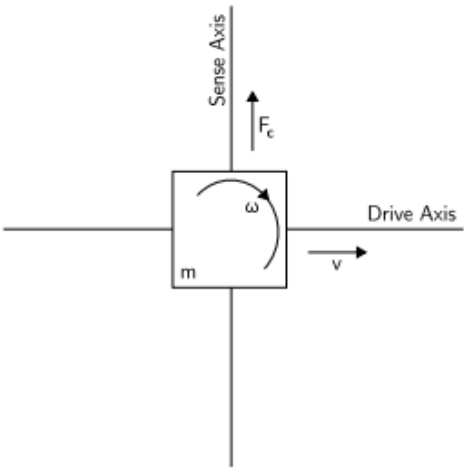


Figure 2.12: A vibrating mass gyroscope

Errors in MEMS gyroscopes

MEMS gyroscopes are subject to several types of errors, which can also affect the accuracy of the integrated orientation signal. The key errors are outlined briefly below:

- **Constant Bias Error**
  The constant bias error of a gyroscope is the fixed deviation in the sensor's output when it is not subjected to any rotational motion. This error represents a persistent offset from the true angular velocity, resulting in a reading that is consistently higher or lower than the actual value. The bias error, often measured in degrees per hour (°/h), leads to a systematic drift in the output signal over time, causing accumulated inaccuracies in the gyroscope's measurements. To correct for this error, the bias must be determined and then subtracted from the sensor readings to ensure accurate angular velocity measurements.

- **Thermo-Mechanical White Noise / Angle Random Walk**
  Thermo-Mechanical White Noise or Angle Random Walk in a gyroscope refers to the random fluctuations in the gyroscope's output caused by internal noise sources such as thermal and mechanical disturbances. This noise fluctuates at a rate much greater than the sampling rate of the sensor, as a result it is typically modeled as a white noise process with zero mean.

  Since we are usually interested in how the noise affects the integrated signal it is common for manufacturers to specify noise using an Angle Random Walk (ARW) measurement, which represents the Root Mean Square (RMS) of the error in the angle measurement over time. Specifically, the random fluctuations result in an increasing uncertainty in the angle estimation, with the standard deviation of this uncertainty growing proportionally to the square root of the time elapsed:

  $$\text{ARW} = \sigma \cdot \sqrt{\delta t \cdot t}$$

  where σ represents the standard deviation of the white noise, δt and t the elapsed time and the actual time instant.

- **Flicker Noise / Bias Stability**
  Flicker Noise, also known as 1/f noise, is a type of low-frequency noise that affects the stability of a gyroscope's bias. This noise originates from various sources such as fluctuations in the electronics of the sensor and becomes more pronounced at lower frequencies. Flicker noise introduces random fluctuations in the gyroscope's output, which can lead to variations in the bias over time.

  Bias Stability refers to the sensor's ability to maintain a consistent bias value despite these fluctuations. It is a measure of how much the gyroscope's bias drifts or varies over a period. Bias stability is typically quantified by assessing the bias drift over a fixed time interval under stable conditions. The goal is to minimize the effects of flicker noise to ensure that the gyroscope's output remains as stable and accurate as possible. These fluctuations can be modeled as a bias random walk with a standard deviation that grows with the square root of time.

The effects on orientation from integrating the gyroscope signal are described by a second-order angle random walk. Bias stability, which measures the change in bias over 100 seconds under fixed conditions, can sometimes be expressed in terms of bias random walk:

$$\text{BRW}\left[\text{rad/s}^{\frac{3}{2}}\right] = \frac{\text{BS}\,[\text{rad/s}]}{\sqrt{t}\,[\text{s}]} \tag{2.2}$$

- **Temperature Effects**
  Temperature changes, whether from environmental shifts or the sensor's own heating, can cause shifts in the bias. Any remaining bias caused by these temperature variations will result in orientation errors that increase linearly over time.

  These temperature-induced changes can cause the gyroscope's measurements to drift or become less reliable, introducing errors in the estimation of angular velocity. To mitigate these effects, gyroscopes may be calibrated across a range of temperatures, or they may include temperature compensation mechanisms to maintain accuracy in varying thermal conditions.

- **Calibration Errors**
  Calibration error in a gyroscope refers to inaccuracies in the sensor's output due to deviations in its calibration parameters. These errors arise from imperfections in the gyroscope's scale factors, misalignment of the sensing axes, or non-linearities in its response. As a result, the gyroscope's measurements may not accurately reflect the true angular velocity, leading to biased or incorrect readings.

  Calibration errors can cause the output to drift over time, particularly during rotations, and may require compensation or recalibration to ensure precise measurements.

The relative importance of each error source varies across different gyroscopes. For MEMS gyroscopes ARW (noise) errors and uncorrected bias errors either due to uncompensated temperature fluctuations or an error in the initial bias estimation are usually the most important sources of error. ARW can be used as a lower bound for uncertainty in the orientation obtained from integrating a rate-gyroscope's signal.

2.2.2 MEMS Accelerometer

A MEMS accelerometer features a proof mass connected to its frame by mechanical springs, allowing the mass to move along the sensitivity axis. Acceleration is measured by tracking the displacement of this mass.

Displacement is detected using electrical capacitance. The accelerometer includes several differential capacitors with electrodes positioned on either side of the proof mass. The proof mass has "fingers" extending into the gap between these electrodes. When the accelerometer is stationary, the fingers are centered between the

electrodes, providing a baseline capacitance that corresponds to zero acceleration. During acceleration, the proof mass shifts due to its different rate of acceleration compared to the reference frame, causing the fingers to move closer to one electrode. This movement changes the capacitance, which is then used to calculate the applied acceleration [32].

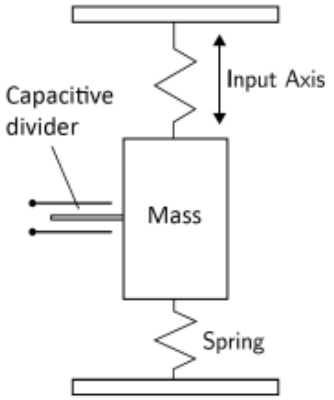


Figure 2.13: Illustration of a vibrating mass accelerometer- The displacement of the mass is converted by the capacitive divider into an electric signal proportional to the acceleration applied to the mass along the input axis

Errors in MEMS accelerometer

The main sources of error for MEMS accelerometers are similar to those for gyroscopes, also in this case errors affect the accuracy of the integrated position signal.

- **Constant Bias Error**
  The bias of an accelerometer refers to the constant error present in the sensor's readings when no actual acceleration is occurring. This bias represents the difference between the measured value and the true acceleration. When this constant error is integrated twice to compute position, it results in a positional error that increases quadratically over time.

  To correct this error, the bias is subtracted from the sensor's measurements. However, estimating the bias can be challenging because gravity's influence on the accelerometer can be mistaken for bias. This issue is addressed during calibration, where the device's orientation is known, allowing for accurate measurement and compensation of the bias.

- **Thermo-Mechanical White Noise / Velocity Random Walk**
  The measurements from a MEMS accelerometer are affected by thermo-mechanical noise that fluctuates at a frequency higher than the accelerometer sampling rate. This noise results in a white noise sequence, which consists of zero-mean, uncorrelated random variables. The white noise introduces a velocity random walk error into the sensor's measurements. The goal is to determine the impact of this white noise on the position obtained by double-integration the accelerometer output.

Accelerometer white noise introduces a second-order random walk error in position, with the standard deviation growing proportionally to $t^{3/2}$:

$$\sigma_s(t) \approx \sigma \cdot t^{3/2} \cdot \sqrt{\frac{\Delta t}{3}}$$

- $\sigma_s(t)$: Standard deviation of the position error due to accelerometer white noise after double integration over time t. It quantifies the uncertainty in the estimated position caused by noise.
- σ: Standard deviation of the white noise in the accelerometer measurements that represents the intensity of the random fluctuations in the measured acceleration.
- Δt: Sampling interval, the time between successive accelerometer measurements. It influences how quickly the noise accumulates in the position estimate.
- $\sqrt{\frac{\Delta t}{3}}$: Scaling factor that accounts for the sampling interval's impact on the position error, derived from the integration process of the white noise.

- **Flicker Noise / Bias Stability**
  In MEMS accelerometers, bias stability and flicker noise significantly influence measurement accuracy. The bias fluctuations due to flicker noise can be modeled as a random walk, where the standard deviation of the noise increases proportionally with the square root of time. As the accelerometer signal is integrated, these fluctuations lead to a third-order random walk in the computed position. Consequently, the standard deviation of the position error grows at a rate proportional to $t^{5/2}$, reflecting the compounded impact of both flicker noise and bias instability over time.

- **Temperature Effects**
  Bias fluctuations in an accelerometer, similar to those in a gyroscope, can result from temperature changes due to environmental conditions and sensor overheating. These fluctuations cause errors in position measurements that increase quadratically over time.

- **Calibration Errors**
  Calibration errors, such as those in scale factors, alignment, and linearity, introduce a bias in the accelerometer measurements. This bias can be detected even when the accelerometer is at rest due to the influence of gravitational acceleration.

2.2.3 Mathematical model

Coordinate Frames

When working with IMUs, two distinct coordinate frames need to be defined:

- **Body Frame (B) or IMU Frame (I)**: This coordinate frame is rigidly attached to the IMU itself. As the IMU moves with the vehicle, the body frame moves accordingly, reflecting the vehicle's motion.

- **Navigation Frame (N) or World Frame (W)**: This frame is fixed relative to the Earth's surface and typically represents the initial reference point of the vehicle's position. The vehicle's movement is recorded relative to this stationary frame.

IMU navigation equations

The measurements from the accelerometer and gyroscope are used to estimate the agent's position and orientation through dead reckoning, removing the need for external references. This means the system can determine the agent's pose using only the internal IMU data, without relying on external sources like GPS.

Dead reckoning

Dead reckoning is a navigation and position determination method that relies on mathematical calculations to estimate an object's current position. It uses data on velocities and directions measured (such as speed, acceleration and rotational angles) starting from a known previous position to estimate the object's new position over time.

Specifically, inertial navigation, starts with a known initial position and updates the position continuously based on data from inertial sensors such as accelerometers and gyroscopes. Error models are employed to account for biases and inaccuracies in the sensor data, ensuring more accurate estimations of the object's position and orientation over time. This procedure is shown in Figures 2.14.

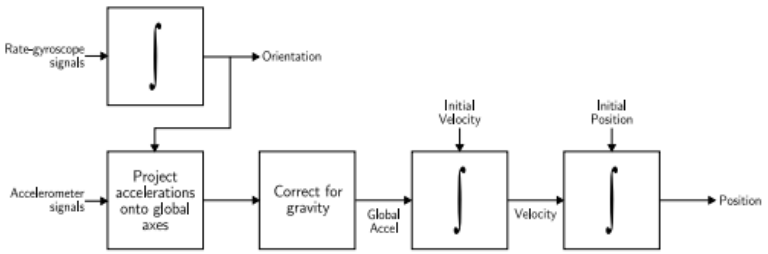


Figure 2.14: Strapdown inertial navigation algorithm

Orientation

The system's orientation relative to a global reference frame is determined by integrating angular velocity, starting from the initial orientation. After estimating the attitude, the linear acceleration is projected into the global frame based on the calculated orientation.

Since gyroscope measurements are subject to noise and bias, error models are employed to correct these inaccuracies, ensuring more precise orientation estimations over time. The integration of these measurements by the attitude algorithm propagates errors through the algorithm, resulting in unreliable attitude estimation.

For a MEMS gyroscope, the most significant sources of error in orientation are angular velocity bias and measured noise. The angular velocity bias introduces an error in orientation that grows linearly with time and it is modeled as a random walk driven by Gaussian white noise (n_{b_ω}) meaning its changes are random and follow a normal distribution. This can be expressed mathematically as:

$$\dot{b_\omega} = n_{b_\omega}$$

where n_{b_ω} is Gaussian white noise. Measurement noise n_ω is modelled as a white Gaussian with zero mean.

Random Walk

A random walk is a mathematical concept used to describe a stochastic process where a variable undergoes a series of random steps, and its future state depends on its current state plus a random increment. A common way to represent a random walk is:

$$X_{t+1} = X_t + \epsilon_t$$

where:

- X_t is the state or position at time t,
- ϵ_t is the random increment at time t, typically drawn from a probability distribution.

Additionally, quantization errors arise in the attitude algorithm due to the quantization of angular velocity samples and the integration model used for updating the orientation.

On the base of the previous analysis, the equation that characterizes the angular velocity measured by the gyroscope is defined as follows. Let $\boldsymbol{\omega}_t$ denote the true angular velocity, free from any external disturbances, and $\boldsymbol{\omega}_m$ represent the measured angular velocity from the sensor. The relationship between $\boldsymbol{\omega}_t$ and $\boldsymbol{\omega}_m$ can be described by:

$$\boldsymbol{\omega}_m = \boldsymbol{\omega}_t + b_\omega + n_\omega \tag{2.3}$$

where b_ω is the angular velocity bias and n_ω is the measurement noise.

Position

The accelerometer measures linear acceleration along the three principal axes of the global reference frame. In this context, the acceleration recorded by the sensor is denoted as $\mathbf{a}_m$, while $\mathbf{a}_t$ represents the true acceleration, free from any external disturbances. The relationship between $\mathbf{a}_t$ and $\mathbf{a}_m$ can be expressed as:

$$\mathbf{a}_m = C_q^{-1}(\mathbf{a}_t - \mathbf{g}) + b_a - n_a \tag{2.4}$$

Here, b_a is the acceleration bias, n_a represents the measurement noise, modeled as Gaussian white noise with zero mean. The vector $\mathbf{g}$ account for the acceleration due to gravity and C_q s the rotation matrix that aligns the body's orientation with the global coordinate system.

The acceleration bias b_a is not constant but varies over time. It is modeled as a random walk with bias noise n_b, which is also Gaussian white noise with zero mean:

$$\dot{b}_a = n_b \tag{2.5}$$

To compute the velocity of the system relative to the global frame, the linear acceleration is integrated after accounting for the gravitational component:

$$\tilde{v}_G(t) = \tilde{v}_G(0) + \int_0^t [\tilde{a}_G(t) - g_G]\, dt \tag{2.6}$$

where $\tilde{v}_G(0)$ is the initial velocity of the system and g_G is the acceleration due to gravity.

The system's position with respect to the global frame is then determined by integrating the computed velocity, starting from the initial position of the system:

$$\tilde{p}_G(t) = \tilde{p}_G(0) + \int_0^t \tilde{v}_G(t)\, dt \tag{2.7}$$

Since the IMU provides linear acceleration measurements at discrete intervals, similar to the attitude estimation process, an integration model must be applied to these samples. Using the rectangular rule for numerical integration, the update equations for velocity and position are given by:

$$\tilde{v}_G(t + \Delta t) = \tilde{v}_G(t) + \Delta t \cdot [\tilde{a}_G(t + \Delta t) - g_G] \tag{2.8}$$

$$\tilde{p}_G(t + \Delta t) = \tilde{p}_G(t) + \Delta t \cdot \tilde{v}_G(t + \Delta t) \tag{2.9}$$

In the previous sections, we discussed how errors can affect measurements from accelerometers. When these measurements are processed through double integration, errors can build up and cause significant problems in determining position. These issues can be made worse by errors from gyroscopes, as the orientation data used to convert acceleration into the global frame might be incorrect. Problems such as wrong adjustments for gravity and mistakes in the direction of integration can lead to large errors in position estimates over time.

Chapter 3

Visual Odometry

The term "odometry" is derived from two Greek words: hodos, meaning "journey" or "travel," and metron, meaning "measure" (Fernandez and Price, 2004). VO is the process of estimating the ego-motion of an agent—such as a vehicle, human or robot—using input from one or more cameras mounted on the agent [2].

Images provide a wealth of meaningful information, including color, texture, and shape, which can be used to estimate the camera's movement within a static environment. In static environment the surroundings (such as objects, structures, or the landscape) remain stationary or unchanged while the camera moves, this makes it easier to estimate the camera's motion because the features in the environment can be assumed to stay in the same place relative to each other (Rone and Ben-Tzvi, 2013).

VO operates by incrementally estimating the agent's pose, analyzing the changes in the images captured as the agent moves.

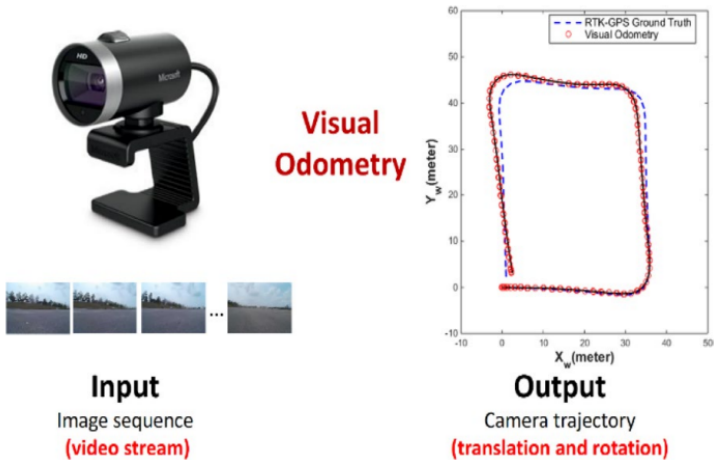


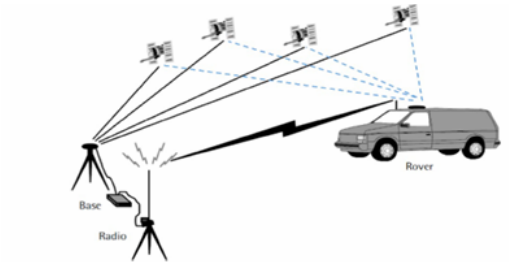
Figure 3.1: Visual Odometry: Input and Output. The input to the visual odometry system consists of a sequence of images captured by a camera, while the output is the estimated trajectory or motion of the camera. The system computes relative motion between consecutive frames to track movement [8]

.

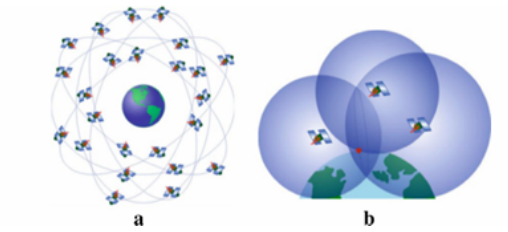
3.1 VO advantages and challenges

VO is an inexpensive and alternative odometry technique that is more accurate than conventional techniques, such as GPS, Inertial Navigation System (INS), wheel odometry, and sonar localization systems, with a relative position error ranging from 0.1% to 2%. This method is characterized by good balance among cost, reliability, and implementation complexity [34].

VO remains robust even in challenging conditions like uneven terrains, where wheel slippage can be an issue and operates independently of external signals, making it highly effective in GPS-denied environments, where GPS is prone to significant errors since the signal is lost. Unlike laser and sonar localization systems, VO does not emit any detectable energy into the environment, allowing it to function discreetly. Additionally, VO experiences less local drift compared to wheel encoders and low-precision INS, and it can be integrated with GPS and INS for enhanced accuracy.



(a) Sonar/ultrasonic sensors [34]



(b) Global positioning system [34]

(c) Wheel odometry with an optical encoder [34]

(d) Monocular camera [34]

Figure 3.2: Different types of localization sensors

Using cameras for robot localization offers significant advantages over other sensors. They reduce costs and allow for seamless integration of ego-motion data with other vision-based algorithms, such as obstacle, pedestrian and lane detection, without requiring calibration between different sensors. Cameras are also small, inexpensive, lightweight, low-powered, and versatile, making them suitable for a wide range of vehicles—whether on land, underwater or in the air—and for various robotic tasks, including object detection and recognition [8].

Robot localization in indoor contexts has been effectively done, however, the issue of robot localization in outdoor settings is still difficult to solve. Localization is challenging in outdoor contexts due to a number of elements, such as the fact

that terrains are typically not flat, direct sunshine, shadows, and dynamic changes in the environment brought on by wind and sunlight.

For VO to work efficiently, sufficient illumination and a static scene with enough texture should be present in the environment to allow apparent motion to be extracted. In areas with smooth and low-textured surfaces, directional sunlight and lighting conditions are critical factors, leading to non-uniform scene lighting. Moreover, shadows from static or dynamic objects or even from the vehicle itself, can disrupt the calculation of pixel displacement, resulting in erroneous displacement estimation [8]. Another challenge for VO is that monocular vision systems, which rely on a single camera, encounters a problem known as scale uncertainty. This issue arises because the system has difficulty determining the true size or distance of objects in the image. When using a monocular camera, there is no direct reference for scale; the system only captures 2D images, lacking depth information that would help determine the actual distances between objects.

3.2 Formulation of the VO Problem

While an agent moves through an environment, it captures images with a rigidly attached camera system at discrete time instants k. In the case of a monocular system, the set of images taken at times k is denoted by $I_{0:n} = \{I_0, \ldots, I_n\}$. For a stereo system, there are left and right images at every time instant denoted by $I_{l,0:n} = \{I_{l,0}, \ldots, I_{l,n}\}$ and $I_{r,0:n} = \{I_{r,0}, \ldots, I_{r,n}\}$, respectively.

Two camera positions at adjacent time instants $k-1$ and k are related by the rigid body transformation $\mathbf{T}_{k,k-1} \in \mathbf{R}^{4x4}$ of the following form:

$$\mathbf{T}_{k,k-1} = \begin{bmatrix} \mathbf{R}_{k,k-1} & \mathbf{t}_{k,k-1} \\ \mathbf{0}^\top & 1 \end{bmatrix}$$

where $\mathbf{R}_{k,k-1} \in SO(3)$ is the rotation matrix, and $\mathbf{t}_{k,k-1} \in \mathbf{R}^{3x1}$ is the translation vector. The set $\{\mathbf{T}_{0,1} \ldots \mathbf{T}_{n,n-1}\}$ contains all subsequent motions. To simplify notation, $\mathbf{T}_k$ will be used instead of $\mathbf{T}_{k,k-1}$. Finally, the set of camera poses $C_{0:n} = \{C_0, \ldots, C_n\}$ contains the transformations of the camera with respect to the initial coordinate frame at $k = 0$.

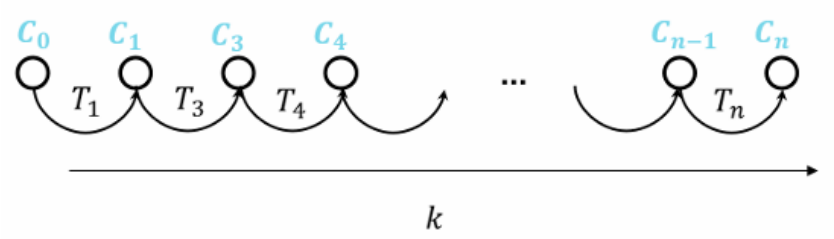


Figure 3.3: Illustration of the global camera path - the camera trajectory C_n is estimated incrementally by concatenating all the relative transformations T_k pose after pose

The current pose $\mathbf{C}_n$ can be computed by concatenating all the transformations $\mathbf{T}_k$ (for $k = 1, \ldots, n$) as follows:

$$\mathbf{C}_n = \mathbf{C}_{k-1} \cdot \mathbf{T}_k$$

where $\mathbf{C}_0$ is the camera pose at the instant $k = 0$, which can be set arbitrarily by the user.

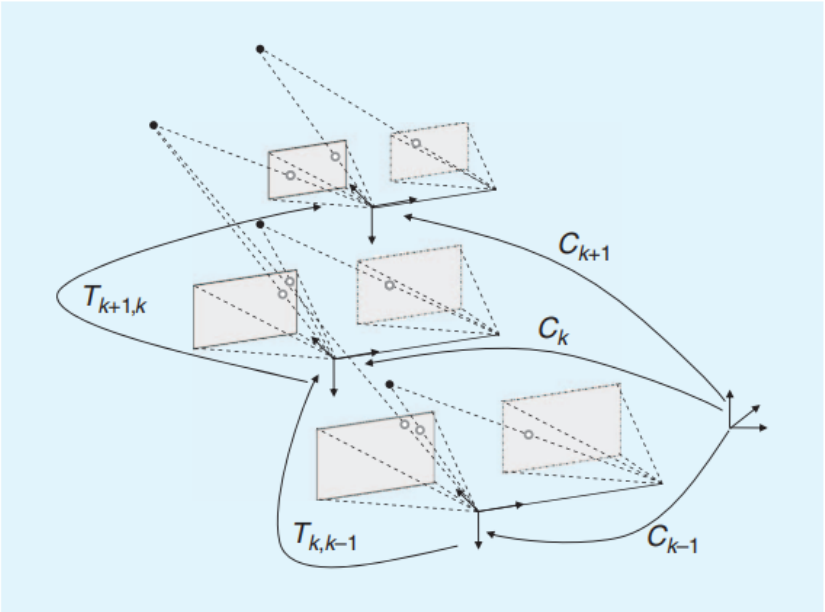


Figure 3.4: An illustration of the visual odometry problem. The relative poses $\mathbf{T}_{k,k-1}$ of adjacent camera positions (or positions of a camera system) are computed from visual features and concatenated to get the absolute poses C_k with respect to the initial coordinate frame at k=0.

3.3 Approaches of VO

The position of a mobile robot with vision-based odometry can generally be estimated in three different ways: through a feature-based approach, an appearance-based approach or a hybrid-based approaches.

3.3.1 Appearance-based methods or direct approach

The appearance-based approach estimates the camera pose by analyzing the intensity of the captured image pixels based on minimizing the photometric error.

Photometric Error

The photometric error quantifies the difference between the intensity values of image pixels in corresponding areas of different images. Specifically, it assesses how well the intensity of pixels in the current image matches the intensity of pixels in the reference image after applying a transformation to align them.

The fundamental assumption in appearance-based methods is that the scene's appearance remains constant over the short time interval between two frames. This means that any change in the observed intensity of a pixel is due to the movement of the camera rather than changes in the scene itself.

When the camera moves, the image of the scene changes, but ideally, the appearance of the same physical point in the scene should be consistent if observed from different viewpoints. Hence, if you have a pixel in one image, its corresponding pixel in the next image should have the same intensity value if the scene has not changed and only the camera has moved. This means that every pixel introduces a constraint in the optimization problem leading to an accurate pose and also a dense map of the environment. Figure 3.5 illustrates the main pipeline of appearance-based VO paradigms.

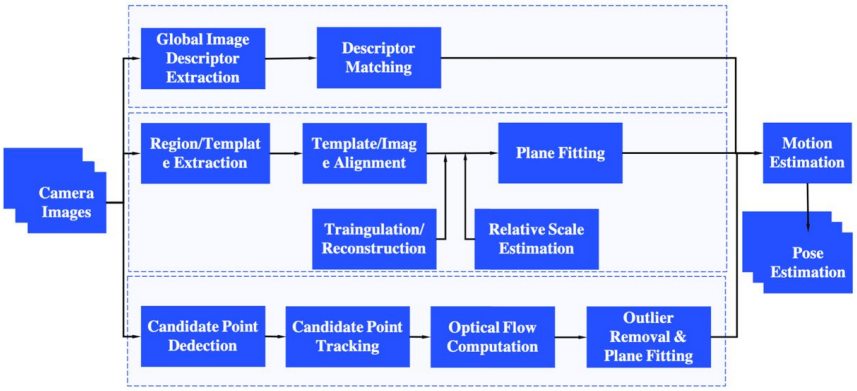


Figure 3.5: Main pipelines of conventional–appearance-based VO technique [1]

The principle of appearance-based VO can be classified into two main approaches:

- **Region/Template matching-based methods:** The motion is estimated by concatenating camera poses through an alignment process for two consecutive images. This technique has been extended by measuring the invariant similarities of local areas and using global constraints.
- **Optical flow-based methods:** In the optical flow-based method, the raw pixel data from consecutive frames are processed using an Optical Flow (OF) algorithm to estimate motion. This algorithm analyzes changes in pixel intensity between two successive frames captured by the camera(s). The core idea is to compute the 2D displacement vectors of points between the frames, which reflect the motion of objects and the camera. As the illumination of pixels changes, the OF algorithm tracks these changes to determine how each pixel's position shifts from one frame to the next, thereby estimating the overall camera motion.

Positive and negative aspects

Since the appearance-based methods work on the assumption that the projection of a point in both frames has the same intensity. This assumption often fails due to

lighting changes, sensor noise, pose errors and dynamic objects. Hence, direct methods require a high frame rate which minimizes the intensity changes, furthermore the light intensity can be handled by normalizing the global intensity. Another issue is high computation due to the use of all pixels over all frames, this problem is addressed by only using pixels with sufficient information, that is, with high intensity gradient. This produces a semi-dense map that mostly contains edges [35].

A positive aspect of this approach is that it uses the complete geometric information from the captured camera frames, considering not only individual pixel values but also their spatial arrangement and relative positions, thus by analyzing the entire geometric layout of the scene, the approach can accurately capture its structure. This comprehensive analysis helps mitigate aliasing issues—situations where similar patterns in the scene lead to ambiguity in feature recognition. Consequently, the appearance-based approach improves both the accuracy of pose estimation and the overall robustness of the system [36].

It is particularly useful in environments where textures are sparse or visibility is poor, in such conditions, traditional methods might struggle because there are fewer distinctive features to track, the use of detailed geometric information helps maintain accurate pose estimation even in less ideal scenarios.

3.3.2 Feature-based methods

Feature-based methods estimate camera motion by detecting and tracking distinctive features or key points across consecutive frames. These features are usually identifiable elements such as corners, edges, or other prominent points within the image. The process begins with feature detection, where algorithms are employed to identify and extract these unique and recognizable points from each frame.

After detecting the features, the next step is to match or track these points across successive frames. This involves establishing correspondences between features in different frames, which allows for tracking how each feature moves over time.

With these correspondences, motion estimation techniques are then used to determine the camera's movement. Methods such as Random Sample Consensus (RANSAC) are applied to handle outliers and ensure robust estimation. Additionally, geometric transformations are utilized to model and compute the motion between frames.

Positive and negative aspects

Feature-based VO techniques provide reliable performance even in environments with significant geometric distortions and varying illumination. This robustness arises because these methods focus on specific, distinctive features within the image—such as corners or edges—rather than relying on the overall appearance of the entire scene.

In complex or distorted environments, these unique features are generally stable and can be consistently tracked despite changes in the scene's geometry. Similarly, for illumination variations, these features tend to remain recognizable even when

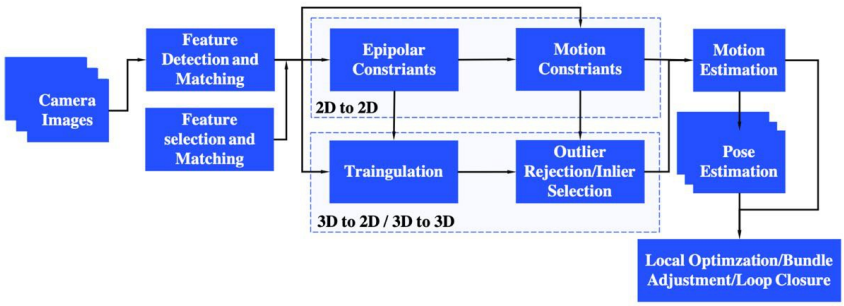


Figure 3.6: Main pipelines of conventional–feature-based VO technique [1]

lighting conditions change. By concentrating on these stable, identifiable features, feature-based VO techniques maintain accuracy and robustness in diverse and challenging conditions.

However, these methods can sometimes discard valuable image data because they focus solely on the key features that are detected. The process of detecting, extracting, and matching these features between frames can be computationally intensive, with the cost increasing proportionally to the number of features extracted. Despite this, a greater number of features generally leads to more accurate pose estimation. To make feature-based VO more practical for resource-constrained platforms, such as UAVs, efforts are made to optimize the process. This involves selecting and maintaining a subset of key features to streamline extraction and matching, thus balancing computational efficiency with the accuracy of pose estimation [2].

3.3.3 Hybrid methods

For low-textured scenarios, feature-based VO schemes are not considered as a robust scheme since only a few features are to be detected and tracked. On the other hand, the appearance-based VO schemes exploit all image information for detecting and matching process between frames, leading to a more efficient outcome at the cost of a considerable computational power.

Thus, hybrid methods have been introduced to combine advantages of the two above mentioned schemes, they merge the tracking of important features across frames with the use of pixel intensity data from the entire or group of images.

3.4 Final considerations on different VO approaches

In urban environments, which are often characterized by complex scenes and dynamic elements, feature-based methods are commonly preferred for VO. This preference is due to several key factors:

- **Partial occlusion handling:** Urban settings frequently include obstructions from buildings, vehicles, and pedestrians. Feature-based methods are particularly adept at managing partial occlusions by focusing on and tracking features that remain visible despite these obstacles.

- **Efficient computation:** These methods typically use efficient algorithms for feature extraction, matching, and motion estimation. This efficiency makes feature-based approaches well-suited for real-time applications in urban environments, where computational resources might be limited.

The following Figure highlights the final choices made so far:

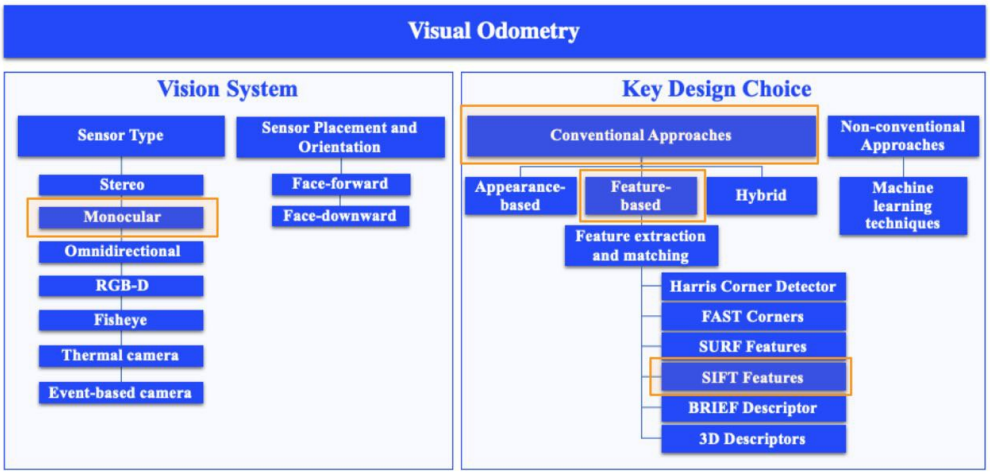


Figure 3.7: General classification of VO techniques [1]

3.5 Monocular Visual Odometry

VO algorithms can be broadly categorized into two types based on the camera setup: stereo cameras and monocular cameras.

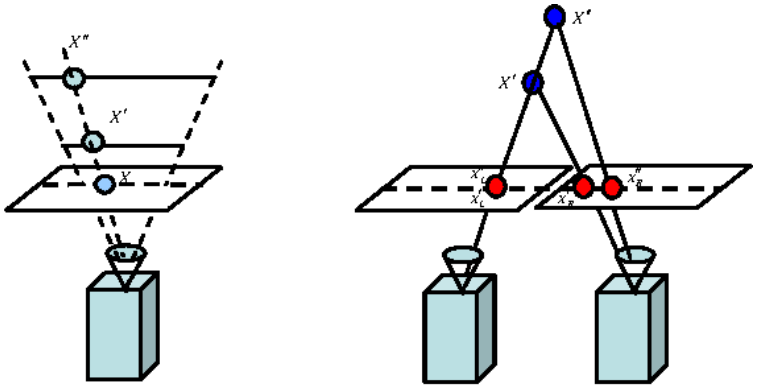


Figure 3.8: Monocular camera and stereo camera [9]

Stereo cameras use two synchronized lenses, similar to the human binocular vision system, to capture images from slightly different perspectives. This setup allows them to directly compute depth information by analyzing the disparity between the two images, making them well-suited for environments where accurate depth perception is crucial.

Stereo cameras can provide accurate depth information in a wide variety of lighting conditions, as they rely on geometric principles rather than light intensity.

However, they require a textured environment to calculate disparity effectively. In texture-less or highly repetitive patterns, stereo cameras might struggle to find correspondences.

Monocular cameras utilize a single lens, capturing only one viewpoint at a time. Since they lack the inherent ability to perceive depth directly, these systems rely on more complex algorithms to estimate depth and motion.

In this thesis, the monocular camera setup has been selected due to its versatility and relevance in various scenarios where stereo vision may become less effective. Specifically, when the distance to the scene is significantly greater than the stereo baseline—the separation between the two cameras in a stereo setup—the advantages of stereo vision diminish, making it difficult to accurately compute depth information. In such situations, the problem effectively reduces to a monocular VO scenario, where only a single camera is utilized to estimate motion [2].

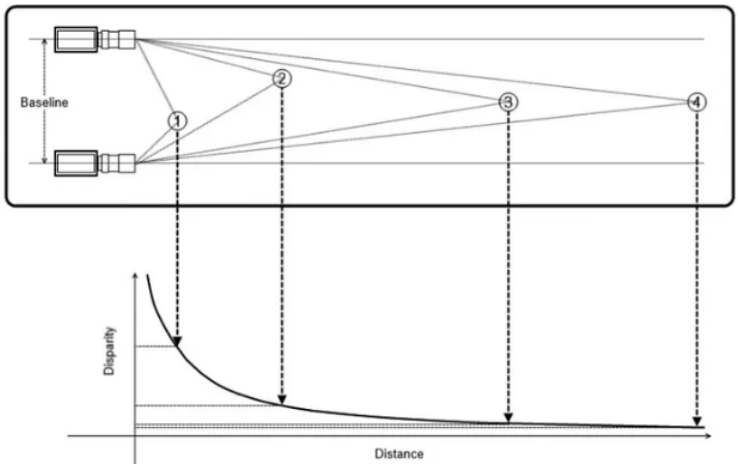


Figure 3.9: Disparity is proportional to baseline. This is easy to visualize. If we have a small baseline distance between the two cameras, then the difference/disparity between the two images is going to be small. As we increase the baseline, the disparity is going to scale up.

Despite the challenges posed by the absence of direct depth information, monocular VO techniques have advanced significantly, employing sophisticated algorithms that leverage temporal image sequences to infer depth and motion.

3.5.1 Challenge in Monocular Visual Odometry: Scale ambiguity

A major problem that affects the accuracy of monocular VO is the scale ambiguity to the loss of one dimension when projecting the three-dimensional world onto a two-dimensional image plane. This limitation can lead to errors in determining the camera's true trajectory, particularly when faced with ambiguous scene geometries, limited visual cues or the inherent constraints of monocular data. These issues can result in scale drift and incorrect trajectory calculations.

Scale

Scale refers to the ratio between the size of the vehicle trajectory estimated from VO and the size of the true trajectory.

Let's look more closely at how the scale ambiguity problem arises. Two images of the same scene are presented. If the entire scene is uniformly scaled up or down by a constant factor, the 2D projections of the points from the scene onto the image, in terms of pixel coordinates, remain exactly the same. This is because the scaling affects both the 3D coordinates of the scene points and the internal parameters of the camera (like focal length and principal point), resulting in proportional changes that cancel out in the image projection.

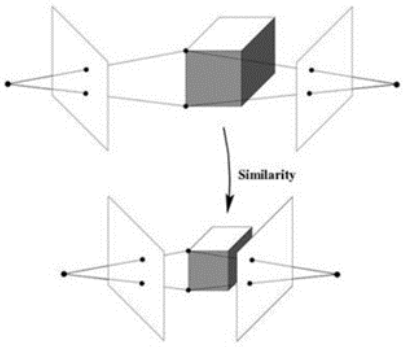


Figure 3.10: Similarity

In monocular vision, it is therefore not possible to recover the absolute scale of the scene. Thus, only 5 DoF are measurable:

- 3 parameters to describe the rotation
- 2 parameters for the translation up to a scale (we can only compute the direction of translation but not its length)

Scale ambiguity can be retrieved by imposing additional information, such as measuring the size of an object in the scene, including known initials, additional constraints, and the addition of other sensors [37]. In the case of VIO techniques, the absolute scale problem is addressed by combining data from the VO algorithm with measurements from the IMU. By integrating acceleration data from the IMU, changes in velocity over time can be tracked. Since acceleration is directly related to velocity, it provides a reference for determining scale, effectively making the previously unknown scale from the monocular camera observable.

Chapter 4

Feature-based approach

For the reasons previously defined in Chapter 3, in this thesis, the relative motion T_k is computed using feature-based methods, which rely solely on salient and repeatable features that are extracted (or tracked) across the images. It focus on the premise that prominent points or regions in each frame can be used to determine camera movement [11]. For this reason, in this section, a detailed description of the feature-based VO approach is provided.

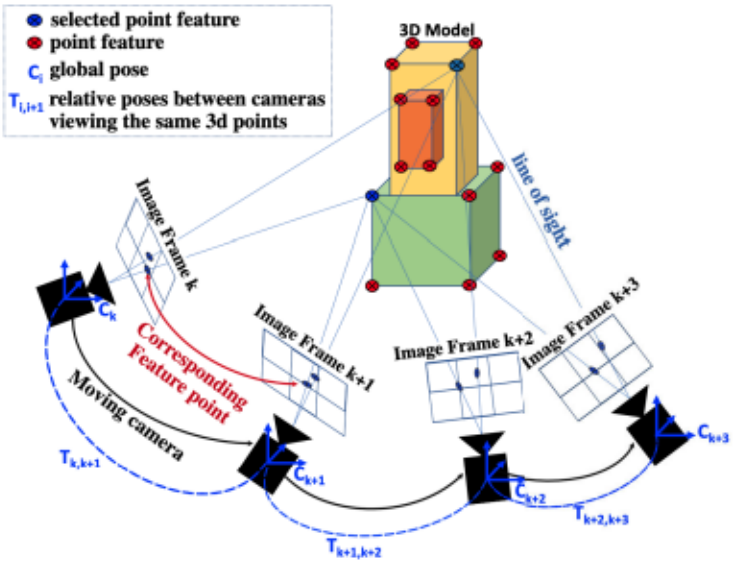


Figure 4.1: Illustration of VO feature based approach scheme [1]

Image Features

Image features are uniquely identifiable areas in the image that are associated with some 3D object in the world, they should be recognizable even as the scene changes. An important role in this first step of the feature-based approach is played by the feature detector and the feature descriptor:

- A feature detector extracts interesting features in the image, such as corners or textured areas.
- Feature descriptors are vectors that are useful to differentiate between two features.

Together, feature detectors and descriptors help in storing the most relevant and unique information about a scene. They are also important for solving the correspondence problem, which involves identifying the same set of features between multiple views of the same scene.

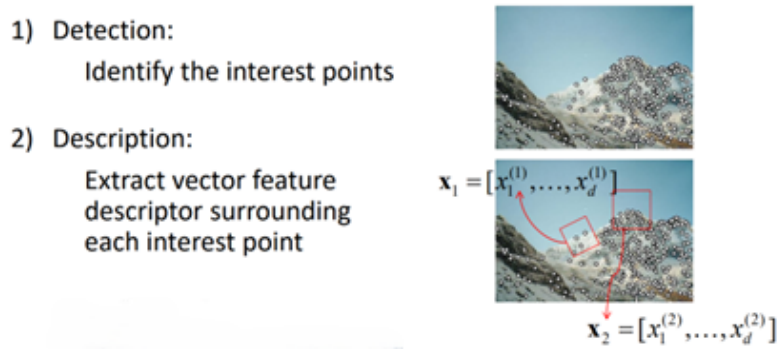


Figure 4.2: Feature detector and feature descriptor [10]

There are two main approaches to find feature points and their correspondences.

- Tracking
- Matching

Tracking

Tracking involves identifying features in one image and then following them in subsequent images using local search techniques, such as correlation.

Initially, features are detected in the first frame and their positions are updated or predicted in the following frames based on their motion and changes in appearance. This process is typically achieved using methods like OF estimation or template matching. The main goal is to maintain the correspondence between feature points across different frames, allowing for the estimation of camera motion over time without the need to re-detect features in each frame.

Figure 4.3: Tracking of features

For successful tracking, three assumptions must be made:

- The displacement of features from one frame to the next is relatively small such that points do not dramatically change position.
- Brightness or color of the features being tracked remains consistent across frames.
- The relative positions of features within a local area are preserved as the scene changes slightly, this means that the spatial coherence is guaranteed

Matching

Matching refers to the process of independently detecting features in all images and then finding correspondences between feature descriptors extracted from different images. Correspondences between descriptors are established based on some similarity metric, such as Euclidean distance or Hamming distance.

For successful matching, two assumptions must be made:

- Features must be easily recognizable and unique enough that they can be distinguished from other features in the image. If features are not distinctive, they may be confused with other features, leading to incorrect matches.

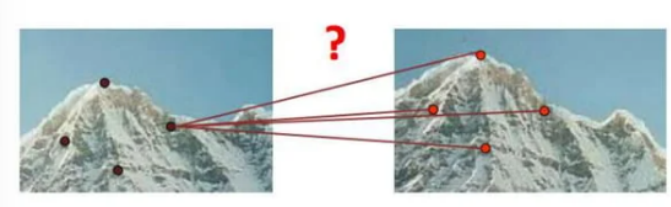


Figure 4.4: A reliable and distinctive feature descriptor is required that is invariant to geometric and illumination changes

- Features should exhibit invariance to factors that can change between images or frames. The most common factors include scale, rotation, illumination and affine transformations

After comparing all feature descriptors between two images, the best correspondence of a feature in the second image is chosen as that with the closest descriptor (in terms of distance or similarity). However, this stage may result in features in the second image matching with more than one feature in the first image.

This problem is solved by performing the mutual consistency check, which consists of 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 cost is quadratic with the number of features. Therefore, when the number of features is large, it is necessary to consider a computationally efficient method. The best approach is constrained matching, which consists of searching for corresponding features in predicted regions of the second frame.

The following table presents a comparison of different tracking and matching techniques.

Tracking	**Matching**
Computationally efficient	Comprehensive coverage: Independent detection of features in all images ensures comprehensive coverage of the scene, capturing a wide range of information.
Robustness: Tracking features maintains consistency and reduces the likelihood of losing track of features due to occlusions or changes in viewpoint.	Flexibility: Each frame is analyzed independently, allowing for adaptability to changes in the scene and variations in lighting conditions.
More suitable when the images are taken from nearby viewpoints.	More suitable when a large motion or viewpoint change is expected, which helps limit motion-drift-related issues.
Limited coverage: Tracking relies on features detected in the initial frame, which may not cover the entire scene or capture all relevant information.	Computationally expensive, particularly for large datasets or real-time applications.
Accumulated errors: Errors in feature tracking can accumulate over time, leading to drift or inaccuracies in pose estimation.	Ambiguity in matching: Matching features based on similarity metrics may lead to ambiguous or incorrect matches, especially in scenes with repetitive patterns or low-texture regions.

Table 4.1: Comparison of Tracking and Matching techniques

4.1 Main pipeline of VO system considering Feature-based Approach

The feature-based VO 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

For each new image I_k, the initial steps involve detecting and matching or tracking 2-D features with those from previous frames. Two-dimensional features that represent the same 3-D object across different frames are known as image correspondences. The next step is to compute the relative motion T_k between the time instants $k-1$ and k. Depending on whether the correspondences are given in three or two dimensions, three different methods can be used to solve this problem. The

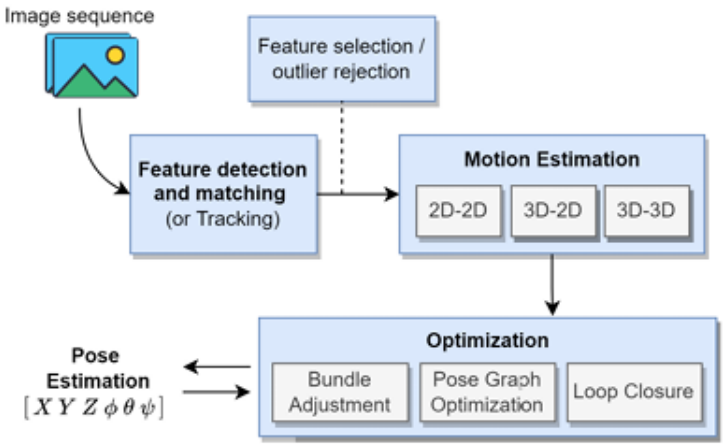


Figure 4.5: Pipeline of feature-based techniques. The final pose estimation is composed of the agents position in space (X, Y, Z) and orientation (roll, pitch, yaw) and can either relate to the previous pose or to a fixed global frame [11]

camera pose C_k is then determined by combining T_k with the previous pose.

To improve the accuracy of the local trajectory estimate, an iterative refinement can be applied to the last m poses. This refinement involves minimizing the sum of squared reprojection errors of the reconstructed 3-D points over the most recent m images. This technique, known as windowed-bundle adjustment, optimizes the trajectory by performing adjustments within a window of m frames [2].

STEP 1: Feature detection

In image processing and computer vision tasks, we need to represent the image by features extracted therefrom. The raw image is perfect for the human eye to extract all information from; however that is not the case with computer algorithms [38].

Feature detection is a crucial step in Computer Vision, where the goal is to identify locations of distinct local features within an image, particularly, the image is examined to find prominent key points that are likely to match or track well in other images. For VO, point detectors, such as corners or blobs, are important because their positions in the image can be measured accurately.

There are three different types of features as shown in Figure 4.24:

- **Corners**: Defined as a point at the intersection of two or more edges where image intensity has a significant variation in all directions.
- **Edges**: Defined as a segment through which a rapid change in the image intensity can be observed.
- **Blobs**: Defined as an image pattern that differs from its neighborhood in terms of intensity, colors, and texture. A blob is a region of an image in which some properties are constant or approximately constant; all the points in a blob can be considered in some sense to be similar to each other.

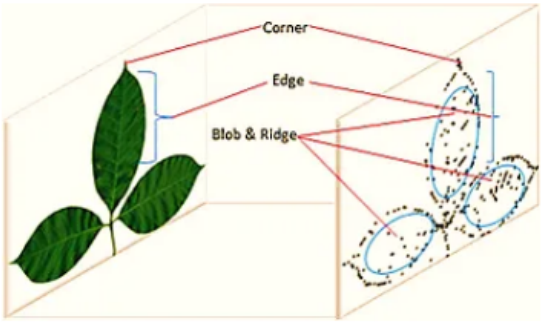


Figure 4.6: Corner, edge and blob [12]

An effective feature detector should exhibit several desirable properties: high localization accuracy (both in terms of position and scale), strong repeatability (detecting many features across different images), computational efficiency, robustness to noise, compression artifacts, and blur, distinctiveness (enabling accurate feature matching across images), and invariance to photometric changes (such as variations in illumination) and geometric transformations (including rotation, scaling, and perspective distortions) [16].

The literature on VO includes a variety of point-feature detectors, each with its own advantages and limitations. Generally, each feature detector operates in two stages. The first stage involves applying a feature-response function to the entire image, this function highlights areas of interest, such as corners or key points, based on local image properties. The second stage applies non-maximum suppression to the output from the first stage. This process aims to identify and save only the local maxima (or minima) of the feature-response function, effectively filtering out weaker responses. The result of this non-maximum suppression process is a set of detected features that are robust and distinctive for further analysis.

STEP 2: Feature descriptor

Once features have been extracted, the next step is to locate these same features in subsequent frames. To find these similarities, we first need to describe these features by generating compact representations of the local image content around each feature point.

Feature description is the process of assigning a numerical fingerprint to a feature, allowing it to be differentiated from others. This fingerprint takes the form of a vector of values that characterizes the image patch surrounding an interest point. It could be as simple as the raw pixel values, or it could be more complicated, such as a histogram of gradient orientations.

An effective descriptor should ideally exhibit several key properties: invariance to translation and rotation, invariance to scale, invariance to changes in illumination and blur, and minimal memory usage. However, meeting all these requirements at once is often challenging, leading to a trade-off between robustness and computational efficiency. However, achieving all these properties simultaneously is often

challenging. Typically, there is a trade-off between the descriptor's robustness and its computational efficiency. While striving for high robustness may involve complex computations and larger memory requirements, optimizing for efficiency can sometimes compromise the descriptor's ability to handle variations in the image. Thus, finding the right balance between these competing demands is a key aspect of developing effective feature descriptors.

The simplest method to define a descriptor involves using the appearance of a feature, or the pixel intensities in a patch surrounding the feature point, with metrics like:

- **Sum of Squared Differences (SSD)** is a similarity measure that uses the squared differences between corresponding pixels in two image [39]. SSD has the following formula:

$$\text{SSD} = \sum_{i=-n}^{n} \sum_{j=-n}^{n} \left(I_1(u_1 + i, v_1 + j) - I_2(u_2 + i, v_2 + j)\right)^2 \tag{4.1}$$

  where I_1 and I_2 are the image patches centered at (u_1, v_1) and (u_2, v_2), respectively.

  In the SSD formula, i and j represent the horizontal and vertical offsets within a window around the center of the image patches. The parameter n defines the half-width of the window, where the total window size is $(2n+1)\times(2n+1)$.

- **Normalized Cross-Correlation (NCC)** is one of the most common and accurate techniques for finding the similarity between two image patches. This technique operates by calculating the sum of the products of the corresponding pixel intensities in the two image patches and then normalizing this sum by a factor related to the pixel intensities. The formula for NCC is as follows:

$$\text{NCC} = \frac{\sum_{i=-n}^{n} \sum_{j=-n}^{n} I_1(u_1 + i, v_1 + j) \cdot I_2(u_2 + i, v_2 + j)}{\sqrt{\sum_{i=-n}^{n} \sum_{j=-n}^{n} I_1(u_1 + i, v_1 + j)^2 \cdot \sum_{i=-n}^{n} \sum_{j=-n}^{n} I_2(u_2 + i, v_2 + j)^2}} \tag{4.2}$$

  The higher the NCC value, the more similarity between the compared image patches [39].

- **Census transform** is more robust alternative, it converts each image patch into a binary vector by comparing the intensity of neighboring pixels with the central pixel. The similarity between patches is then measured using the Hamming distance, making it more resilient to changes in lighting.

Due to its normalization process, NCC generally performs better than SSD in situations where there are variations in brightness levels between the patches. However, NCC is slower because it requires more complex calculations, including multiplications, divisions, and square root operations [39]. However, SSD and NCC are not invariant to transformations such as rotation, scale, or viewpoint changes, limiting their effectiveness to images captured from nearby positions. Appearance-based descriptors, in general, may struggle to distinguish between similar-looking

patches in different areas of an image, reducing their discriminative power in more complex scenarios [16].

4.1.1 Keypoint detectors and descriptors: An overview

The most used and the keypoint feature points detectors and descriptor are going to be explained in the following sections.

Harris Corner Detector

Harris Corner Detection was first introduced by Chris Harris and Mike Stephens in 1988 upon the improvement of Moravec's corner detector. It is a method used in computer vision for detecting corners in images and operates by identifying significant changes in intensity in different directions, which typically occur at corners.

Corners

Corners are the important features in the image, and they are generally termed as interest points which are invariant to translation, rotation, and illumination.

The algorithm calculates a corner response function for each pixel, which measures the amount of variation for shifting a small window in all directions, particularly corners have high values of this response function. In this case, the evaluation of 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. The measurement is done by computing the SSD of the intensity of the pixels before and after the shift using the change function $E(u, v)$:

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

where $I(x, y)$ represents the intensity of the image at point (x, y) and $w(x, y)$ is the window function that assigns weights to the pixels of the window as shown in the Figure 4.7.

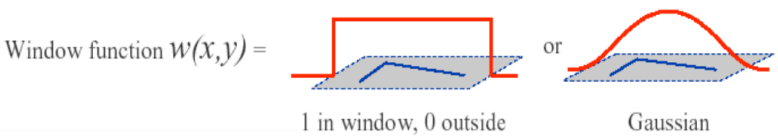


Figure 4.7: Rectangular and Gaussian window functions [13]

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 $E(u, v)$.

Since u and v are small, the shifted intensity $I(x+u, y+v)$ can be approximated by the following first-order Taylor expansion:

$$I(x+u, y+v) \approx I(x,y) + uI_x(x,y) + vI_y(x,y)$$

where I_x and I_y are partial derivatives of I in x and y direction, respectively. By substituting this approximation in the corner function we obtain:

$$E(u,v) = \sum_{x,y} w(x,y) \left[u^2 I_x^2 + 2uvI_xI_y + v_2 I_y^2\right]^2$$

This equation can be expressed in the following matrix form:

$$E(u,v) \approx \begin{bmatrix} u \\ v \end{bmatrix}^T M \begin{bmatrix} u \\ v \end{bmatrix}$$

where the matrix M, called structure tensor, is a 2×2 matrix computed from image derivatives:

$$M = \sum_{(x,y)\in W} w(x,y) \begin{bmatrix} I_xI_x & I_xI_y \\ I_xI_y & I_yI_y \end{bmatrix}$$

We aim for the $E(u,v)$ to be large when the window is shifted in all the directions, or alternatively, for the $E(u,v)$ to not be small in any direction. By solving for the eigenvectors of the matrix M, we can determine the directions that correspond to the largest and smallest increases in $E(u,v)$, since the associated eigenvalues provide the actual magnitudes of these changes. A response score R is then computed for each window based on these values, indicating how corner-like the window is. The response function R is defined as:

$$R = \det(M) - k \cdot \mathrm{tr}(M)^2$$

where:

$$\det(M) = \lambda_1 \lambda_2$$
$$\mathrm{tr}(M) = \lambda_1 + \lambda_2$$

Here, λ_1 and λ_2 are the eigenvalues of the matrix M, which help determine the nature of the region:

- When $|R|$ is small, which occurs when both λ_1 and λ_2 are small, the region is considered **flat**.
- When $R < 0$, which happens if one eigenvalue is significantly larger than the other (i.e., $\lambda_1 \gg \lambda_2$ or $\lambda_2 \gg \lambda_1$), the region is classified as an **edge**.
- When R is large, which occurs when both eigenvalues are large and approximately equal (i.e., $\lambda_1 \approx \lambda_2$), the region is identified as a **corner**.

These relationships can be visually represented in a diagram as follows:

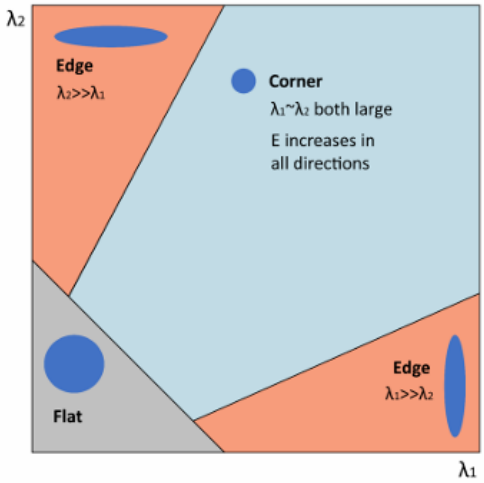


Figure 4.8: Harris regions - Classification of image points using eigenvalues of M

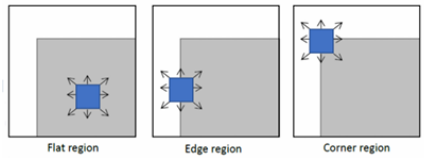


Figure 4.9: Representation of the Harris window on different regions

The following diagram will illustrate the key steps involved in the Harris corner algorithm:

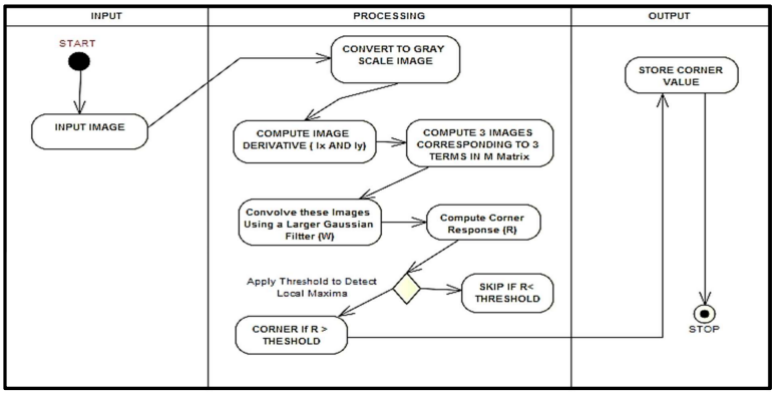


Figure 4.10: Activity diagram of Harris corner detection algorithm [14]

ORB

Oriented FAST and Rotated BRIEF (ORB) is an algorithm that was presented by Ethan Rublee, Vincent Rabaud, Kurt Konolige and Gary R. Bradski in their paper *'ORB: An efficient alternative to SIFT or SURF'* in 2011.

It is an advanced feature detection algorithm that enhances both the Features from Accelerated Segment Test (FAST) key point detector and the Binary Robust Invariant Scalable Keypoints (BRIEF) descriptor with several important modifications. Initially, ORB uses the FAST algorithm to quickly identify key points in an image by comparing the brightness levels in a given pixel area. To refine these key points, it applies a Harris corner measure to select the top N points, ensuring the most significant features are retained after that it is applied scaled pyramid to account for scale. Since FAST does not compute key point orientation, ORB calculates the orientation by finding the intensity-weighted centroid of the image patch centered on each key point. The direction from the key point to this centroid provides the orientation and moments are used to improve rotation invariance. Additionally, to address the BRIEF descriptor's poor performance under in-plane rotation, ORB introduces a rotation matrix based on the key point orientation that allows BRIEF descriptors to be adjusted or "steered" according to the key point's orientation, thus enhancing their robustness to rotation. Through these innovations, ORB combines the speed of FAST with the improved rotation invariance of BRIEF, making it a powerful tool for feature detection and description in computer vision [40].

The ORB image matching algorithm is generally divided into three steps: feature point extraction, generating feature point descriptors and feature point matching. The specific flow chart is shown in Figure 4.11

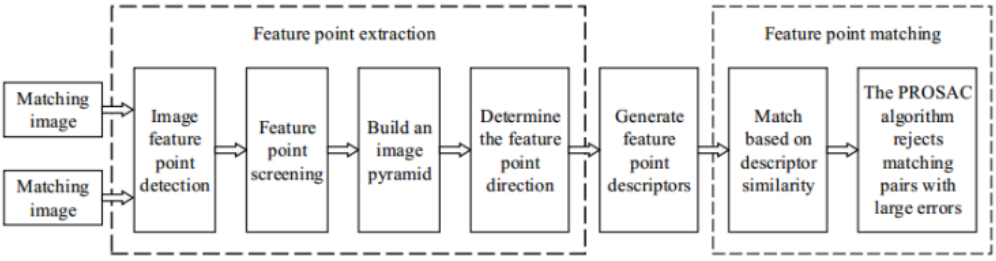


Figure 4.11: Image matching flow chart based on ORB algorithm [15]

ORB offers several advantages that make it highly effective for various computer vision applications. Its speed is a significant benefit, as ORB is designed to be very fast, making it suitable for real-time applications where processing speed is crucial. Additionally, SURF provides scale invariance, enabling it to detect features at different scales within an image, which enhances its robustness to changes in the size of objects or scenes. It also has the ability to be invariant to rotation, allowing it to detect and match features regardless of their orientation, thereby making it robust to changes in viewpoint. Furthermore, ORB is not patented, unlike SIFT or SURF, which means it can be used commercially without incurring licensing fees.

However, ORB has some drawbacks. One issue is its memory consumption; although it is faster than some alternatives, it may still require a considerable amount of memory. Additionally, ORB can exhibit limited distinctiveness, struggling to differentiate between similar-looking features, especially in scenes with repetitive patterns or textureless regions. These limitations can affect its performance in certain challenging environments.

SIFT

SIFT, introduced by David G. Lowe in 1999, is a technique used for detecting and describing features in images. The features it identifies are local, based on the appearance of an object at key points of interest, and are invariant to changes in scale and rotation. Additionally, SIFT is highly robust to variations in illumination, noise, and small changes in viewpoint, making it effective in a wide range of conditions.

The steps involved in SIFT, defined in [41], are:

1. **Scale-space extrema detection**: This step identifies interest points, or keypoints, by detecting extrema in the scale space.

2. **Keypoint localization**: After detecting numerous keypoint candidates, some may be unstable. To refine their positions, a more precise fitting is applied to the nearby data, determining accurate location, scale, and principal curvature ratios. This process helps discard points with low contrast (which are sensitive to noise) or those poorly localized along edges.

3. **Orientation assignment**: Each keypoint is assigned one or more orientations based on the directions of local image gradients. This step is crucial for achieving rotation invariance, as the keypoint descriptor is calculated relative to these orientations, ensuring robustness to image rotation.

4. **Keypoint descriptor**: In this final step, a histogram of local oriented gradients is generated around each keypoint. These gradients are stored in a 128-dimensional vector, forming highly distinctive descriptors for each keypoint.

The following diagram will illustrate the key steps involved in the SIFT algorithm in 4.12.

4.1.2 Characteristics of different feature detectors and descriptors

Each detector has its own pros and cons, they are summarized in Table [2].

The choice of a feature detector should be carefully evaluated based on computational constraints, real-time requirements, the environment, and the motion baseline of the images. While SIFT, SURF, and CENsus-based SUbmap REfinement (CENSURE) are not truly affine invariant—meaning they do not consistently

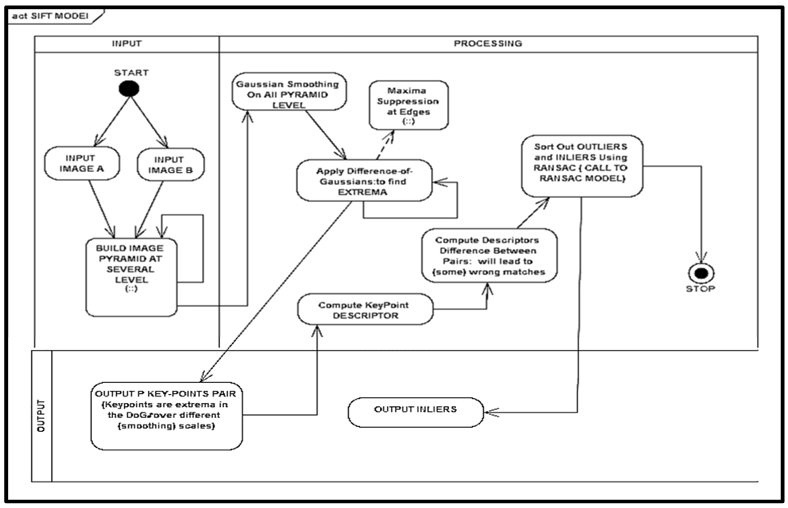


Figure 4.12: Activity diagram of Scale Invariant Feature Transform (SIFT) detector [14]

	Corner Detector	Blob Detector	Rotation Invariant	Scale Invariant	Affine Invariant	Repeatability	Localization Accuracy	Robustness	Efficiency
Haris	x		x			+++	+++	++	++
Shi-Tomasi	x		x			+++	+++	++	++
FAST	x		x	x		++	++	++	++++
SIFT		x	x	x	x	+++	++	+++	+
SURF		x	x	x	x	+++	++	++	++
CENSURE		x	x	x	x	+++	++	+++	+++

Figure 4.13: Comparison of feature detectors: properties and performance [16]

detect the same features under all affine transformations—they demonstrate empirical invariance to certain viewpoint changes. Affine invariance would ideally ensure consistent feature detection across various transformations like translation, rotation, scaling, and skewing.

Among the tested detectors, ORB is noted for its speed, whereas SIFT excels in performance across most scenarios. However, ORB and SURF outperform SIFT in cases where the rotation angle is close to 90 degrees. Additionally, in noisy images, ORB and SIFT exhibit similar performance levels. In ORB, features are typically concentrated in the image's center, whereas SIFT, SURF, and FAST distribute keypoints more evenly across the image. This comparison highlights the strengths and weaknesses of each detector, emphasizing the need to select the appropriate algorithm based on specific application requirements and conditions.

The detectors described above in the table can be classified into two main classes,

blob detector and corner detector. Blob detectors are a class of feature detection algorithms in computer vision designed to identify regions within an image that share similar intensity or color characteristics. These detectors are generally more distinctive than corner detectors but tend to be slower and less localized in scale. For instance, while corner detectors offer rapid computation and precise localization in image position, they struggle with scale changes compared to blob detectors. This makes blob detectors more suitable for scenarios with significant scale and viewpoint changes. However, in environments rich of corners, such as urban settings, blob detectors might not be ideal, as seen with SIFT's automatic disregard for such corners.

4.1.3 Performance of feature detection algorithms

The selection of an appropriate feature detector plays a crucial role in the overall performance of a system. Therefore, it is essential to analyze and compare different detectors to determine which one is best suited for a specific task. To choose the right one, the performance of feature detection algorithms must be evaluated based on criteria that measure both their speed and accuracy across different frames and transformations. Below, we outline several key estimation criteria that help assess the efficiency and effectiveness of these algorithms [17]:

- **Speed per frame** – absolute total time in milliseconds spent on the feature detection of a single frame.
- **Speed per keypoint** – detection time for a single keypoint. Evaluated as the total time divided by the number of detected keypoints. This helps estimate the computational cost of detecting each keypoint.
- **Percent of tracked features** – the percentage of successfully tracked features from the original to the transformed image. Ideally, this value should be close to 100%.
- **Average tracking error** – the average distance between the position of the tracked feature and its calculated position on the transformed frame. This value indicates the accuracy of feature detection. Large values suggest a high number of false positives or "drift" of feature points between frames.
- **Features count deviation** – the difference between the number of keypoints in the reference frame and the number of detected keypoints in the transformed frame, divided by the number of keypoints in the reference frame. This helps estimate how slight changes in exposure affect feature detection.
- **Average detection error** – the average distance between the nearest keypoints in the original and transformed frames.

Eugene Khvedchenya, the author of *Comparison of the OpenCV's Feature Detection Algorithms – II*, worked with a set of images and, for each one, obtained five measurements for each detection algorithm. He then calculated the average for each type of measurement and here are the results:

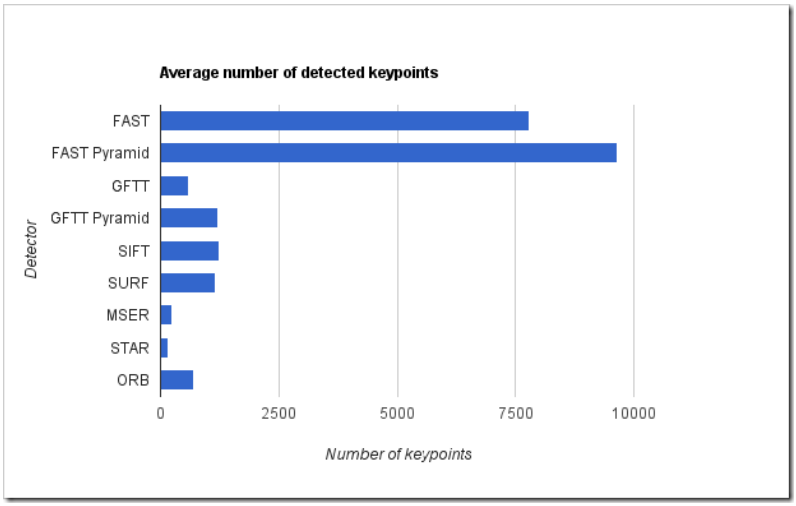


Figure 4.14: Average number of detected keypoints [17]

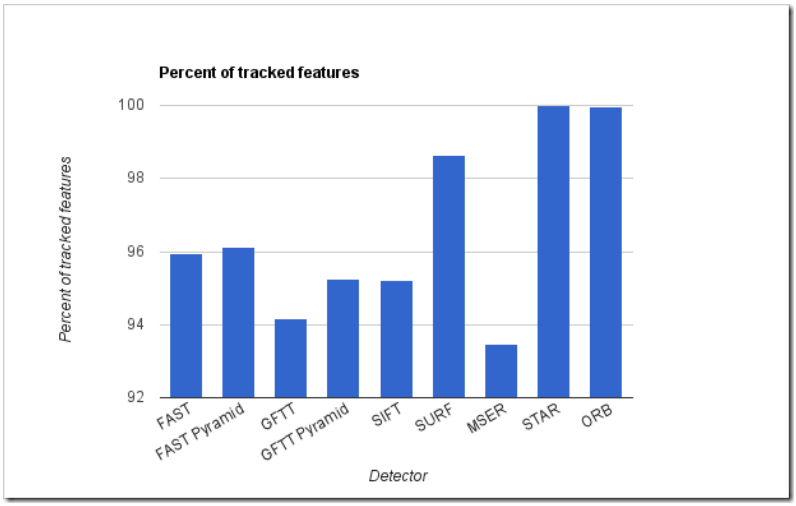


Figure 4.15: Percentage of tracked features [17]

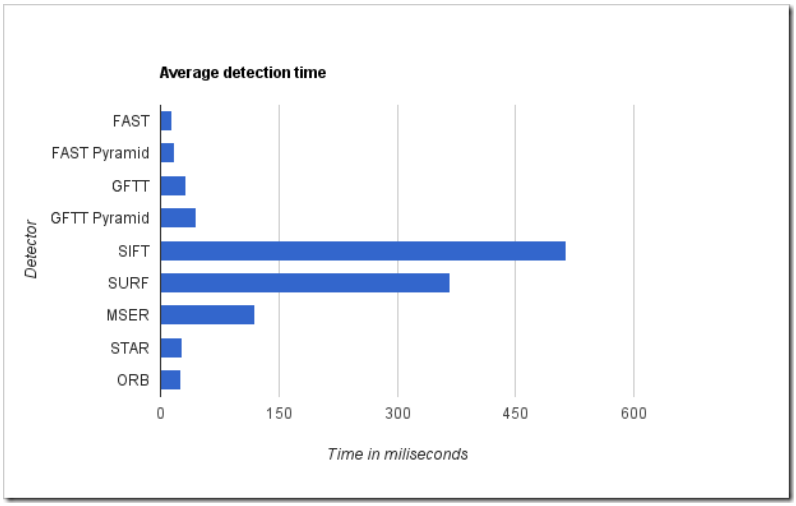


Figure 4.16: Average detection time [17]

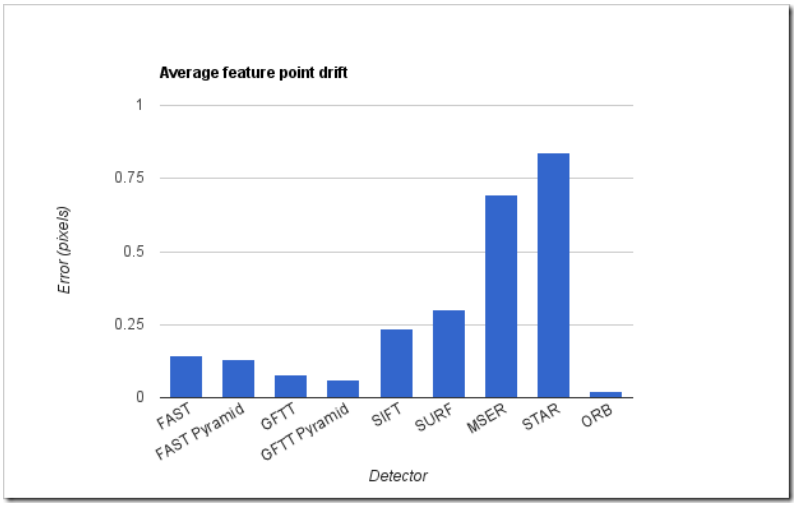


Figure 4.17: Average feature point drift [17]

As it is possible to see in Figure 4.14, FAST detector detects a large number of feature points, but their quality is typically lower. By adjusting the detection threshold, the number of detected points can be controlled. Other detectors, though they detect fewer points, offer higher quality features. The ORB detector consistently finds exactly 702 features per image, indicating a fixed maximum.

In terms of tracked features, Figure 4.15 demonstrates that the best tracking results were achieved using the SURF, STAR, and ORB detectors.

Regarding the detection time, SIFT and SURF are slower, while ORB is notably faster, taking approximately 25 ms for a 512x512 image. Despite its speed, ORB still calculates feature orientation efficiently. The results are illustrated in Figure 4.16.

Finally, tracking quality is estimated by measuring the distance between the actual position of tracked points and the expected position precalculated earlier. From Figure 4.17 ORB shows the smallest drift, which helps decrease systematic error when tracking a long image sequence.

STEP 3: Motion Estimation

Motion estimation is the core computational step performed for every image in a VO system. More precisely, in the motion estimation step, the camera motion between the current image and the previous image is computed, by concatenating all these single movements, the full trajectory of the camera and the agent can be recovered.

Figure 4.18: Motion estimation [18]

There are three standard VO motion estimation methods, which are categorized into 2D-to-2D, 3D-to-2D, and 3D-to-3D motion estimation techniques. These

methods are used to compute the transformation matrix between two consecutive images (the current and previous image). They depend on the captured features $\mathbf{f}$ and their correspondences, whether specified in 2D or 3D.

- **2D-to-2D:** Both $\mathbf{f}_i$ and $\mathbf{f}_{i-1}$ are specified in 2D image coordinates.
- **3D-to-3D:** Both $\mathbf{f}_i$ and $\mathbf{f}_{i-1}$ are specified in 3D.
- **3D-to-2D:** $\mathbf{f}_{i-1}$ are specified in 3D and $\mathbf{f}_i$ are their corresponding 2D reprojections on the image I_k.

3D-to-3D Motion Estimation

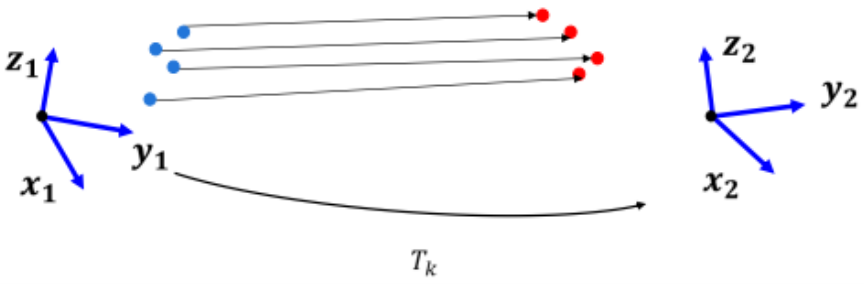


Figure 4.19: Illustration 3D-3D Motion estimation [18]

The 3D-to-3D motion estimation is carried out using corresponding 3D features, which are derived by triangulating 3D points from 2D image correspondences at each time instant. The camera motion is computed by determining transformation - rotation and translation - that aligns the set of 3D points at the earlier time instant (t_{k-1}) with the set of 3D points at the current time instant (t_k).

The transformation is estimated by minimizing the 3D Euclidean distance between the corresponding 3D features. The goal is to find the rotation and translation that make the points from the previous frame match as closely as possible with the points in the current frame, as shown in the following equation:

$$T_k = \min_{\mathbf{T_k}} \sum_i \|\mathbf{X}_k - \mathbf{T}\mathbf{X}^i_{k-1}\|$$

where $\mathbf{X}_k$ and $\mathbf{X}_{k-1}$ are the corresponding 3D-to-3D feature points and k is the minimum number of feature points required to constrain the transformation.

The minimum number of feature pairs, needed for the motion estimation, is chosen according to the system's DoF and the model used for motion estimation. As shown in [42], the minimal case solution involves three 3-D-to-3-D non-collinear correspondences, which can be used for robust estimation in the presence of outliers. Obviously, the use of more points implies greater accuracy but also a higher computational cost.

Furthermore, the computed transformations have absolute scale, this means that the transformations include information about the actual distances and sizes

in the real world, thus, the trajectory of a sequence can be computed by directly concatenating the transformations.

The algorithm for the 3D-to-3D motion estimation is described as follows [2]:

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 step 1

Table 4.2: 3-D-to-3-D Motion Estimation Algorithm

3D-to-2D Motion Estimation

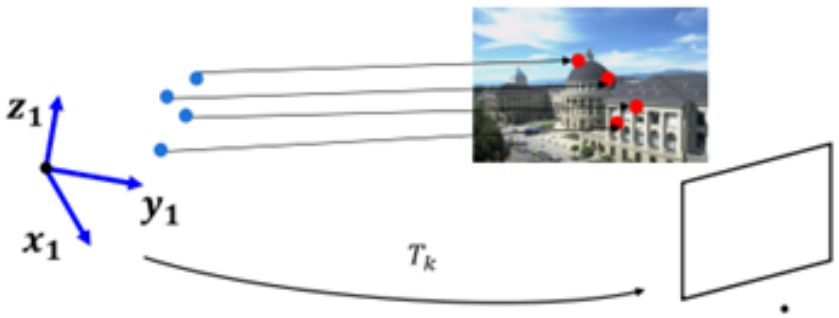


Figure 4.20: Illustration 3D-2D Motion estimation [18]

The transformation $\mathbf{T_k}$ is computed from the 3D-to-2D correspondences $\mathbf{X}_{k-1}$, feature in the 3D world and $\mathbf{p}_{k-1}$ 2D feature on the image plane, this problem is known as the Perspective-n-Point (PnP) problem.

This method is similar to the previous approach but here the 2D reprojection error is minimized to find the required transformation $\mathbf{T_k}$. The cost function for this method is as follows:

$$\mathbf{T_k} = \min_{\mathbf{T_k}} \sum_i \|\mathbf{p}_k^i - \mathbf{p}_{k-1}^i|^2$$

where $\mathbf{p}_k^i$ is the observed feature point in the current frame I_k, and $\mathbf{p}_{k-1}^i$ is the reprojection of the 3D point $\mathbf{X}_{k-1}^i$ into the frame I_k after applying the transformation $\mathbf{T_k}$.

The Perspective-3-Point (P3P) problem is the minimal case solution for estimating the camera's pose from 3D points and their 2D projections. It typically results in four potential solutions and in order to determine the correct pose, additional information or points are used to disambiguate among them. In the following, the algorithm for the 3D-to-2D motion estimation, presented in [2], 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 step 2.1

Table 4.3: 3-D-to-2-D Motion Estimation Algorithm

In the monocular case, the 3-D feature point $\mathbf{X}_{k-1}$ can be triangulated starting from the seats of features $\mathbf{p}_{i-1}$ and $\mathbf{p}_{i-2}$, however, in this case image correspondences must be triangulated across three views (I_{k-2}, I_{k-1}, and I_k)[2].

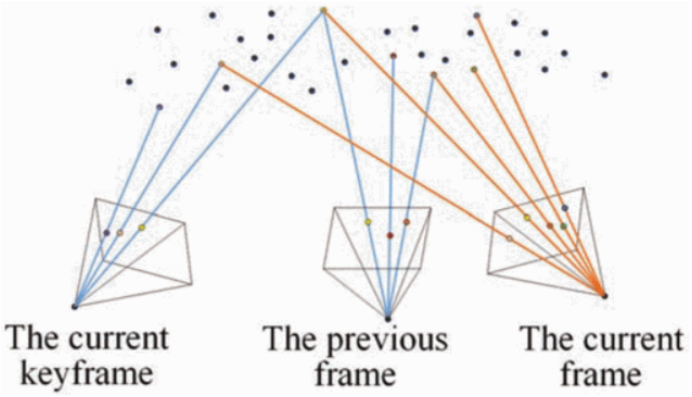


Figure 4.21: Monocular setup 3D-2D Motion estimation [19]

Initial setup with two views : To start, capture two consecutive images from the monocular camera, denoted as I_{k-2} and I_{k-1}. In these frames, detect features using feature detectors like SIFT, ORB, etc., and match the detected features between the two frames to find correspondences. Next, estimate the initial pose and triangulate points. Compute the fundamental matrix $\mathbf{F}$ from the matched points. If the camera intrinsics are known, use the fundamental matrix to compute the essential matrix $\mathbf{E}$. Decompose the essential matrix to obtain the initial rotation $\mathbf{R}$

and translation $\mathbf{t}$ between the two frames. Using these matched feature points and the initial poses, triangulate the 3D points to create an initial 3D point cloud.

Iterative process for subsequent frames : The process alternates between pose estimation and triangulation for each new frame, while ensuring features are matched over at least three frames.

First, a new frame I_k is captured and features are detected. These features are then matched with those in the previous frame I_{k-1} to establish correspondences.

Subsequently, using the correspondences, the camera pose (rotation $\mathbf{R}_k$ and translation $\mathbf{t}_k$) is computed with the PnP algorithm. This step, using the previously triangulated 3D points and their 2D projections in the new frame I_k, estimates the camera pose for the new frame, resulting in the rotation $\mathbf{R}_k$ and translation $\mathbf{t}_k$.

Then, features are matched between the new frame I_k and the previous frame I_{k-1}, and the matched features are triangulated to obtain new 3D points. The 3D point cloud is updated with these newly triangulated points. This iterative process continues for each subsequent frame, ensuring a consistent 3D map and accurate tracking of camera motion.

2D-to-2D Motion Estimation

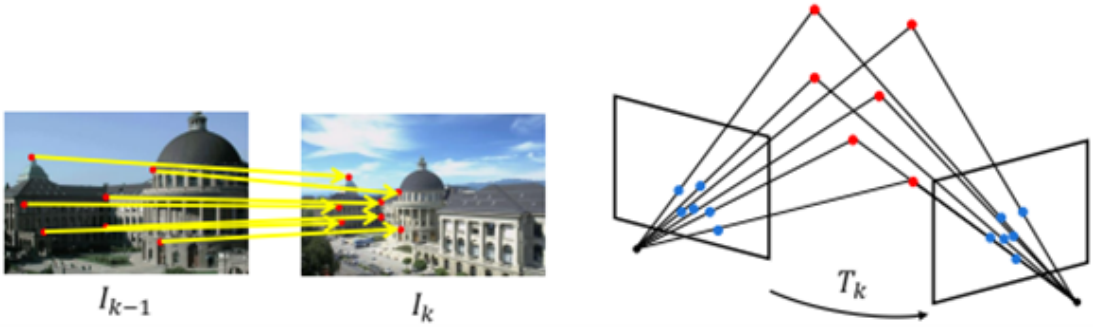


Figure 4.22: 2D-2D Motion estimation [18]

When 3D data is not available, such as in the initial frames of a monocular setup where 3D points have not yet been triangulated, the alternative method for the computation of the camera motion is **2D-to-2D motion estimation**. This approach, which is based on the epipolar geometry shown in Figure 4.23, uses the epipolar constraint for the computation of the transformation $\mathbf{T}_k$ between two consecutive camera frames.

In the context of calibrated cameras, which is the case considered in this thesis, the epipolar constraint can be expressed through the fundamental matrix $\mathbf{F}$. It relates corresponding points between two images through the equation:

$$\mathbf{p}_i'^T \mathbf{F} \mathbf{p}_i = 0$$

where $\mathbf{p}_i$ and $\mathbf{p}_i'$ are the feature correspondences in two consecutive frames, and $\mathbf{F}$ is the fundamental matrix, expressed by the relationship:

$$\mathbf{F} = \mathbf{K}^{-T} \mathbf{E} \mathbf{K}^{-1}$$

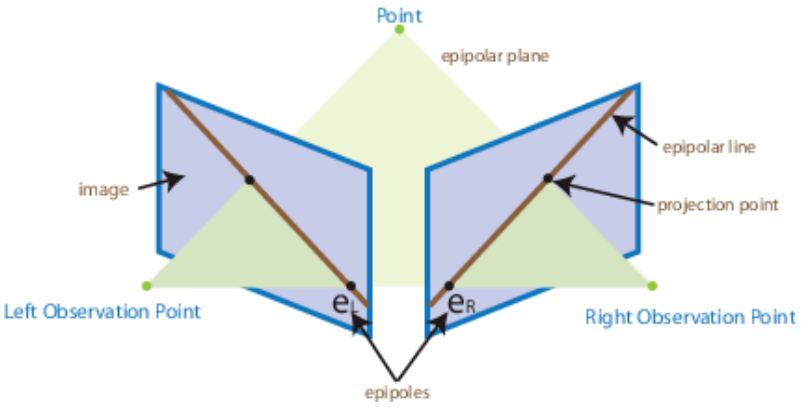


Figure 4.23: Epipolar geometry

The fundamental matrix $\mathbf{F}$ contains the same information as the essential matrix $\mathbf{E}$, in addition to information about the intrinsic parameters. If we are using rectified images and normalize the points by dividing by the focal lengths, $\mathbf{F} = \mathbf{E}$. The essential matrix $\mathbf{E}$ is represented as:

$$\mathbf{E} = \mathbf{R}[\mathbf{t}]_\times$$

where $\mathbf{R}$ is the rotation matrix, and $[\mathbf{t}]_\times$ is the skew-symmetric translation matrix, given by:

$$[\mathbf{t}]_\times = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}$$

In the following, the algorithm for the 2D-to-2D motion estimation, presented in [2], 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 $\mathbf{R}_k$ and $\mathbf{t}_k$ from the essential matrix
5: Compute the relative scale to rescale $\mathbf{t}_k$
6: Create $\mathbf{T}_k$
7: Estimate the current pose by computing $C_k = C_{k-1}\mathbf{T}_k$
8: Repeat the entire procedure from step 1

Table 4.4: 2-D-to-2-D Motion Estimation Algorithm

Let's analyze more in detail the concept of epipolar constraint and the main steps of the 2D to 2D motion estimation algorithm.

To begin the process, a new image is captured, denoted as I_k. Once this image is obtained, the next step involves extracting and matching features between this newly captured frame I_k and the previous frame I_{k-1}. The geometric relations between two images I_{k-1} and I_k of a calibrated camera are described by the so-called

essential matrix **E**.

Essential matrix

The first main step of the algorithm is the computation of the essential matrix, which can be computed from 2D-to-2D feature correspondences by solving the epipolar constraint using Singular Value Decomposition (SVD). On the base of the SVD the essential matrix E can be rewritten as:

$$E = \mathrm{U}\Sigma\mathrm{V}^T$$

It must have two singular values which are equal and another which is zero:

$$\Sigma = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

The minimal solution for the computation of **E** involves five 2D-to-2D point correspondences and an efficient implementation is proposed by Nistér.

Nistér five-point algorithm

The Nistér five-point algorithm uses a set of five matched points and on the base of the epipolar constraint define the geometric relation between two sequential images.

Extracting R and t from E

The second step is the decomposition of the essential matrix into the rotation and translation matrices, using SVD.

Since both **E** and $-\mathbf{E}$ satisfy the same set of epipolar constraints, they generally give rise to $2 \times 2 = 4$ possible solutions for $(\mathbf{R}, \mathbf{t})$. However, this does not pose a potential problem because only one of them guarantees that the depths of the 3D points being observed by the camera are all positive. Generally, three out of the four solutions will be physically impossible and hence may be discarded.

The four solutions illustrated in Figure 4.24 are the following :

$$(\mathbf{R}, \mathbf{t}),\ (\mathbf{R}, -\mathbf{t}),\ (-\mathbf{R}, \mathbf{t}),\ (-\mathbf{R}, -\mathbf{t})$$

where R_k and t_k are defined as:

$$\mathbf{R}_k = \mathbf{U}(\pm\mathbf{W}^T)\mathbf{V}^T$$

$$\hat{\mathbf{t}}_k = \mathbf{U}(\pm\mathbf{W})\mathbf{S}\mathbf{U}^T$$

while W has the following structure:

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

Disambiguating the correct solution

To identify the correct solution, it is triangulated a single point that is visible in both camera views. For each of the four solutions, triangulation is performed to get the 3D point's position. The correct solution is the one where the triangulated point lies in front of both cameras. In other words, the point should have a positive depth (z-coordinate) in the coordinate system of both cameras.

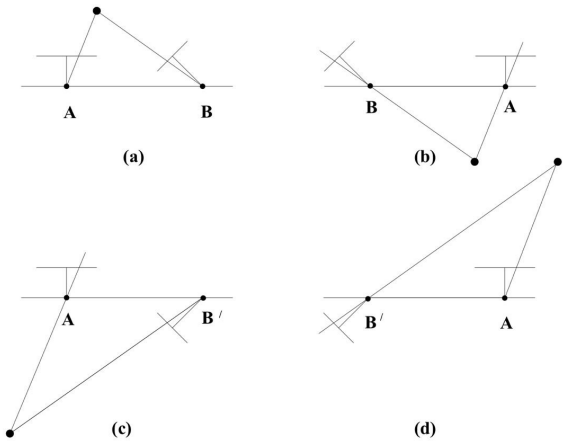


Figure 4.24: The four possible solutions for calibrated reconstruction from E. Between the left and right sides there is a baseline reversal. Between the top and bottom rows camera B rotates 180° about the baseline. Note, only in (a) is the reconstructed point in front of both cameras.

Non-linear optimization

Once the correct solution is identified, the final step is to refine the rotation $\mathbf{R}$ and translation $\mathbf{t}$ matrices using non-linear optimization. The objective of this optimization is to minimize the reprojection error, which measures how closely the projected 3D points, when mapped back to the 2D image plane using the estimated camera parameters, align with the observed 2D points.

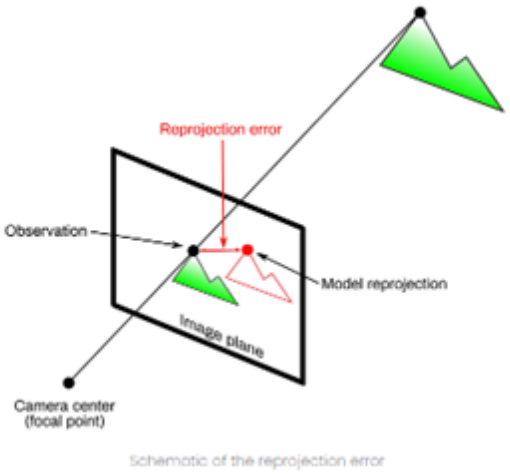


Figure 4.25: Reprojection error [20]

The reprojection error shown in Figure 4.25 is quantified by the difference between the position of the projected 3D point in the image and the actual observed 2D point. Mathematically, this error can be expressed as:

$$T_k = \begin{pmatrix} \mathbf{R}_k & \mathbf{t}_k \\ \mathbf{0} & 1 \end{pmatrix} = \arg\min_{\mathbf{X}_i, \mathbf{C}_k} \sum_{i,k} \|\mathbf{p}_k^i - g(\mathbf{X}_i, \mathbf{C}_k)\|^2$$

where $\mathbf{p}_k^i$ is the observed 2D point and the other term is the projection of the 3D points $\mathbf{X}_i$ through the camera model $g(\mathbf{X}_i, \mathbf{C}_k)$.

The algorithm iteratively adjusts the rotation $\mathbf{R}$ and translation $\mathbf{t}$ parameters to find the values that minimize this cost function. The output of the optimization process is the refined rotation and translation matrices that best fit the observed image points.

Computing the relative scale

At this point, to recover the trajectory of an image sequence, the different transformations must be concatenated. However, the absolute scale of the translation cannot be estimated from only two frames, so the relative scales for the subsequent transformations must be computed.

After triangulating a pair of 3D points $\mathbf{X}_{k-1}$ and $\mathbf{X}_k$ from consecutive images, the relative distances between the two 3D points can be computed. The correct scale is determined from the distance ratio r between a pair of points $\mathbf{X}_{k-1}$ in one image and $\mathbf{X}_k$ in another:

$$r = \frac{\|\mathbf{X}_{k-1,i} - \mathbf{X}_{k-1,j}\|}{\|\mathbf{X}_{k,i} - \mathbf{X}_{k,j}\|}$$

A good practical approach is to use the mean value of the scale ratios computed for multiple point pairs. The translation component of the essential matrix is then multiplied by this scale factor at each iteration.

4.1.4 Considerations about motion estimation methods

As highlighted by Scaramuzza in [2], the 2-D-to-2-D and 3-D-to-2-D approaches offer advantages over the 3-D-to-3-D method for motion computation. Nister in [43] demonstrates this by comparing the VO performance between the 3-D-to-3-D and 3-D-to-2-D methods, using a stereo camera system, showing that the former outperforms the latter. This difference arises because triangulated 3-D points tend to be less accurate in the depth direction, consequently when using 3-D-to-3-D feature correspondences for motion computation, this uncertainty can severely impact the motion estimate. For these reasons,in the 3-D-to-3-D method, the error in 3-D positioning is minimized, whereas in the 3-D-to-2-D method, the focus is on minimizing the image reprojection error.

Another aspect to consider is related to the fact that motion estimation methods in VO involve the triangulation of 3D points from image correspondences, typically from at least two frames. However, real-world conditions such as image noise, camera model inaccuracies, and feature matching uncertainties mean that back-projected rays from these correspondences rarely intersect perfectly. Consequently,

the point at the minimal distance from these rays is considered, leading to uncertainties in the triangulated 3D points.

The triangulated 3-D points show a greater uncertainty if the images are captured at nearby positions compared to the scene distance.

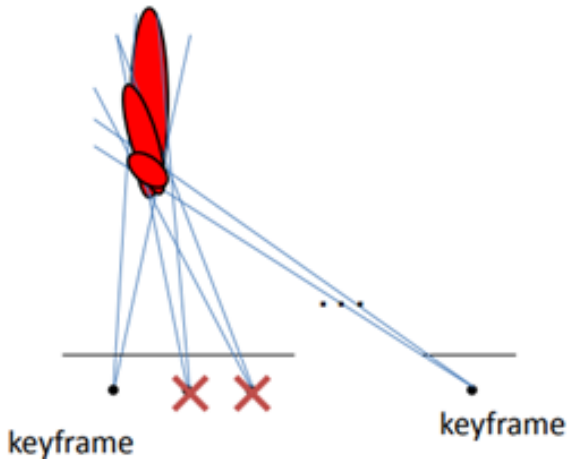


Figure 4.26: Error introduced by triangulation [18]

This uncertainty makes 3D-to-3D methods, which minimize the 3D position error, less accurate compared to 2D-to-2D and 3D-to-2D methods. The 3D-to-2D approach is generally preferred due to its higher accuracy, as it minimizes the image reprojection error instead.

Keyframe selection, an essential process where some frames are skipped until the 3D point uncertainty falls below a threshold, is critical to improving accuracy in 3D-to-3D methods. This important process, which is called keyframe selection, should be performed at each iteration before updating the motion.

Type of correspondences	Monocular	Stereo
2D–2D	X	X
3D–3D		X
3D–2D	X	X

Figure 4.27: Correspondences between motion estimation and monocular/stereo camera [18]

In a monocular system, as highlighted in Figure 4.27, although the 2-D-to-2-D method is typically favored since it bypasses the need for triangulating points, the 3-D-to-2-D approach is more frequently employed. This is primarily because the 3-D-to-2-D method allows for faster data association, making it more practical in many applications.

Additionally, ensuring that the input data are free of outliers is crucial for precise motion computation. The process of outlier rejection is particularly sensitive, and

the time required for this operation is closely related to the minimum number of points needed to accurately estimate the motion:

- The 2-D-to-2-D case requires a minimum of five-point correspondences.
- In the 3-D-to-2-D motion case only three correspondences are necessary.

This lower number of points used in the 3D-to-2D approach results in a much faster motion estimation.

STEP 4: Error propagation

In VO, the current robot pose C_k is determined by concatenating individual transformations $T_{k,k-1}$. Each transformation $T_{k,k-1}$ carries its own uncertainty, which impacts the overall uncertainty of the camera pose C_k. This cumulative effect of uncertainties from past transformations is illustrated in Figure 4.28. The uncertainty associated with each transformation $T_{k,k-1}$ arises from factors such as camera geometry and image features. Since the uncertainty of the camera pose tends to increase as transformations are concatenated, it is crucial to keep the uncertainties of individual transformations small to reduce drift.

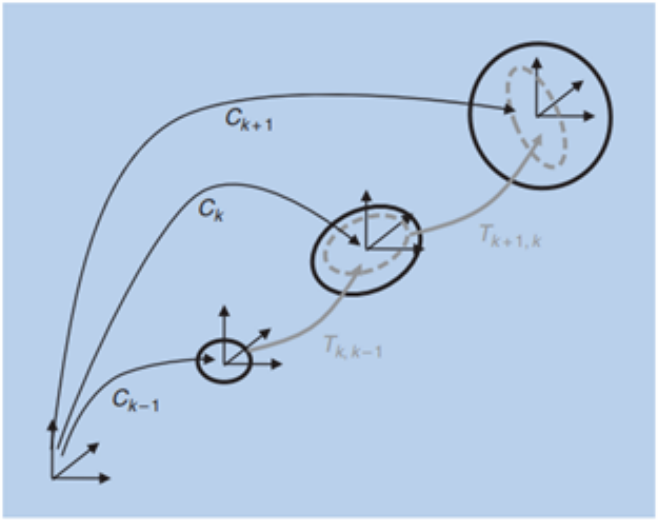


Figure 4.28: The uncertainty of the camera pose at C_k is a combination of the uncertainty at C_{k-1} (black solid ellipse) and the uncertainty of the transformation $T_{k,k-1}$ (gray dashed ellipse) [16]

4.1.5 Camera pose optimization

In VO, the standard approach to determining camera poses involves linking transformations from successive views, typically between times k and $k-1$. Alternatively, it is possible to calculate transformations not just between consecutive time steps but also between the current time k and earlier time points, such as $T_{k,k-2}$ through $T_{k,k-n}$, or even between any two arbitrary time instances $T_{i,j}$. When these additional transformations are available, they can be leveraged as supplementary constraints

in a pose-graph optimization framework, thereby refining the estimation of camera poses.

The final step in the VO pipeline is local optimization of the camera pose. The most common approaches for pose optimization are Pose-Graph Optimization and Windowed (or Local) Bundle Adjustment.

4.1.6 Bundle Adjustment

Bundle Adjustment simultaneously optimizes 3D geometry, relative poses, and intrinsic camera parameters. The optimization problem is formulated to minimize the weighted sum of squared reprojection errors. Given p 3D points observed from q different frames, let $\mathbf{x}_{ij}$ be the i-th projection of the i-th point in the j-th frame. The minimization objective is:

$$\min_{\mathbf{X},\mathbf{P}_i} \sum_{i=1}^{p} \sum_{j=1}^{q} b_{ij} \|\mathbf{x}_i - \hat{\mathbf{x}}_i(\mathbf{P}_i, \mathbf{X})\|^2$$

where $\mathbf{X}_i$ is the 3D point, $\mathbf{P}_j$ is the camera projection matrix for the j-th frame, $\pi(\cdot)$ denotes the projection function, and b_{ij} is a binary indicator (1 if point i is visible in frame j, otherwise 0).

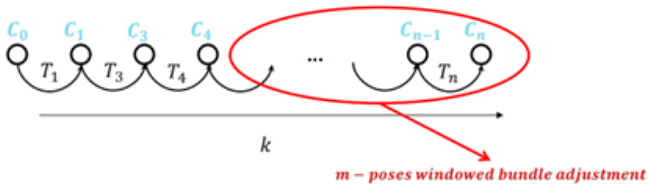


Figure 4.29: An iterative refinement over last m poses can be performed to obtain a more accurate estimate of the local trajectory [18]

Local Bundle Adjustment optimizes the poses and 3D points for a set of co-visible frames while keeping the rest fixed, as shown in Figure 4.29. Global Bundle Adjustment optimizes all frames and 3D points, typically fixing only the initial frame.

Performing local bundle adjustment over the last m frames yields a more accurate estimate of the trajectory, attenuating errors from previous motion estimation steps.

4.2 Outlier removal

The presence of outliers in matched points can lead to incorrect data associations. These outliers may arise from various issues such as image noise, occlusions, blurring, or changes in the scene's lighting and perspective that are not adequately accounted for by the feature detection or description algorithms. For example, many feature-matching methods rely on simplifying assumptions, such as linear variations in lighting, uniform camera rotation and zoom, or affine transformations,

these models serve as approximations and do not capture more complex phenomena like image saturation, perspective distortions, and motion blur. Achieving precise camera motion estimation requires the identification and elimination of outliers, making the management of these discrepancies one of the most challenging aspects of VO.

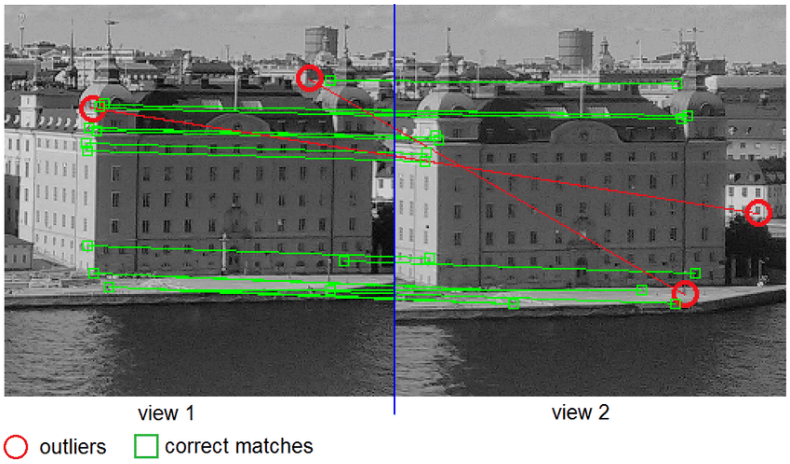


Figure 4.30: Outlier and inlier identification between two matched images [21]

RANSAC for outlier removal

The RANSAC is a widely used technique for robust motion estimation in the presence of outliers. The algorithm works by generating model hypotheses from randomly sampled data points and evaluating these hypotheses against other remaining data points. The hypothesis that fits the most data points, or achieves the highest consensus, is selected as the best model. For two-view motion estimation, the model to estimate is the relative motion $(\mathbf{R}, \mathbf{t})$ between two camera positions, and the data points are the feature correspondences.

Even when a large number of feature correspondences are incorrect due to mismatches or independently moving objects, RANSAC is able to extract valid information by iteratively sampling subsets of the data and applying a consensus criterion to find the correct transformation. This robustness, combined with the ability to tune parameters like the number of iterations and error thresholds, makes RANSAC a powerful tool for balancing accuracy and computational efficiency in real-world motion estimation tasks.

The outline of RANSAC is given in the Table 4.5.

The number of iterations N,which represents the number of times the algorithm randomly selects a subset of data points to generate and evaluate a model hypothesis with a desired probability p is given by:

$$N \geq \frac{\log(1-p)}{\log(1-(1-\epsilon)^m)} \tag{4.3}$$

where m is the minimum number of matches required, and ϵ is the outlier percentage. For robustness, N is often multiplied by a factor of ten in practice.

Algorithm 1. VO from 2-D-to-2-D correspondences

1) **Initial:** Let A be a set of N feature correspondences.
2) **Repeat:**
   2.1) Randomly select a sample of s points from A.
   2.2) Fit a model to these points.
   2.3) Compute the distance of all other points to this model.
   2.4) Construct the inlier set (i.e., count the number of points whose distance from the model $< d$).
   2.5) Store these inliers.
   2.6) Until maximum number of iterations reached.
3) The set with the maximum number of inliers is chosen as a solution to the problem.
4) Estimate the model using all the inliers.

Table 4.5: Steps of the VO algorithm from 2-D-to-2-D correspondences.

The number of iterations N grows exponentially with the number of data points required to estimate the model s, as can be seen from the equation for N. This emphasizes the importance of using a minimal parameterization to reduce computational complexity. Particularly, when the camera is calibrated, its six DoF motion can be determined using a minimum of five point correspondences. An additional advantage is that the five-point solver also works effectively for planar scenes.

RANSAC is probabilistic and non-deterministic; it guarantees a solution with increasing probability as the number of iterations grows.

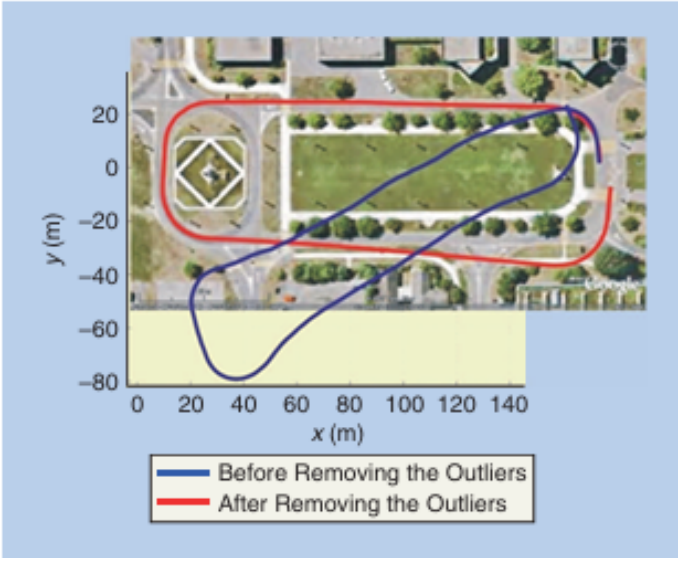


Figure 4.31: Comparison between VO trajectories estimated before and after removing the outliers. (Photo courtesy of Google Maps © 2007 Google, © 2007 TeleAtlas.)

Chapter 5

Visual Inertial Odometry

VIO is the technique of estimating an agent's pose and velocity using data from a combination of one or more cameras and IMU attached to it. The cameras and IMUs are rigidly fixed together, implying that their positions relative to each other remain constant while the system generates visual and inertial measurements at different rates.

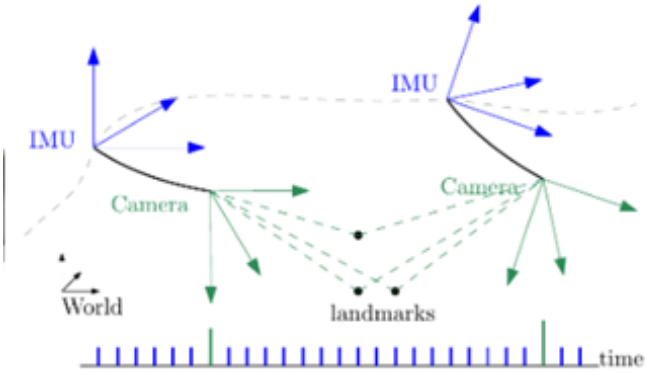


Figure 5.1: Visual inertial odometry scheme [22].

In this thesis, a monocular camera and an IMU are utilized because, in most cases, a monocular setup is preferred. A single camera combined with an IMU forms the minimal sensor suite required for accurate state estimation. This configuration is particularly advantageous for flying robots, as it offers lower weight and power consumption compared to other sensor setups, such as stereo or multi-camera systems.

As highlighted by Scaramuzza in [22], cameras and IMUs complement each other as sensor types. Cameras capture 2D images by accumulating photons during exposure, making them highly accurate for slow movements while providing rich visual data. However, cameras have limitations, including a relatively low output rate (around 100Hz) and reduced performance in scenes with low texture and high-speed motion which causes the camera to capture the movement of objects during the exposure time, resulting in motion blur or in High Dynamic Range (HDR) environments, where images can be over or under exposed.

Based on the previous camera considerations, VO methods, which rely on camera images to estimate motion, encounter several specific challenges. One of the main problems is tracking loss, which occurs when there is insufficient overlap of features between successive images. This typically happens when the camera undergoes sudden movements or operates at a low frame rate, leading to tracking loss and decreased accuracy. Another challenge is scale ambiguity, particularly in monocular VO setups. In such cases, while motion estimation may be accurate, the scale of the scene cannot be directly determined, resulting in the incorrect representation of the relative size of movements.

On the other hand, an IMU provides good odometry information for large sudden movements across a small time interval and information coming from the IMU is scaled correctly. Furthermore IMUs are proprioceptive sensors that measure angular velocity and external acceleration, providing data on the device's motion and orientation relative to its surroundings without relying on external references. Since IMUs do not rely on visual data, they are not affected by the limitations that impact cameras such as low texture or high-speed conditions. Even the IMU has some limitations, they offer a high output rate (around 1,000Hz), but they suffer from noise and inaccuracies at low velocities or accelerations and are prone to drift due to sensor biases. This drift accumulates quickly when relying only on IMU data, therefore the performance of inertial-based odometry methods deteriorate with time and are unreliable for long-term pose estimation [22].

Given these characteristics, combining cameras with IMUs can deliver robust and accurate state estimation across various scenarios. Such information, coming from these two sensors, must be combined in order to obtain a single pose estimate and technically, this process of merging data from different sensors can be done in various ways as shown in the literature, as we can see in Figure 5.2.

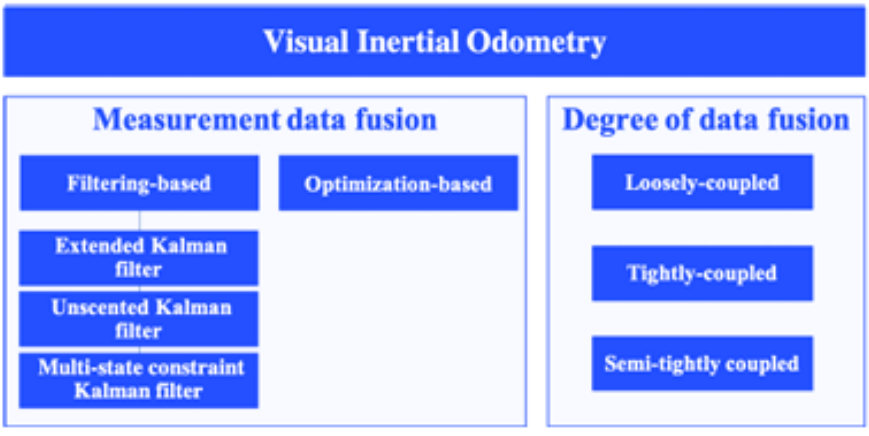


Figure 5.2: General classification of VIO techniques proposed in literature [1].

The main categories of VIO solutions can be classified into three groups and these can be divided into three categories depending on the stage at which sensor fusion is performed: loosely-coupled, semi-tightly coupled and tightly-coupled approaches. Furthermore, VIO systems can be further classified based on the method of data fusion between visual data and IMU, distinguishing between filtering-based and

optimization-based approaches.

5.1 Different approaches VIO

The VIO algorithms can be classified into loosely-coupled and tightly-coupled approaches according to the type of information used for visual and inertial sensor fusion.

5.1.1 Loosely-coupled approach

In a loosely coupled VIO system, visual and inertial information are processed separately. Each module independently estimates the vehicle's pose, treating them as distinct entities. Then the poses estimated by the VO and IMU modules are fused and processed to refine the vehicle's ego-motion estimation [1].

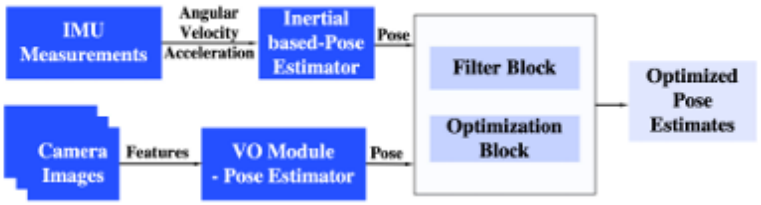


Figure 5.3: General framework for Loosely-coupled Visual inertial odometry [1]

The loosely-coupled framework is efficient because it requires less computational power, making it advantageous for real-time applications, at the same time it also allows for easier integration of additional sensors. However, a significant drawback is the potential loss of accuracy during the fusion of separately estimated poses. This inaccuracy arises from the decoupled approach, which fails to account for the interactions between visual and inertial parameters.

5.1.2 Tightly-coupled approach

The tightly-coupled approach involves estimating motion by directly using raw data from both the camera and IMU, specifically feature positions, angular velocity, and linear acceleration readings. In this method, the camera and IMU data are combined to simultaneously construct the motion and observation equations, enabling more accurate pose estimation.

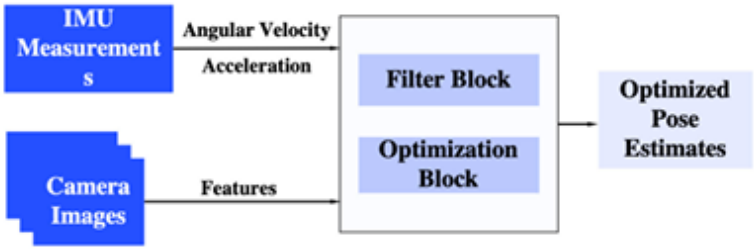


Figure 5.4: General framework for Tightly-coupled Visual inertial odometry [1]

One of the key advantages of the tightly-coupled approach is that the IMU can be used to guide feature matching between frames. The IMU provides detailed information about the device's motion—such as acceleration and rotation—between camera captures, allowing for more accurate predictions of where visual features are likely to appear in the next frame. By focusing the search for features within these predicted regions, rather than across the entire image, the algorithm significantly improves the likelihood of finding correct feature correspondences, known as inliers.

With more inliers, the motion estimation algorithm is able to compute a more precise estimate of the camera's movement between frames, which is essential for tasks like VO and Simultaneous Localization and Mapping (SLAM). This guided feature matching reduces errors and enhances the robustness of motion estimation by constraining the search space for feature matches, thereby boosting accuracy.

Additionally, the tightly-coupled approach inherently accounts for the coupling between visual and inertial data, allowing for better drift compensation over time. This leads to a more stable and reliable system compared to the loosely-coupled approach, where the independent processing of sensor data can result in cumulative errors. Despite its complexity, the tightly-coupled method offers a significant improvement in accuracy due to the deeper integration and cooperation between the camera and IMU sensors.

5.2 Data fusion

In VIO systems, the fusion of data from both cameras and IMU sensors allows to achieve accurate and robust motion estimation. One of the most common methods for fusing sensor data is through filtering approaches, such as the KF and its extended versions. These filtering techniques enable efficient state estimation by focusing on the system's current state, which is updated based on the most recent observations while accounting for uncertainty in the measurements [22]. This approach is computationally efficient and suitable for real-time applications, making it popular in resource-constrained platforms like UAVs, such as in our case [1].

In filtering methods, the fusion process typically involves two stages: predicting the state based on IMU data and correcting the state using vision-based measurements [1]. By integrating the IMU's acceleration and rotational data, these filters help address the monocular vision system's inherent scale ambiguity, improving the accuracy of motion estimation.

However, filtering approaches have limitations, and a major source of error comes from how they handle older states. In these systems, information from past states is absorbed into estimating the current state, and older states are then discarded or "marginalized." While this makes the algorithm more efficient, it introduces a significant drawback: any errors present in those older states, such as inaccuracies resulting from linearization or incorrect sensor readings (outliers), are permanently embedded in the current state. Once these errors are incorporated, they cannot be corrected because the filter no longer retains the original data. This means

that errors made in previous estimates remain "stuck" in the system, potentially reducing the accuracy of future estimates.

5.2.1 Bayesian Inference

An agent, like an UAV, cannot measure directly the state of the environment, but it must infer its pose from data. This represents the belief of the robot that can be defined as its internal knowledge about the state of the environment. Probabilistic robotics represents beliefs through conditional probability distributions and assigns a probability to each possible hypothesis with regards to the true state [23].

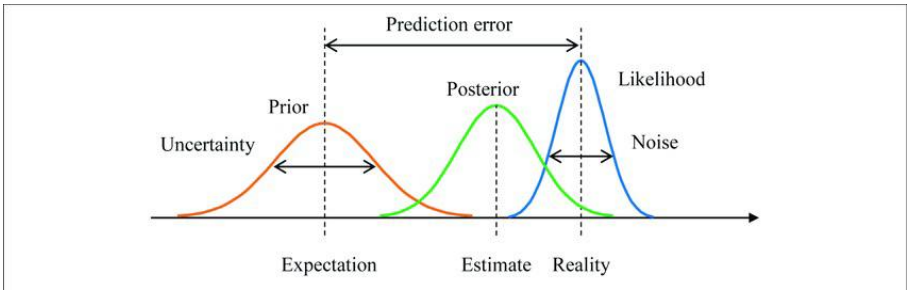


Figure 5.5: Description of a belief updating process according to Bayesian inference, where an initial estimate (Prior) is combined with observed data (Likelihood) to form a new estimate (Posterior), with the possibility of prediction error highlighted.

The belief over the state variable x_t can be denoted by $\text{bel}(x_t)$:

$$\text{bel}(x_t) = p(x_t \mid z_{1:t}, u_{1:t})$$

that represents the posterior probability over the state x_t at time t, conditioned on all past measurements $z_{1:t}$ and all past controls $u_{1:t}$.

A posterior distribution before incorporating z_t, immediately after performing the u_t control action. This posterior distribution will be denoted as follows:

$$\overline{\text{bel}}(x_t) = p(x_t \mid z_{1:t-1}, u_{1:t})$$

It is often called a prediction and reflects the fact that $\text{bel}(x_t)$ predicts the state at time t based on the posterior belief of the previous state. The calculation of $\text{bel}(x_t)$ from $\overline{\text{bel}}(x_t)$ is called correction or update of the measurement.

Bayesian inference is an important tool for statistical estimation that uses the prior belief about the system state, combines it with the current observations and determines belief regarding the current system state.

The prediction belief depends on the probability associated with the transition from state x_{t-1} to state x_t and the probability density of the previous state:

$$p(x_t|z_{1:t-1}) = \int p(x_t|x_{t-1})p(x_{t-1}|z_{1:t-1})dx_{t-1} \tag{5.1}$$

The probability density of the current state, after the current observation has been made, depends on the measurement likelihood and the prior belief:

$$p(x_t|z_{1:t}) = \frac{p(z_t|x_t)p(x_t|z_{1:t-1})}{p(z_t|z_{1:t-1})} \tag{5.2}$$

The term $p(z_t|z_{1:t-1})$ is determined by:

$$p(z_t|z_{1:t-1}) = \int p(z_t|x_t)p(x_t|z_{1:t-1})dx_t \tag{5.3}$$

The Equations 5.1 to 5.3 give us a recursive Bayesian estimation to determine the conditional probability density $p(x_t|z_{1:t})$.

5.2.2 Kalman Filter

The KF is a Gaussian filter that is part of the family of recursive Bayesian state estimators designed for multivariate normal distributions.

The KF represents beliefs through moment representation, this means that at time t, the belief is characterized by the mean μ_t and the covariance Σ_t. Posteriors are Gaussian if the following three properties hold, alongside the Markov assumptions of the Bayes filter.

1. The state transition probability $p(x_t|u_t, x_{t-1})$ must be a linear function of its arguments with additive Gaussian noise, so the KF assumes linear system dynamics. This is expressed by the following equation:

   $$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t \tag{5.4}$$

   Here, x_t and x_{t-1} are state vectors at time t and t-1 respectively and u_t is the control vector at time t.

   The random variable ϵ_t is a Gaussian random vector that models the randomness in the state transition. Its mean is zero, and its covariance is denoted as Q_t. ϵ_t takes into account various factors that can influence the evolution of the system, including modeling errors, external disturbances and process noise. Including ϵ_t in the state transition equation, the KF can effectively handle these uncertainties, leading to more accurate state estimates and predictions.

2. The measurement probability $p(z_t|x_t)$ must also be linear in its arguments, with added Gaussian noise:

   $$z_t = C_t x_t + \omega_t \tag{5.5}$$

   The vector ω_t describes the measurement noise, it captures inherent uncertainties in the measurement process, allowing the KF to effectively integrate sensor data and produce reliable state estimates despite the presence of noise. The distribution of ω_t is a multivariate Gaussian with zero mean and covariance R_t.

3. Finally, the initial belief $p(x_0)$ must be normally distributed. We denote the mean of this belief as μ_0 and the covariance as Σ_0:

$$p(x_0) = \mathrm{N}(x_0; \mu_0, \Sigma_0) \tag{5.6}$$

These three assumptions are sufficient to ensure that the posterior $p(x_t)$ is always Gaussian, for any time point t [23].

Main steps of Kalman Filter

The filter consists of two parts as we can see in the Figure 5.6: **prediction** and **correction**. The prediction step uses the system equations to determine a prior estimate of the state at the current time step. The correction step updates the prior estimate based on observations made.

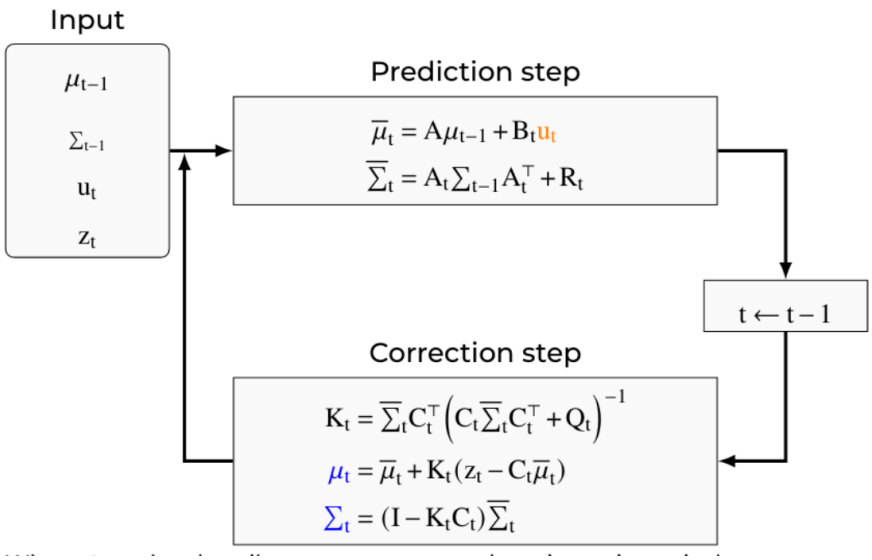


Figure 5.6: The Kalman filter algorithm

The KF algorithm illustrates how beliefs are updated over time. At time t, the KF represents the belief $\mathrm{bel}(x_t)$ using the mean μ_t and the covariance Σ_t. The algorithm takes as input the previous belief from time $t-1$, represented by μ_{t-1} and Σ_{t-1}, along with the control u_t and the measurement z_t.

To compute the predicted belief, $\bar{\mu}$ and $\bar{\Sigma}$, which represent the belief one time step later, the control input is incorporated. The mean is updated using the deterministic state transition function, substituting μ_{t-1} for the state x_{t-1}. The covariance update accounts for the dependence of states on previous states through the linear matrix A_t.

Next, the predicted belief is transformed into the updated belief $\mathrm{bel}(x_t)$ by incorporating the measurement z_t. The Kalman gain K_t determines how much the measurement influences the new state estimate. The mean is adjusted in proportion to the Kalman gain and the difference between the actual measurement z_t and the predicted measurement based on the measurement probability. Finally, the new covariance of the posterior belief is calculated, adjusting for the information gain obtained from the measurement.

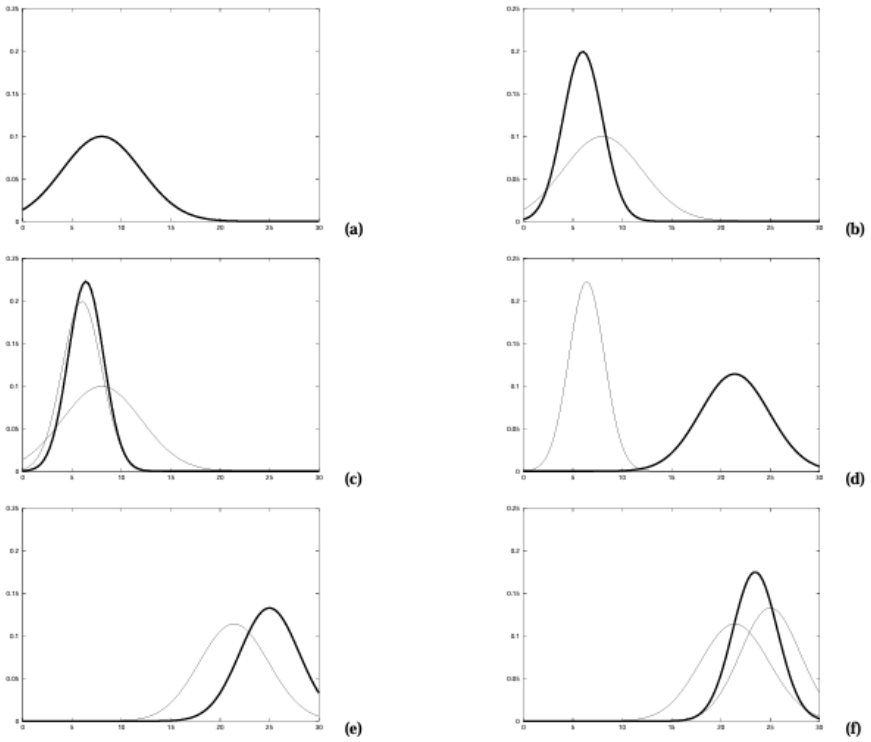


Figure 5.7: Illustration of Kalman filters: (a) initial belief, (b) a measurement (in bold) with the associated uncertainty, (c) belief after integrating the measurement into the belief using the KF algorithm, (d) belief after motion to the right (which introduces uncertainty), (e) a new measurement with associated uncertainty, and (f) the resulting belief [23].

Positive and negative aspects of Kalman filter

The KF is computationally quite efficient, since only two parameters, μ and Σ are needed to describes the state of a system rather than estimating the full probability distribution. However, two critical factors influence the complexity of the algorithm:

- $(C_t\overline{\Sigma_t}C_t^T+Q_t)^{-1}$: The complexity of matrix inversion is approximately $O(k)^{2.376}$ for polynomial in the dimensionality of the measure k.
- K_tC_t: the complexity of this product of matrices is $O(n^2)$, where n is the dimension of the state space.

Moreover, this approach is optimal for linear Gaussian systems, but many real-world systems are nonlinear, making the KF unsuitable for such applications. Furthermore, the KF can only model unimodal beliefs, which presents a significant limitation for various real-world scenarios.

5.2.3 Extended Kalman Filter

The KF is built on the assumptions of linear state transitions and linear measurements with added Gaussian noise, that are rarely fulfilled in practice [23].

The Extended Kalman Filter (EKF) overcomes the linearity assumption. Here, the next state probability and the measurement probabilities are represented by nonlinear functions g and h, respectively:

$$x_t = g(x_{t-1}, u_t) + \epsilon_t$$

$$z_t = h(x_t) + \omega_t$$

Under these conditions, performing the belief update exactly is usually impossible for nonlinear functions g and h, in the sense that the Bayes filter does not possess a closed-form solution. The distribution is Gaussian as long as we start with Gaussian distribution and perform only linear transformations, as shown in Figure 5.8. However, the distribution don't remain Gaussian, when non-linear transformations are applied to an initially Gaussian distribution as shown in Figure 5.9.

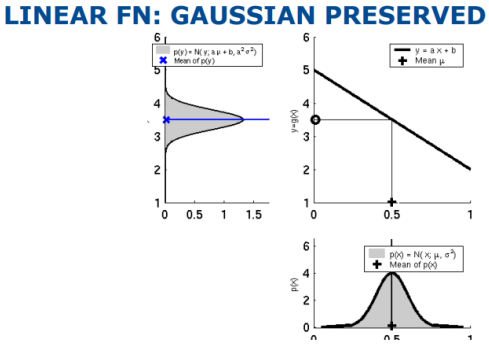


Figure 5.8: Gaussian distribution is preserved after performing linear transformation.

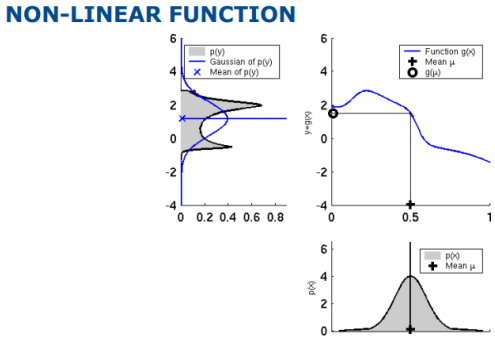


Figure 5.9: Non-linear transformations distort the distribution, resulting in a non-Gaussian form.

Linearization Via Taylor Expansion

The standard KF can be applied in this scenario if the system is linearized at each time step. The EKF achieves this, by linearizing the system around the current mean estimate by applying a first-order Taylor expansion to the functions $g(.)$ and $h(.)$ around the mean.

Linearization approximates g and h by a linear function that is tangent to them at the mean of the Gaussian. As mentioned earlier, projecting the Gaussian through this linear approximation, the posterior is Gaussian. In fact, once g and h are linearized, the mechanics of belief propagation are equivalent to those of the KF.

Linearization of the state transition function $g(.)$:

$$\begin{aligned} g(x_k, u_k) &\approx g(\mu_{k-1}, u_k) + \left.\frac{\partial f(x_{k-1}, u_{k-1})}{\partial x_{k-1}}\right|_{x_{k-1}=\hat{x}_{k-1}} (x_{k-1} - \mu_{k-1}) \\ &= g(\mu_{k-1}, u_k) + G_{k-1}(x_{k-1} - \mu_{k-1}) \end{aligned}$$

G_t is a matrix of size $n \times n$, with n denoting the dimension of the state. This matrix is often called the Jacobian. The value of the Jacobian depends on u_t and

μ_{t-1}, hence it differs for different points in time.

When linearizing a function like g in the context of state estimation, a suitable approach is to select an argument that reflects the state considered most probable at the time of linearization. In the case of Gaussian distributions, the state that has the maximum probability is the mean of the posterior distribution, denoted as μ_{t-1}. By using the mean μ_{t-1} as the argument for g, we ensure that our linear approximation is closely aligned with the most probable state, thus improving the accuracy of the linearization. This choice is fundamental to maintain the effectiveness of the extended Kalman filter.

The same linearization is implemented for the measurement function h. Here, the Taylor expansion is developed around $\bar{\mu}_t$, the state considered most probable when h is linearized.

$$h(x_t) \approx h(\bar{\mu}_t) + \left.\frac{\partial h(x_t)}{\partial x_t}\right|_{x_t=\hat{x}_k} (x_t - \bar{\mu}_t) = h(\bar{\mu}_t) + H_t(x_t - \bar{\mu}_t)$$

where H_t is the Jacobian matrix associated to the measurement model.

The EKF Algorithm

Being the EKF an extension of the KF for non-linear applications. The EKF algorithm, shown in 5.1, is similar to the standard KF algorithm described in the previous paragraph.

Extended_Kalman_filter(μ_{t-1}, Σ_{t-1}, u_t, z_t):	**Kalman_filter**
Prediction	
$\bar{\mu}_t = g(\mu_{t-1}, u_t)$	$\mu_t = A_t\mu_{t-1} + B_t u_t$
$\bar{\Sigma}_t = G_t\Sigma_{t-1}G_t^\top + R_t$	$\Sigma_t = A_t\Sigma_{t-1}A_t^\top + R_t$
Correction	
$K_t = \bar{\Sigma}_t H_t^\top (H_t\bar{\Sigma}_t H_t^\top + Q_t)^{-1}$	$K = \Sigma_t C_t^\top (C_t\Sigma_t C_t^\top + Q_t)^{-1}$
$\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$	$\mu_t = \bar{\mu}_t + K_t(z_t - C_t\bar{\mu}_t)$
$\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$	$\Sigma_t = (I - K_t C_t)\bar{\Sigma}_t$
Return μ_t, Σ_t	

Table 5.1: Extended Kalman filter algorithm

Positive and negative aspects of Extended Kalman Filter

The EKF is appreciated for its simplicity and computational efficiency. Each update takes $O(k^{2.8} + n^2)$, where k represents the size of the measurement vector z_t and n the size of the state vector x_t. This makes it very efficient, especially because it approximates the probability distribution with a multivariate Gaussian. However, an important limitation comes from the use of the Gaussian distribution.

Another crucial limitation of the EKF is due to its approximation of state transitions and measurements model by linear Taylor expansions. The accuracy of this

approximation depends mainly on two factors: the degree of nonlinearity of the functions and the level of uncertainty, that means the width of the posterior.

The EKF is based on linearizing nonlinear functions via a first-order Taylor expansion around the current estimate. When the underlying system functions are nearly linear, this approximation is close to the true model, allowing the EKF to provide reliable estimates. In such cases, the linearized model accurately captures most of the system dynamics and the Gaussian distribution that the EKF uses to represent uncertainty can follow the true state evolution without significant errors. However, when the functions are highly nonlinear, this linear approximation becomes inadequate. Non-linearities, especially those involving significant changes or complex dynamics, cannot be effectively captured by a first-order Taylor expansion. As a result, linearization introduces significant errors, which accumulate over time, leading to increasingly inaccurate state estimates, as illustrated in the Figure 5.10 and 5.11.

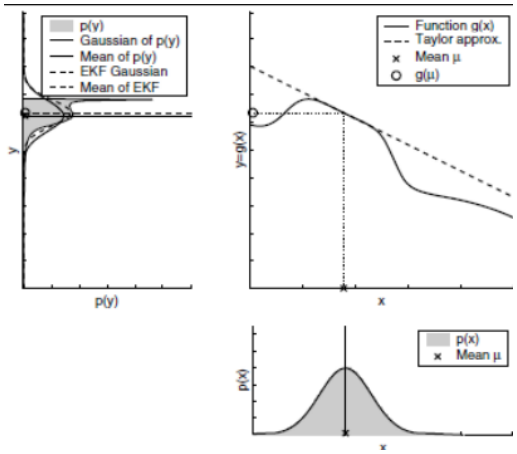


Figure 5.10: In a system with low nonlinearity, the EKF's linearization provides accurate state estimates.

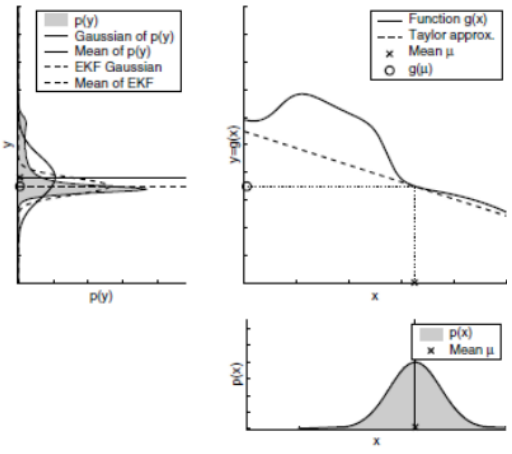


Figure 5.11: In a system with high non-linearity, the EKF struggles to approximate the true dynamics, leading to significant estimation errors.

Furthermore, greater uncertainty in the system leads to a wider Gaussian estimate, accentuating the effect of nonlinearity and reducing the accuracy. Therefore, it is essential to keep the uncertainty in the estimate low to achieve good performance. This is clearly visible in the Figure 5.12 and 5.13.

Finally, the calculation and inversion of the Jacobian matrix, which is required for linearization, can be computationally expensive. The EKF works well with moderate nonlinearity, but with highly nonlinear models or under high uncertainty, it can suffer from approximation errors that accumulate over time, leading to drift in the estimates.

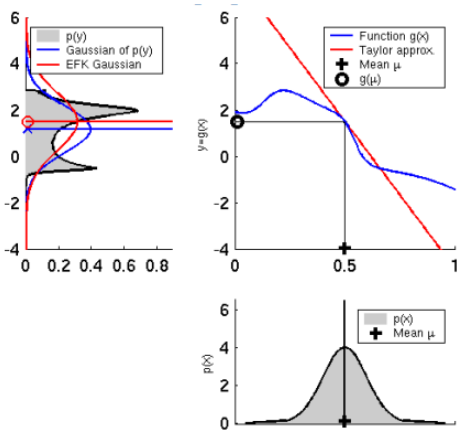


Figure 5.12: With low uncertainty, the Gaussian estimate is narrow, allowing the EKF's linearization to maintain accuracy.

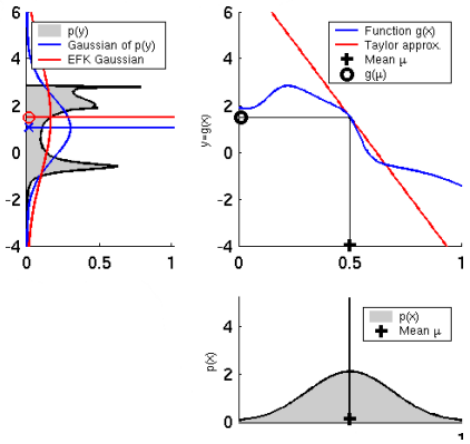


Figure 5.13: As uncertainty grows, the Gaussian estimate becomes wider, amplifying the effect of nonlinearity and leading to a less accurate state prediction

5.2.4 Error-State Extended Kalman Filter

The ES-EKF is an advanced version of the standard EKF specifically designed for state estimation tasks in systems with nonlinear dynamics and measurement models. This formulation is used to overcome the drawbacks in a full-state KF, estimating the accumulated error instead of the true state.

The ES-EKF formulations is based on the concept of true state, nominal state and error state. The true state is expressed as a combination through a linear sum, quaternion multiplication, or matrix multiplication — of the nominal state and the error state. Essentially, the nominal state represents the system's principal behavior, which can be integrated in a non-linear manner. In contrast, the error state captures smaller discrepancies or adjustments with respect to the nominal state, making it suitable for linear integration and Gaussian filtering.

The error-state is simply the arithmetic difference of the estimated state $\hat{x}$ to the true state x, which is, $\tilde{x} = x - \hat{x}$. While an error quaternion δq is defined as,

$$\delta q = q \otimes \hat{q}^{-1} = \begin{bmatrix} \delta q_w \\ \delta q_v \end{bmatrix} = \begin{bmatrix} \cos(\delta\theta/2) \\ \mathbf{k}\sin(\delta\theta/2) \end{bmatrix} \approx \begin{bmatrix} 1 \\ \frac{1}{2}\mathbf{k}\delta\theta \end{bmatrix},$$

where q_v is the imaginary part and q_w is the real part of quaternion q. The error quaternion is approximated by $\mathbf{k}\delta\theta$, which is a three-dimensional vector that represents the error angle. All the variables in the ES-EKF are summarized in the following image.

Error-State Extended Kalman filter algorithm

The ES-EKF operates as follows: High-frequency data is utilized to determine the nominal state. However, this nominal state doesn't account for noise w or any other model inaccuracies, so errors will gradually accumulate. These errors are tracked

Magnitude	True	Nominal	Error	Composition	Measured	Noise
Full state ([1])	$\mathbf{x}_t$	$\mathbf{x}$	$\delta\mathbf{x}$	$\mathbf{x}_t = \mathbf{x} \oplus \delta\mathbf{x}$		
Position	$\mathbf{p}_t$	$\mathbf{p}$	$\delta\mathbf{p}$	$\mathbf{p}_t = \mathbf{p} + \delta\mathbf{p}$		
Velocity	$\mathbf{v}_t$	$\mathbf{v}$	$\delta\mathbf{v}$	$\mathbf{v}_t = \mathbf{v} + \delta\mathbf{v}$		
Quaternion ([2,3])	$\mathbf{q}_t$	$\mathbf{q}$	$\delta\mathbf{q}$	$\mathbf{q}_t = \mathbf{q} \otimes \delta\mathbf{q}$		
Rotation matrix ([2,3])	$\mathbf{R}_t$	$\mathbf{R}$	$\delta\mathbf{R}$	$\mathbf{R}_t = \mathbf{R}\,\delta\mathbf{R}$		
Angles vector ([4])			$\delta\boldsymbol{\theta}$	$\delta\mathbf{q} = e^{\delta\boldsymbol{\theta}/2}$ $\delta\mathbf{R} = e^{[\delta\boldsymbol{\theta}]_\times}$		
Accelerometer bias	$\mathbf{a}_{bt}$	$\mathbf{a}_b$	$\delta\mathbf{a}_b$	$\mathbf{a}_{bt} = \mathbf{a}_b + \delta\mathbf{a}_b$		$\mathbf{a}_w$
Gyrometer bias	$\boldsymbol{\omega}_{bt}$	$\boldsymbol{\omega}_b$	$\delta\boldsymbol{\omega}_b$	$\boldsymbol{\omega}_{bt} = \boldsymbol{\omega}_b + \delta\boldsymbol{\omega}_b$		$\boldsymbol{\omega}_w$
Gravity vector	$\mathbf{g}_t$	$\mathbf{g}$	$\delta\mathbf{g}$	$\mathbf{g}_t = \mathbf{g} + \delta\mathbf{g}$		
Acceleration	$\mathbf{a}_t$				$\mathbf{a}_m$	$\mathbf{a}_n$
Angular rate	$\boldsymbol{\omega}_t$				$\boldsymbol{\omega}_m$	$\boldsymbol{\omega}_n$

Figure 5.14: All variables in Error state extended Kalman Filter [24]

within the error state and are estimated through the ES-EKF, which accounts for all noise sources and disturbances. The nominal and error states are predicted in parallel.

At this level, without external measurements available for correction, the filter performs only predictions. Correction is introduced when external data, such as GPS or visual inputs, become available. These measurements typically occur less frequently and make it possible to observe the errors. At this point, the correction gives a posterior Gaussian estimate of the error state. The error-state mean is then added to the nominal state to determine the corrected state estimates, shown by

$$\hat{x}_k = \hat{x}_k^n + \delta\hat{x}_k,$$

where $\hat{x}_k$ is the state estimate, $\hat{x}_k^n$ is the nominal state, and $\delta\hat{x}_k$ is the error state. At the end the error state is reset to zero. The error-state covariance matrix is then updated to reflect this reset, and the process repeats indefinitely.

The main steps followed in the ES-EKF are shown in the Figure 5.15.

Positive and negative aspects of Error - State Extended Kalman filter

The ES-EKF offers significant advantages for state estimation in complex systems, especially those involving nonlinear dynamics and quaternion-based orientation representations. Furthermore a comparative analysis in [44], [45] and [46] between the classic EKF and the ES-EKF shows that the ES-EKF is robust to a variety of aircraft maneuvers as well as imperfect tuning of the sensor noise covariance [47]. Moreover, it ensures numerical stability and allows the quaternions to be treated in their minimal representation.

One of its key strengths is computational efficiency, since by focusing on the error state rather than the full state, the ES-EKF simplifies the system dynamics, making it highly effective for applications such as UAV localization and stabilization, where timely processing is critical. Furthermore, it ensures numerical stability

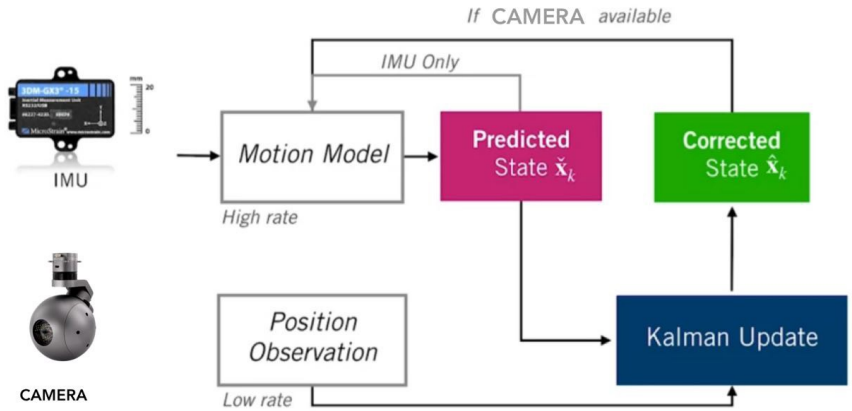


Figure 5.15: Error-State Extended Kalman filter algorithm

by addressing quaternion constraints, since it works with perturbations around a nominal trajectory rather than directly manipulating quaternions. This approach mitigates the risks of singularities and gimbal lock, ensuring a reliable orientation estimate. The small magnitude of the error state further contributes to computational efficiency, since it makes second-order terms negligible and simplifies Jacobian calculations. Additionally, since corrections can be made less frequently than expected, the overall filtering process becomes more efficient.

Despite these advantages, the ES-EKF also has its disadvantages. The filter implementation can be complex, requiring a solid understanding of the dynamics of the error state and covariance propagation. Additionally, there is a risk of error propagation, as estimation errors in the error state can accumulate over time, potentially leading to divergence or suboptimal performance if not carefully managed. Furthermore, while the ES-EKF excels in constrained nonlinear systems, it may be useless for simpler systems, where a traditional full-state KF could provide adequate performance without the additional computational overhead.

Chapter 6

Simulation environment and synthetic dataset

To evaluate and test the performance of the VIO algorithm, a simulation platform was developed to generate a synthetic dataset. This platform uses ROS2, PX4 Autopilot, Gazebo and QGroundControl, providing a versatile and flexible framework to simulate real-world conditions. The synthetic dataset generated by this simulation environment contains various sensor readings at different timestamps, along with the corresponding GT, position, and orientation data, which are important to validate the localization and state estimation algorithm.

6.0.1 ROS2

ROS2 is an open source middleware framework that provides an infrastructure for developing robotic systems. It supports multi-robot communication, hardware abstraction and complex sensor integration, making it ideal for the simulation environment. ROS2 acts as a backbone to manage communication between the various modules within the simulation, allowing for seamless interaction between the drone, sensors and control systems.

6.0.2 PX4

PX4 is an open source flight control software for drones and other unmanned vehicles [25]. By integrating PX4 Autopilot with ROS2 and Gazebo, the flight control system is able to generate realistic flight behavior and coordinate the motion of the UAV during simulation.

The ROS2-PX4 architecture provides tight integration between ROS2 and PX4, allowing ROS2 publisher and subscriber nodes to interact directly with PX4 uORB topics [25]. Communication between ROS2 and the autopilot is facilitated by the ROS2-PX4 bridge, which ensures smooth and real-time data exchange for precise control. In this configuration, two bridges are required, one to handle camera-related topics and the other to handle IMU-related topics. The following commands are used to launch the ROS2 nodes implementing the previously mentioned bridges:

```
ros2 run ros_gz_image image_bridge /camera
```

```
ros2 run ros_gz_bridge parameter_bridge/world/baylands/model
/x500_mono_cam_0/link/base_link/sensor/imu_sensor/imu@sensor_msgs
/msg/Imu[gz.msgs.IMU
```

PX4 leverages ROS2 to enable offboard control via an accompanying Linux computer, allowing external software to manage the PX4 flight stack independently of the autopilot. The PX4 flight stack gives GT information in the North-East-Down (NED) frame, as well as data from the IMU and camera sensors. The three topics used to collect data relating to the GT, IMU and camera are the following:

```
/camera
```

```
/world/baylands/model/x500_mono_cam_0/link/base_link/sensor/imu_sensor/imu
```

```
/model/x500_mono_cam_0/odometry
```

6.0.3 Gazebo

The PX4 autopilot supports several simulators and the one used in this thesis is Gazebo. Gazebo is a powerful open source simulation environment that allows the creation of realistic 3D models and environments to test and develop robots [48].

UAV model

Gazebo is used to simulate UAV sensors, including camera and IMU, which are needed to generate synthetic data used within the algorithm. For this thesis, the X500 Mono Cam model, part of the Software In The Loop (SITL) section of the PX4 autopilot, was selected. Vehicle models in Gazebo are defined using the .sdf format through which it is possible to adjust UAV parameters and add custom plugins. The X500 Mono Cam drone shown in the image below works with:

- IMU operating at 130 Hz
- camera with a frequency of 30 Hz
- GT system with a 100 Hz update rate

World model

A custom world is created with the goal of simulating an urban environment where the drone can be flown to collect data to test the algorithm. The base world used is the existing Baylands environment in Gazebo, which is then modified by adding urban elements such as buildings, houses, cars and other typical urban landscape elements. The resulting world is shown in Figure 6.2. These additions are made by modifying the original world's .sdf file, customizing it to better fit the needs of the project and create a realistic test environment for urban navigation.

Figure 6.1: X 500 mono cam drone model

Figure 6.2: Custom urban environment in Gazebo

6.0.4 Reference frames

In the context of using Gazebo and PX4 for simulation, one important consideration is the difference in coordinate frame conventions between the two systems, as defined in the following table [25].

Frame	**PX4**	**Gazebo**
Body	FRD (X Forward, Y Right, Z Down)	FLU (X Forward, Y Left, Z Up)
World	FRD or NED (X North, Y East, Z Down)	FLU or ENU (X East, Y North, Z Up)

Table 6.1: Coordinate Frame conventions in PX4 and Gazebo

The image below illustrates both frames, with Forward-Right-Down (FRD) on the left and Forward-Left-Up (FLU) on the right.

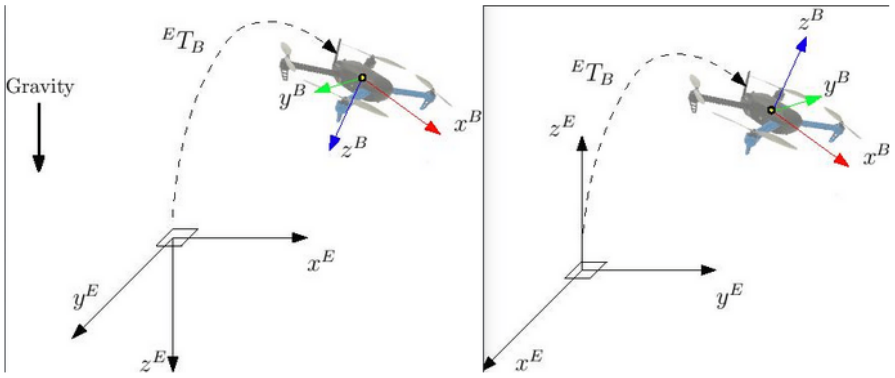


Figure 6.3: Reference frame of PX4 (left) and Gazebo (right) [25]

The two main transformations performed concern the IMU data and camera images, both of which are expressed in their respective Gazebo reference frames: the body frame for IMU data and the camera frame for camera images. The transformations are summarized as follows:

- **IMU Data:** The linear acceleration and angular velocity data from the IMU, initially in the Gazebo body frame (FLU), are transformed into the PX4 body frame (FRD) before being saved. This ensures that the data can later be integrated into the PX4 world frame for further processing.

- **Camera Images:** The camera images are published in matrix format in the associated topic. When the images are read and saved, the OpenCV `cv_bridge.imgmsg_to_cv2` function is used, which automatically converts the images to the default OpenCV reference frame. In this frame, the camera is oriented so that the Z-axis points forward, the Y-axis points down, and the X-axis points right.

These transformations allow the IMU data and camera images to be correctly aligned and integrated into the PX4 system for further algorithmic processing.

6.0.5 QGroundControl

QGroundControl is an open source software for GCSs that provides an interface for controlling UAVs and monitoring their flight status. It communicates with the UAV via ROS2 and PX4, allowing real-time monitoring of sensor data and flight parameters. In this project, QGroundControl was used to define the trajectories followed by the drone in the simulated environment inside Gazebo. By setting key mission parameters such as waypoints, takeoff sequences, orientation and initial starting point, this allowed precise management of the drone's path. The following Figure illustrates the QGroundControl configuration used for trajectory planning in the simulation.

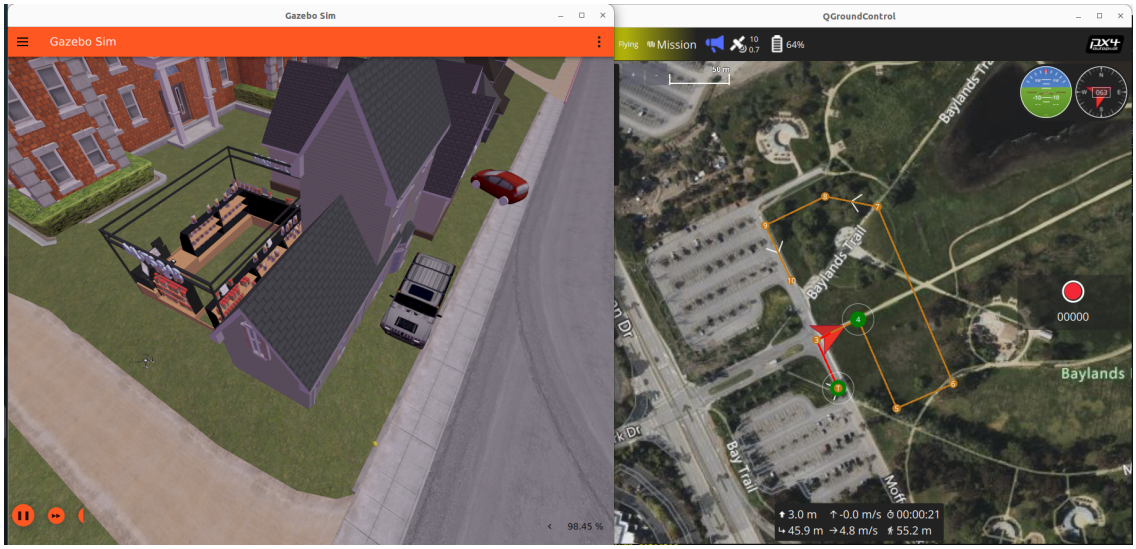


Figure 6.4: QGroundControl setup. The satellite view displayed does not represent the virtual environment, but the trajectory shown corresponds to the one executed in the simulation. This allows for monitoring the UAV's real-time position and movement during the simulation process.

Chapter 7

Synchronization

7.1 Time synchronization between PX4 and Gazebo

Time synchronization between PX4 and Gazebo is important to ensure accurate communication and data exchange within the system. A common challenge arises from the fact that Gazebo uses simulation time as a time source, which differs from the operating system clock used by PX4. To ensure correct data processing, it is critical to align these two time sources.

To synchronize Gazebo with PX4, Gazebo's clock must be used as a time source. This requires using the necessary ROS2 and Gazebo interface packages and configuring the `/clock` bridge via the `ros_gz_bridge` package.

```
ros2 run ros_gz_bridge parameter_bridge /clock@rosgraph_msgs
/msg/Clock[gz.msgs.Clock]
```

After setting up the bridge, the `use_sim_time` parameter must be set to `true` for each ROS2 node, signaling that Gazebo simulation time should be used. On the PX4 side, synchronization with uXRCE-DDS can be disabled by setting the `UXRCE_DDS_SYNCT` parameter to `false`, ensuring that simulation time becomes the sole time reference for both PX4 and Gazebo [25].

7.2 Data synchronization

The data from GT, IMU and the camera are time-aligned to ensure consistency across all sources. To achieve this synchronization, two key steps were followed. First, the IMU timestamp, which has the highest frequency among the available datasets, is used as the reference time. Then, the timestamps from the other sources, GT and camera, are aligned to those of the IMU. If an exact match is not found, the GT and camera timestamps are aligned with the closest available IMU timestamp.

This approach minimizes discrepancies caused by small time offsets between different sources. The final result is a unified dataset where the IMU, camera and GT data are all aligned in time.

Chapter 8

Implementation

In this chapter, the implementation of VIO using a loosely couple approach with an ES-EKF is discussed. The chapter begins with a brief description of coordinate frames involved and continues with the details of the filter design in its various steps.

The Figure 8.1 outlines the general structure and the workflow followed in the thesis.

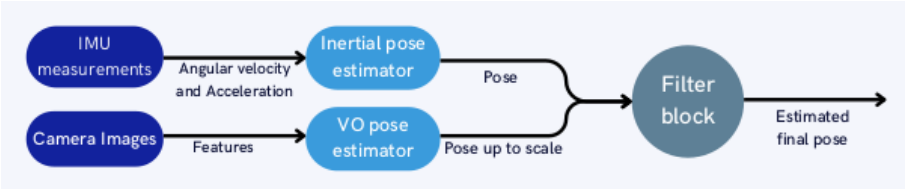


Figure 8.1: Loosely coupled approach VIO

The VO algorithm estimates the UAV's pose relative to the surrounding environment by processing visual data. At the same time, the trajectory is computed using only the data from the IMU, which provides measurements of linear accelerations a_m and angular velocities ω_m.

To create an integrated navigation solution, pose estimates from both the camera and the IMU are converted into a unified navigation reference frame, known as the World Frame and are then combined using an ES-EKF framework.

8.1 Coordinate Frames

The Visual-Inertial system includes three main coordinate systems, shown in the image below. Understanding these systems and the transformations between them is important in the implementation of the algorithm and for effective filter design.

- **World Frame:** The World frame serves as a static reference frame for estimating the drone's motion. This frame is defined according to the NED convention:

  +X: north

  +Y: east

+Z: down

- **Body Frame:** The body frame coincide with the IMU frame, it is rigidly attached to the drone body and moves along with it. The measurements from the IMU are expressed in this coordinate system, which follows the FRD convention:

  +X: forward

  +Y: right

  +Z: down

- **Camera Frame:** The Camera frame describes the coordinate system of the camera, which moves along with the vehicle. The Camera coordinate frame follows the default reference frame established by OpenCV:

  +X: right

  +Y: down

  +Z: forward

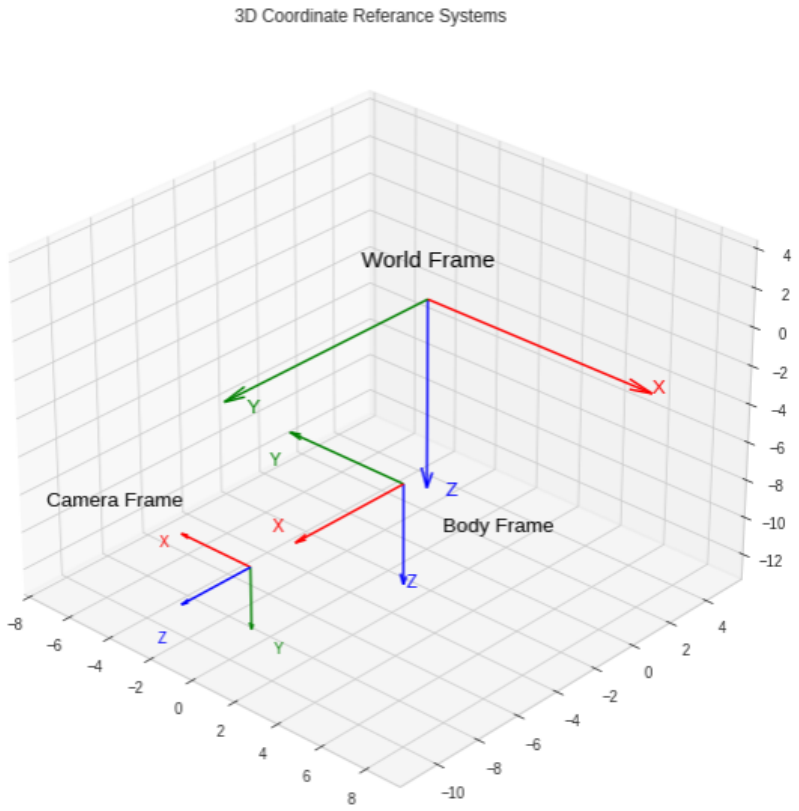


Figure 8.2: Reference systems

8.2 Error State Extended Kalman Filter design

This section describes the key procedures involved in implementing the ES-EKF filter; the following image illustrates the main steps taken during the implementation [24].

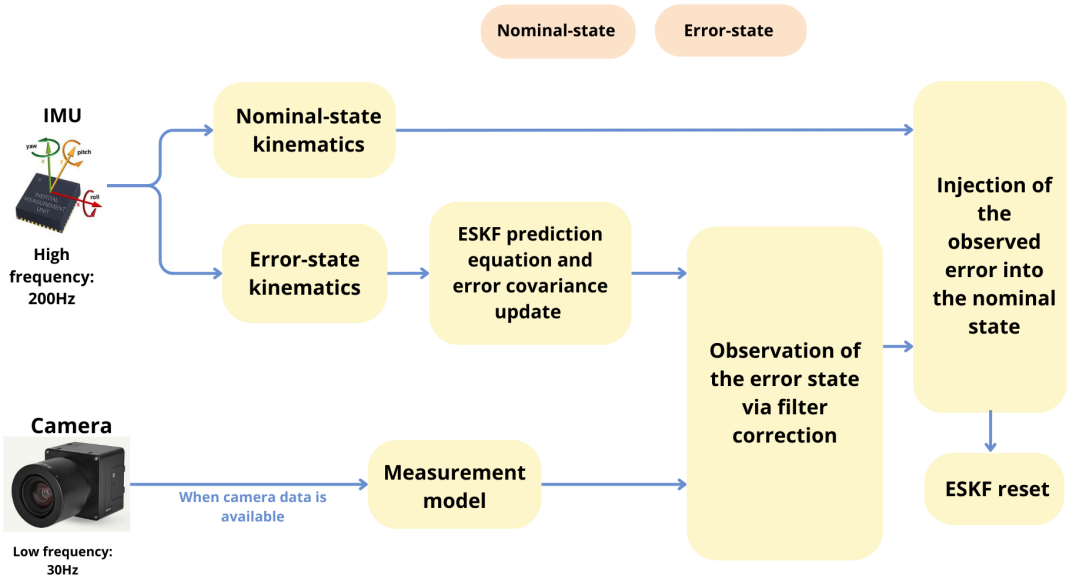


Figure 8.3: ES-EKF pipeline

8.2.1 True-state kinematics in continuous time

The filter's states include the position of the IMU in the World frame, denoted as p_w^i, its velocity in the World frame v_w^i and the attitude quaternion q_w^i, which describes the rotation from the IMU frame the World frame. Additionally, it accounts for the accelerometer and gyro biases, b_a and b_w, as well as a scaling factor λ that relates the trajectory obtained from the VO to the GT trajectory.

$$x = \begin{bmatrix} p_w^i & v_w^i & \theta_w^i & b_a & b_w & \lambda \end{bmatrix}$$

The drone's motion, determined by the IMU sensor, is described by a series of navigation equations that receive as inputs from the sensor the linear acceleration a_m and angular velocity ω_m. Additionally, the system is affected by various noise types, including acceleration noise a_n, gyro noise ω_n, acceleration bias noise a_w and gyro bias noise ω_w.

The true kinematic equations and bias model for the IMU are formulated as follows:

$$\dot{p}_w^i = v_w^i \tag{8.1}$$
$$\dot{v}_w^i = C_{q_w^i}(a_m - a_b - a_n) + g \tag{8.2}$$
$$\dot{q}_w^i = \frac{1}{2} q_w^i \otimes (\omega_m - \omega_b - \omega_n) \tag{8.3}$$
$$\dot{a}_b = a_w \tag{8.4}$$
$$\dot{\omega}_b = \omega_w \tag{8.5}$$
$$\dot{\lambda} = 0 \tag{8.6}$$

where $C_{q_w^i}$ represents the rotation matrix that transforms a vector from the Inertial frame to the World frame.

8.2.2 State kinematics in discrete time

The differential equations presented in the problem above require transformation into difference equations to effectively accommodate the concept of discrete time intervals. To achieve this transformation, the integration technique applied is the Euler method. Integration must be performed for both the nominal state and the error state of the system. For the error state, two main components are integrated: the deterministic part, which includes the state dynamics and the stochastic part, which accounts for noise and disturbances. The integration of both components ensures complete capture of system dynamics.

The nominal state kinematics

The differences equations of the nominal-state can be rewritten as:

$$p_w^i \leftarrow p_w^i + v_w^i \Delta t + \frac{1}{2}\left(C_{q_w^i}(a_m - a_b) + g\right)\Delta t^2 \tag{8.7}$$
$$v_w^i \leftarrow v_w^i + \left(C_{q_w^i}(a_m - a_b) + g\right)\Delta t \tag{8.8}$$
$$q_w^i \leftarrow q_w^i \otimes q_w^i \left\{(\omega_m - \omega_b)\,\Delta t\right\} \tag{8.9}$$
$$a_b \leftarrow a_b \tag{8.10}$$
$$\omega_b \leftarrow \omega_b \tag{8.11}$$
$$\lambda \leftarrow \lambda \tag{8.12}$$

where $x \leftarrow f(x, \cdot)$ stands for a time update of the type $x_{k+1} = f(x_k, \cdot_k)$ and q_v is the quaternion associated with the rotation v.

The error-state kinematics

Error-state kinematics describe the accumulation of errors in a state estimation process. In the error-state equation, the deterministic component is integrated as usual, while the integration of the stochastic component results in random fluctuations.

$$\delta p_w^i \leftarrow \delta p_w^i + \delta v \Delta t \tag{8.13}$$
$$\delta v_w^i \leftarrow \delta v_w^i + \left(-C_{q_w^i} \left[a_m - a_b\right]_x \delta\theta_w^i - R\delta a_b + \delta g\right) \Delta t + v_i \tag{8.14}$$
$$\delta\theta_w^i \leftarrow C_{q_w^i}^T \left\{\left(\omega_m - \omega_b\right) \Delta t\right\} \delta\theta_w^i - \delta\omega_b \Delta t + \theta_i \tag{8.15}$$
$$\delta a_b \leftarrow \delta a_b + a_i \tag{8.16}$$
$$\delta\omega_b \leftarrow \delta\omega_b + \omega_i \tag{8.17}$$
$$\delta\lambda \leftarrow \delta\lambda \tag{8.18}$$

Here, v_i, θ_i, a_i, and ω_i are the random impulses applied to the velocity, orientation and bias estimates, modeled as white Gaussian processes. Their mean is zero and their covariance matrices are obtained by integrating the covariances of a_n, ω_n, a_w, and ω_w over the time step Δt.

$$V_i = \sigma^2 \tilde{a}_n \Delta t^2 I \quad [\mathrm{m}^2/\mathrm{s}^2] \tag{8.19}$$
$$\Theta_i = \sigma^2 \tilde{\omega}_n \Delta t^2 I \quad [\mathrm{rad}^2] \tag{8.20}$$
$$A_i = \sigma^2 a_w \Delta t I \quad [\mathrm{m}^2/\mathrm{s}^4] \tag{8.21}$$
$$\Omega_i = \sigma^2 \omega_w \Delta t I \quad [\mathrm{rad}^2/\mathrm{s}^2] \tag{8.22}$$

8.2.3 Prediction Step

When implementing the Kalman filter framework, it is essential to linearize the system. This requires computing Jacobian matrices at each time step to create a locally linearized model based on the current state estimate.

The relevant variables can be expressed in compact form by defining the nominal state vector x, the error state vector δx, the input vector u_m and the perturbation impulses vector i as follows:

$$x = \begin{bmatrix} p \\ v \\ q \\ a_b \\ \omega_b \\ \lambda \end{bmatrix}, \quad \delta x = \begin{bmatrix} \delta p \\ \delta v \\ \delta\theta \\ \delta a_b \\ \delta\omega_b \\ \delta\lambda \end{bmatrix}, \quad u_m = \begin{bmatrix} a_m \\ \omega_m \end{bmatrix}, \quad i = \begin{bmatrix} v_i \\ \theta_i \\ a_i \\ \omega_i \end{bmatrix}$$

The error-state linearized system is defined as:

$$\delta x \leftarrow f(x, \delta x, u_m, i) = F_x(x, u_m) \cdot \delta x + F_i \cdot i \tag{8.23}$$

while the prediction equations are given by:

$$\hat{\delta x} \leftarrow F_x(x, u_m) \cdot \hat{\delta x} \tag{8.24}$$

$$P \leftarrow F_x P F_x^T + F_i Q_i F_i^T \tag{8.25}$$

where $\delta x \sim \mathcal{N}(\hat{\delta x}, P)$; F_x and F_i represent the Jacobians of the inertial equation with respect to the error state and perturbation vectors, respectively, and Q_i is the

covariance matrix of the perturbation impulses.

In the ES-EKF prediction equations, the predicted error state $\hat{\delta x}$ is based on the previous estimate and the system's dynamics, focusing on how the state changes with nominal inputs. The matrix F_i is not included in the prediction of $\hat{\delta x}$ because it doesn't account for perturbations. However, in the covariance update P, F_i is included to consider the uncertainty from these disturbances. Essentially, $\hat{\delta x}$ models the state progression, while the covariance update accounts for both state changes and external uncertainty.

Another important consideration in the prediction step is that the error state vector, δx, is initially set to zero. This assumption implies that the initial estimate of the state is considered perfect. The prediction equation 8.24 is used to update the error state. However, since the error state δx is initialized to zero, this equation will always result in zero, meaning there is no actual update to the error state during the prediction step.

The error-state Jacobian and perturbation matrices

The Jacobian and covariance matrices mentioned above are described below.

$$F_x = \left.\frac{\partial f}{\partial \delta x}\right|_{x,u_m} = \begin{bmatrix} I_3 & \Delta t I_3 & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times1} \\ 0_{3\times3} & I_3 & -\Delta t C_{q_w^i}[a_m - a_b]_\times & -\Delta t C_{q_w^i} & 0_{3\times3} & 0_{3\times1} \\ 0_{3\times3} & 0_{3\times3} & C_{q_w^i}^T\{(\omega_m - \omega_b)\Delta t\} & 0_{3\times3} & -\Delta t I_3 & 0_{3\times1} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & I_3 & 0_{3\times3} & 0_{3\times1} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & I_3 & 0_{3\times1} \\ 0_{1\times3} & 0_{1\times3} & 0_{1\times3} & 0_{1\times3} & 0_{1\times3} & 1 \end{bmatrix},$$

$$F_i = \left.\frac{\partial f}{\partial i}\right|_{x,u_m} = \begin{bmatrix} 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ I_3 & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & I_3 & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & I_3 & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & I_3 \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \end{bmatrix} \qquad Q_i = \begin{bmatrix} V_i & 0 & 0 & 0 \\ 0 & \Theta_i & 0 & 0 \\ 0 & 0 & A_i & 0 \\ 0 & 0 & 0 & \Omega_i \end{bmatrix}$$

8.2.4 Correction step

Until now, IMU data has been used to make predictions in the ES-EKF. Considering that the IMU and the camera operate at different frequencies, the update procedure is triggered every time the VO algorithm generates a visual measurement, allowing additional information to correct the filter.

The correction process involves three key steps:

1. Observing the error state through filter correction
2. Injecting the observed errors into the nominal state
3. Resetting the error state

Measurement Model

In the VIO system, the measurement model defines the relationship between the measured quantities and the system state, integrating both visual and inertial data.

The camera position z_p, representing the position of the body in the world frame frame p_{WI}, is given by:

$$z_p = p_{\text{WI}}^{\text{vo}} = p_w^i \lambda + n_{zp} \tag{8.26}$$

where n_{zp} denotes the position measurement noise.

Similarly, the camera orientation z_q is defined as:

$$z_q = q_{\text{WI}}^{\text{vo}} = q_w^i + n_{zq} \tag{8.27}$$

where q_w^i represents the orientation of the body frame with respect to the world frame and n_{zq} is the orientation measurement noise.

Jacobian computation for the filter correction

To update the state estimate, the residual between the measured state and the propagated state is computed:

$$\tilde{z} = z \ominus \hat{z} = \begin{bmatrix} p_{\text{WI}}^{\text{vo}} - p_{\text{WI}}^{\text{imu}} \\ q_{\text{WI}}^{\text{vo}} \otimes q_{\text{WI}^{-1}}^{\text{imu}} \end{bmatrix} \tag{8.28}$$

The residual associated to the orientation can be rewritten as:

$$\mathbf{q}_{\text{WI}}^{\text{vo}} \otimes \mathbf{q}_{\text{WI}^{-1}}^{\text{imu}} \approx \begin{bmatrix} 1 \\ -0.5\boldsymbol{\theta}^\top \end{bmatrix} \tag{8.29}$$

Next, the Jacobian matrix H is defined with respect to the error state δx and evaluated at the best true-state estimate $\hat{x}_t = x \oplus \hat{\delta x}$. Since the error state mean is initially set to zero, $\hat{x}_t \approx x$, allowing the nominal error x to be used as the evaluation point:

$$H \equiv \left. \frac{\partial h}{\partial \delta x} \right|_x$$

The two equation 8.26 and 8.27 are linearized with respect to the error-state in order to get the Jacobian of measured observation:

$$\tilde{z} = H\tilde{x} = \begin{bmatrix} \lambda I_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & p_w^i \\ 0_{3\times3} & 0_{3\times3} & I_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times1} \end{bmatrix} \tilde{x} \tag{8.30}$$

where $\tilde{x} \in R^{16\times1}$.

The Jacobian with respect to the measurement noise is as follows:

$$M_k = \begin{bmatrix} I_{3\times3} & 0_3 \\ 0_3 & I_{3\times3} \end{bmatrix} \tag{8.31}$$

Finally, the measurement noise covariance matrix R_k is defined as:

$$R_k = \begin{bmatrix} \sigma_{nzp}^2 & 0_{3\times4} \\ 0_{4\times3} & \sigma_{nzq}^2 \end{bmatrix} \tag{8.32}$$

Update data

Once the H_k and M_k matrices are determined, the state estimates are updated using the following equation:

$$\text{Innovation:} \quad \tilde{z}_k = z_k - h(\hat{x}_{k-1}) \tag{8.33}$$

$$\text{Innovation covariance:} \quad S_k = H_k P_{k|k-1} H_k^T + M_k R_k M_k^T \tag{8.34}$$

$$\text{Kalman gain:} \quad K_k = P_{k|k-1} H_k^T S_k^{-1} \tag{8.35}$$

$$\text{Correction:} \quad \tilde{x}_k = K_k \tilde{z}_k \tag{8.36}$$

Finally, the state covariance matrix is updated using:

$$P_{k|k} = (I - K_k H_k) P_{k|k-1} (I - K_k H_k)^T + K_k M_k R_k M_k^T K_k^T \tag{8.37}$$

8.2.5 Injection of the observed error into the nominal state

After the update of the ES-EKF, the nominal state is modified by incorporating the observed error state. This is represented as:

$$x \leftarrow x \oplus \hat{\delta x}$$

that is

$$p \leftarrow p + \hat{\delta p} \tag{8.38}$$
$$v \leftarrow v + \hat{\delta v} \tag{8.39}$$
$$q \leftarrow q \otimes q\{\hat{\delta\theta}\} \tag{8.40}$$
$$a_b \leftarrow a_b + \hat{\delta a_b} \tag{8.41}$$
$$\omega_b \leftarrow \omega_b + \hat{\delta\omega_b} \tag{8.42}$$
$$\lambda \leftarrow \lambda + \hat{\delta\lambda} \tag{8.43}$$

8.2.6 ESKF reset

Since the error has been corrected when it is injected into the nominal state, the error mean $\hat{\delta x}$ is reset to zero to prevent incorrect error propagation in subsequent iterations. To complete the ES-EKF update, the error covariance needs to be updated to reflect this modification. The error reset operation is therefore given by:

$$\hat{\delta x} \leftarrow 0 \tag{8.44}$$

$$P \leftarrow GPG^T \tag{8.45}$$

where, in the majority of cases, $G = I_{16\times16}$.

Chapter 9

Results

The chapter begins with an analysis of the results, initially examining the performance of the VO algorithm and the IMU integration separately. Then, the information from both sensors is fused within the filter, allowing a complete evaluation of the system. The strengths and weaknesses of each sensor are highlighted to optimize the overall performance and identify the best configuration. Finally, the chapter concludes with a discussion on the system's ability to recover scale.

9.1 Visual Odometry implementation and results

The implementation process and results of VO are defined in this section. Following a theoretical examination of various methodologies, the decision was made to adopt a feature-based approach, as it is considered the most suitable for the objectives of this thesis.

The algorithm, along with the distinct stages of VO, is executed employing OpenCV functions.

9.1.1 Feature detection

The first step in VO process is feature detection. The SIFT algorithm is employed for extracting feature vectors that describe local patches of an image which are not only invariant to scale but also to translation and rotation. The SIFT algorithm consists of two main successive and independent operations: the detection of interesting points, known as keypoints, and the extraction of a descriptor associated with each keypoint.

SIFT is composed of four primary steps:

1. Scale-Space extrema detection
2. Keypoint localization
3. Orientation assignment
4. Local descriptor creation

SIFT detects a series of keypoints from a multiscale image representation, which consists of a family of increasingly blurred images. Each keypoint is characterized as a blob-like structure, with its center position (x, y) and characteristic scale σ accurately located. Additionally, SIFT computes the dominant orientation θ over a region surrounding each keypoint. Thus, for each keypoint, the quadruple (x, y, σ, θ) defines the center, size and orientation of a normalized patch where the SIFT descriptor is computed. The descriptor encodes the spatial gradient distribution around a keypoint as a 128-dimensional vector, which is generally used to match keypoints extracted from different images.

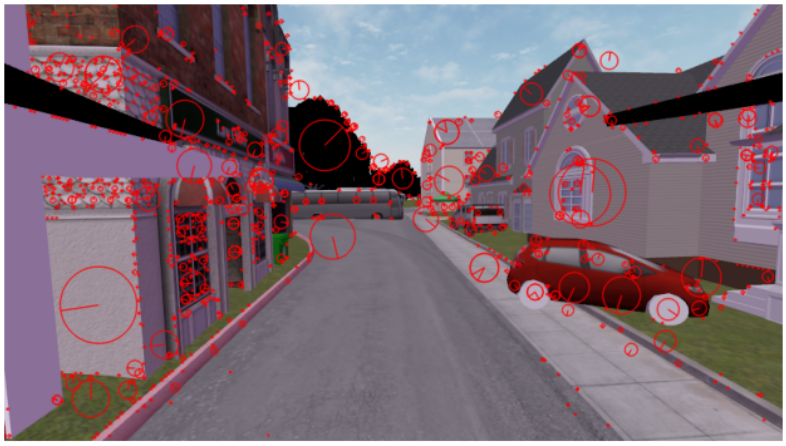


Figure 9.1: SIFT keypoints detection

In Figure 9.1, the image shows the features detected by SIFT. The circles represent the identified keypoints, with the center of each circle indicating the location of the feature in the image. The size of the circle corresponds to the scale at which the feature was detected, with larger circles indicating larger features and smaller ones representing finer details. Additionally, the orientation of each feature is also represented.

9.1.2 Feature Matching

After detecting features, the next step involves feature matching between successive images. For this purpose, the Brute Force Matcher (BFMatcher) is employed, which compares the SIFT descriptors using Euclidean distance. This method computes the distance between each descriptor in the first image and all descriptors in the second image, allowing for the identification of potential matches based on similarity.

To improve the reliability of the matches, a k-nearest neighbors (k-NN) approach is utilized, specifically looking for the two best matches for each descriptor. By retrieving two matches for each keypoint, the algorithm can apply a ratio test, as proposed by Lowe in the original SIFT paper [49]. This test involves comparing the distance of the closest match to the distance of the second closest match, if the ratio is below a specified threshold, the first match is considered a good match, helping to filter out ambiguous correspondences and reducing the impact of potential outliers.

The results of the feature matching process are illustrated in the following images, where corresponding points between successive images are connected by lines.

Figure 9.2: Result of feature matching between images

However, despite the application of the ratio test in this initial phase, recurring patterns in the images can lead to situations where one feature may be confused for another, resulting in false matches. These erroneous correspondences will be addressed in the subsequent phase using RANSAC, which effectively identifies and eliminates outliers based on geometric consistency. By examining only a portion of all the matched pairs, it becomes possible to identify incorrect matches, which are highlighted in red.

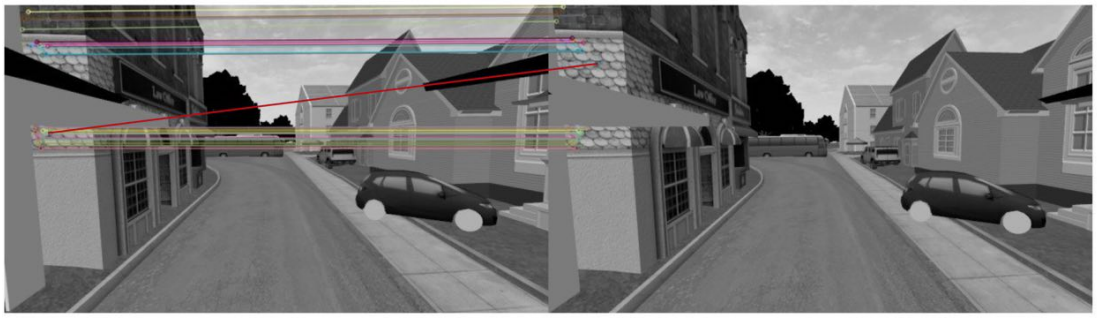


Figure 9.3: Recurring patterns in the images can cause confusion between similar features, leading to false matches

9.1.3 Motion Estimation

Once feature matching is completed, the next step is motion estimation. A 2D-to-2D motion estimation approach is employed, which computes the camera's movement based on corresponding points between consecutive images. The implementation utilizes OpenCV's built-in functionality to estimate the essential matrix using the five-point algorithm, allowing for the calculation of the relative rotation and translation between the two images. This algorithm is based in epipolar geometry and on the epipolar constraint, which states that corresponding points in the two images must lie along their respective epipolar lines.

Because of the way the chosen trajectory is defined, the camera moves along the optical axis. This represents a special case of epipolar geometry, where the epipoles

in both images coincide at the same point, known as the focus of expansion. This happens because the translation along the optical axis does not alter the relative orientation of the two cameras, it only affects their position along the axis, leaving the horizontal and vertical alignment unchanged. As a result, the epipoles appear in the same location in both images. This situation is illustrated in 9.4, which shows the relationship between the two images and the coincident epipoles at the focus of expansion. As can be observed, the camera movement creates a radial pattern of epipolar lines in both images, so points in the 3D scene, when projected onto the two image planes, will have corresponding epipolar lines radiating outwards from the epipole.

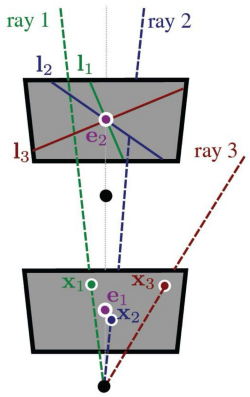


Figure 9.4: Special case of epipols [26]

To enhance the robustness of the motion estimation, the RANSAC method was applied. RANSAC effectively identifies and eliminates outliers, ensuring that only the most reliable key points contribute to the computation of the essential matrix.

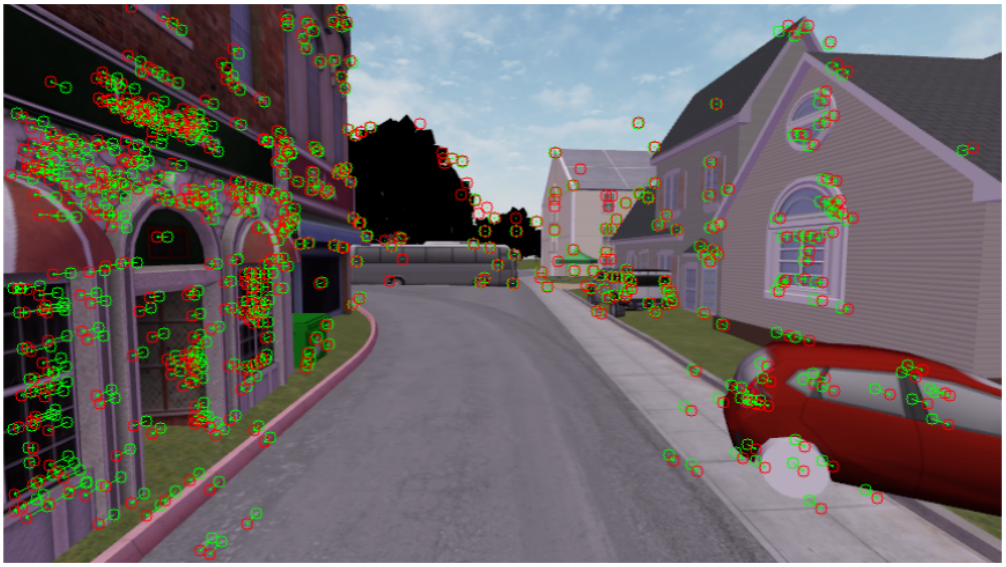


Figure 9.5: Visualization of "motion flow"

The image in Figure 9.5 displays the results of the motion estimation algorithm. Red circles mark the initial positions of the keypoints, while green circles show

their final locations. Green arrows connect these points, illustrating the displacement vectors and creating a clear visualization of "motion flow". This depiction effectively shows how features change due to the drone's estimated movement.

The arrangement of these motion vectors is particularly insightful, as it reflects the calculated motion parameters, specifically the rotation and translation of the UAV. This visualization not only demonstrates how the algorithm predicts feature trajectories but also highlights the spatial relationships between matched features and their expected movements.

9.1.4 Final Trajectory

The final step of VO is to combine the transformations obtained between consecutive image frames in order to construct the trajectory.

At time step $t = 0$, the transformation matrix between the world frame and the camera sensor is denoted as $\boldsymbol{T}_{wc}$. To derive the final trajectory, it involves combining the transformation matrices at each time step as follows:

$$\boldsymbol{T}_{wT} = \boldsymbol{T}_{wc}\boldsymbol{T}_{cc_1}\boldsymbol{T}_{c_1c_2}\cdots\boldsymbol{T}_{c_fT}$$

Here, T is the final time step. Additionally, since this final transformation is between the world frame and the camera, to obtain the trajectory with respect to the body frame (where the GT is provided), the combined transformation at each time step is multiplied by T_{cb}.

As shown in the figure above, after estimating the motion, the camera's final trajectory using the collected data is plotted. This trajectory represents the actual path the camera followed as it moved through the environment.

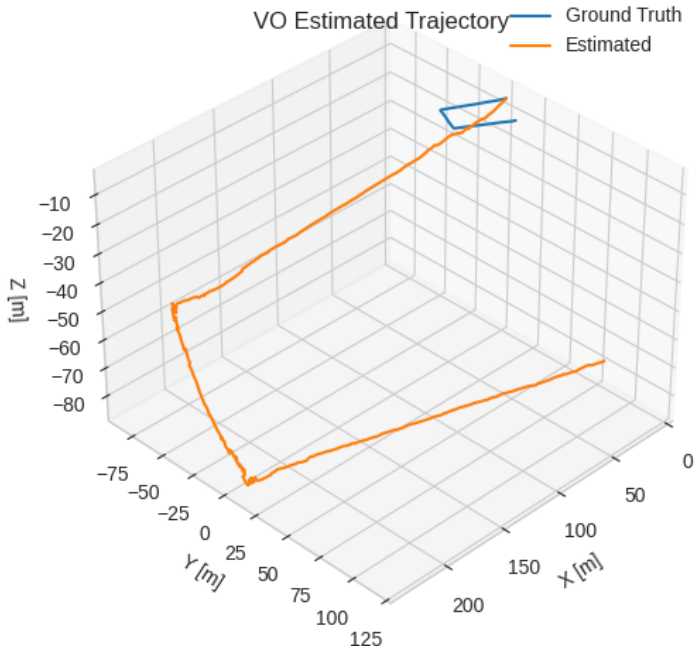


Figure 9.6: VO final trajectory

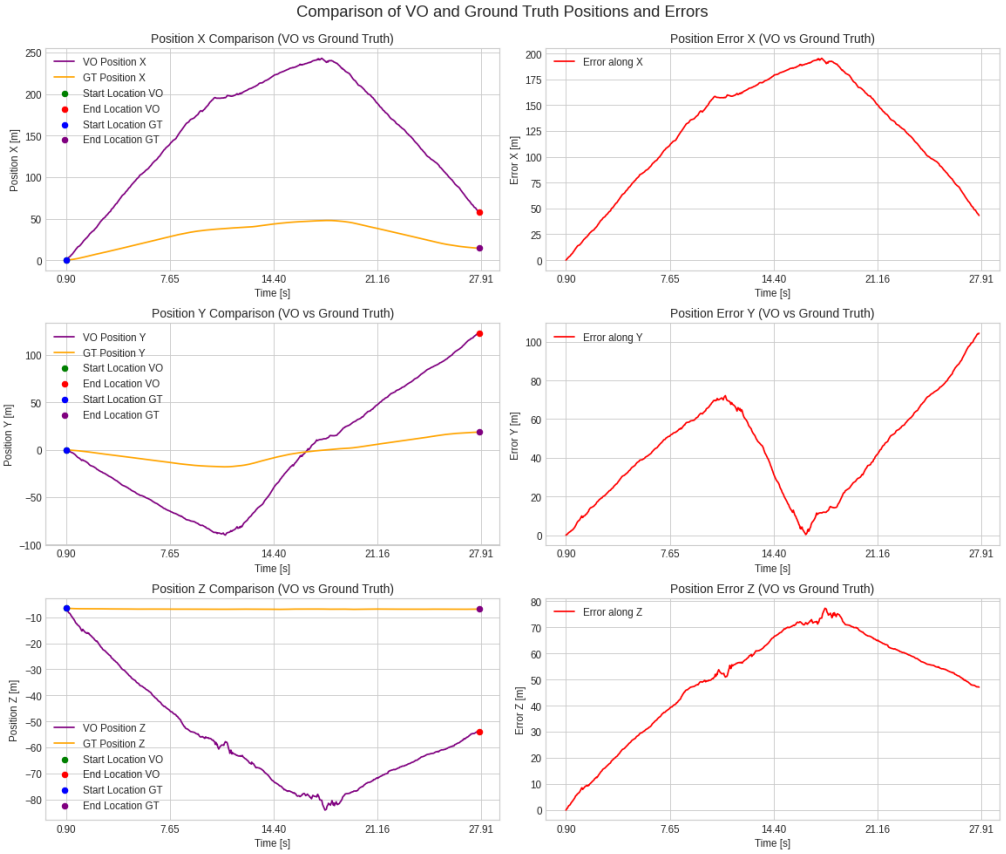


Figure 9.7: Comparison of VO and Ground Truth positions and errors over time

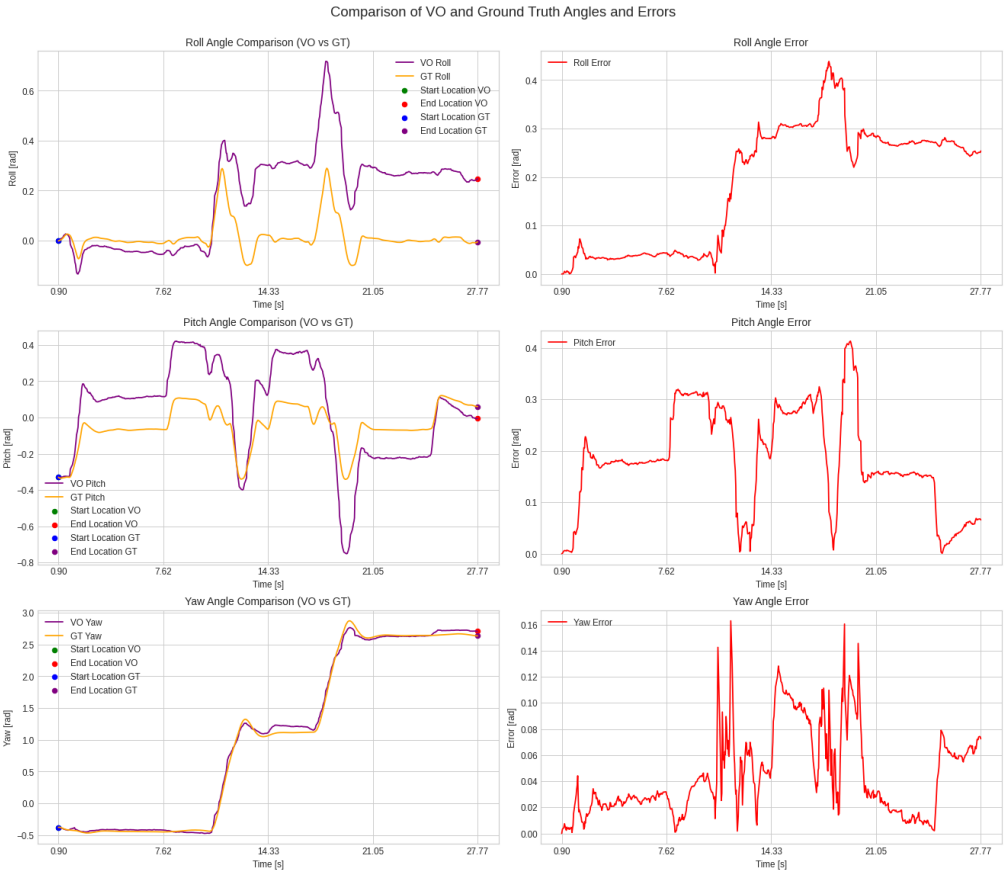


Figure 9.8: Comparison of VO and Ground Truth angles and errors over time

The resulting images indicate a discrepancy in scale between the GT and the estimated trajectory due to the use of a monocular camera. Several factors may contribute to the inaccuracy of the trajectory, including the scale ambiguity inherent in monocular systems, sensor noise and potential drift over time. Furthermore, the camera's orientation relative to the environment, the type of trajectory followed and the limitations of the feature extraction algorithm all play a role in affecting the precision of the trajectory estimation.

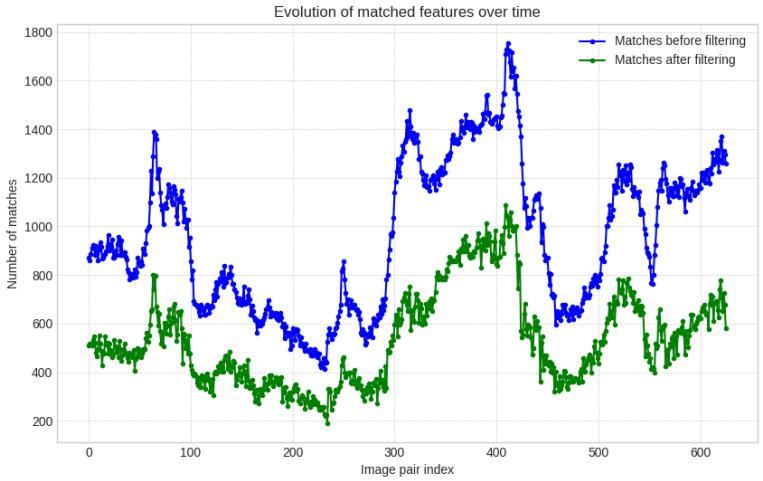


Figure 9.9: Trend of the number of matching features on the image pair index

As shown in image 9.9, which illustrates the trend of both unfiltered and filtered matches after the ratio test, a significant drop in the number of matches occurs around image indices 250 and 550, corresponding to the two curves the drone follows. Despite this reduction, the camera trajectory is still estimated accurately. This is because, although fewer matches remain after filtering, the remaining keypoints are well distributed and reliable, enabling the motion estimation algorithm to effectively track the camera's movement, even during curved sections of the path.

Computation scale factor between VO and Ground truth trajectory

To calculate the scale factor, a common segment of the trajectory is analyzed for both the VO and GT, as shown in the image below. First, the distance traveled along the same segment is measured for both trajectories. The scale factor is then determined by taking the ratio of the distance covered in the VO segment to the distance covered in the corresponding GT segment. This ratio aligns the VO trajectory with the GT trajectory, except for the drift or any error present.

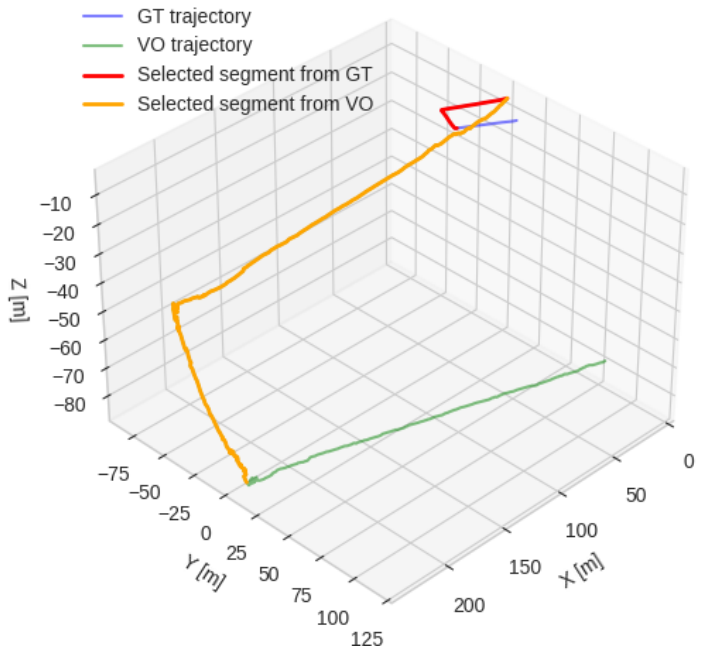


Figure 9.10: Selected segment from GT and VO for scale ratio calculation

Cumulative distance for GT segment: For each consecutive pair of points in the selected GT segment, the distance is calculated as:

$$d_{\text{GT}}(i) = \sqrt{(x_{\text{GT}}(i+1) - x_{\text{GT}}(i))^2 + (y_{\text{GT}}(i+1) - y_{\text{GT}}(i))^2 + (z_{\text{GT}}(i+1) - z_{\text{GT}}(i))^2}$$

Then, the total distance traveled along the GT segment is the sum of all these pairwise distances:

$$D_{\text{GT}} = \sum_{i=\text{start}}^{\text{end}-1} d_{\text{GT}}(i)$$

Cumulative Distance for VO segment: Similarly, the distance for each consecutive pair of points in the VO trajectory is calculated as:

$$d_{\text{VO}}(i) = \sqrt{(x_{\text{VO}}(i+1) - x_{\text{VO}}(i))^2 + (y_{\text{VO}}(i+1) - y_{\text{VO}}(i))^2 + (z_{\text{VO}}(i+1) - z_{\text{VO}}(i))^2}$$

The total distance traveled along the VO segment is the sum of these distances:

$$D_{\text{VO}} = \sum_{i=\text{start}}^{\text{end}-1} d_{\text{VO}}(i)$$

Scale Factor calculation: The scale factor is computed as the ratio of the total distance traveled in the VO segment to the total distance in the GT segment:

$$\text{Scale} = \frac{D_{\text{VO}}}{D_{\text{GT}}} = 6.17$$

By dividing each component of the VO trajectory by the calculated scale ratio, the scale between the VO and GT trajectories is recovered, as shown in Figure 9.11. This approach reveals the actual error associated with VO, where, once the scale error is eliminated, the remaining error reflects the intrinsic inaccuracies of the VO algorithm. In VO, the vehicle's current pose is determined by concatenating transformations between successive frames. Since each transformation is prone to errors, the accuracy of the current pose depends on the errors from previous transformations. As the system progresses, these errors propagate and accumulate over time, resulting in drift.

Additionally, calculating an indicative scale value is important to initialize the scale within the filter, however, this topic will be explored further in the next chapter.

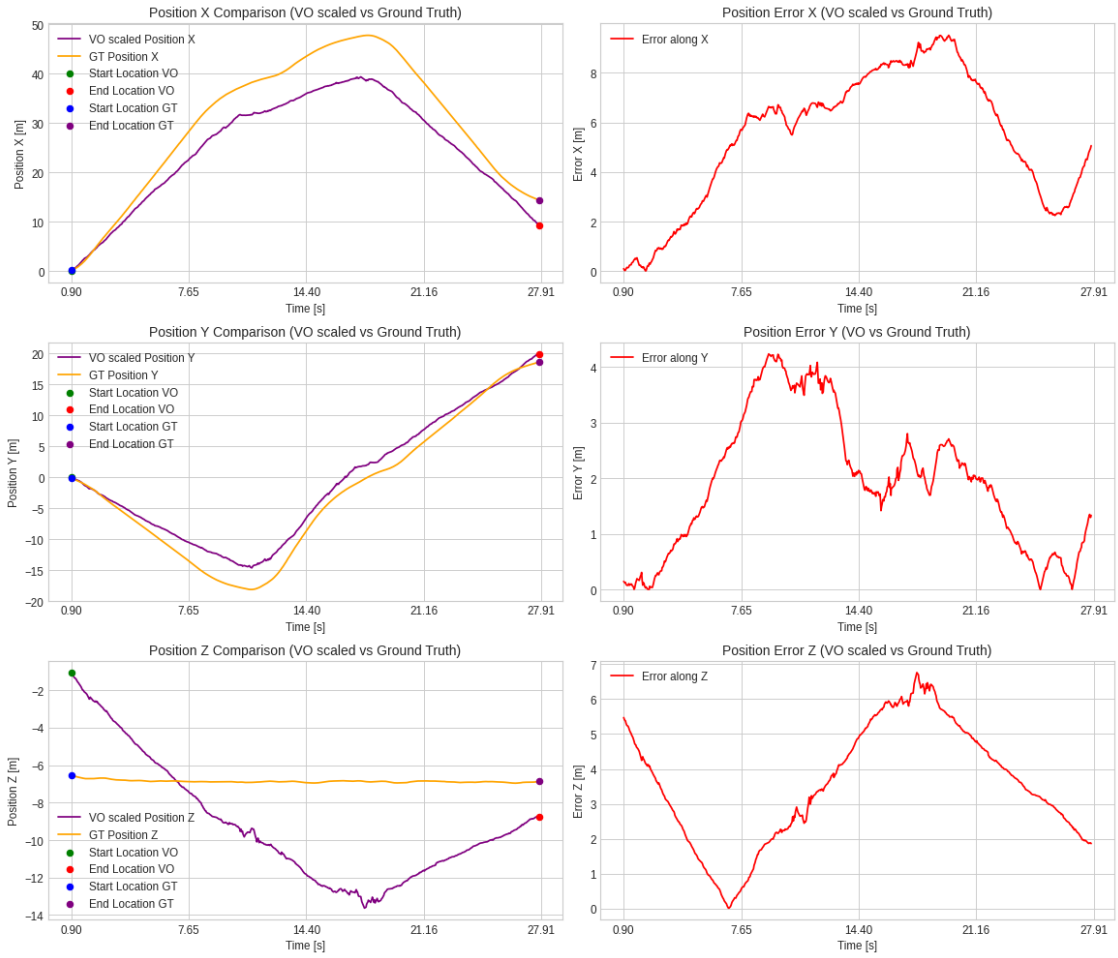


Figure 9.11: Scaled VO positions and corresponding errors over time

Drift along vertical direction

Even if the scaling factor is applied, the trajectory along the z axis is very different from the GT, as shown in the Figure 9.12.

In trajectory estimation using monocular VO, this mainly occur because a single camera does not have direct depth perception of objects within a scene. Without accurate depth information, the system struggles to accurately estimate the distance

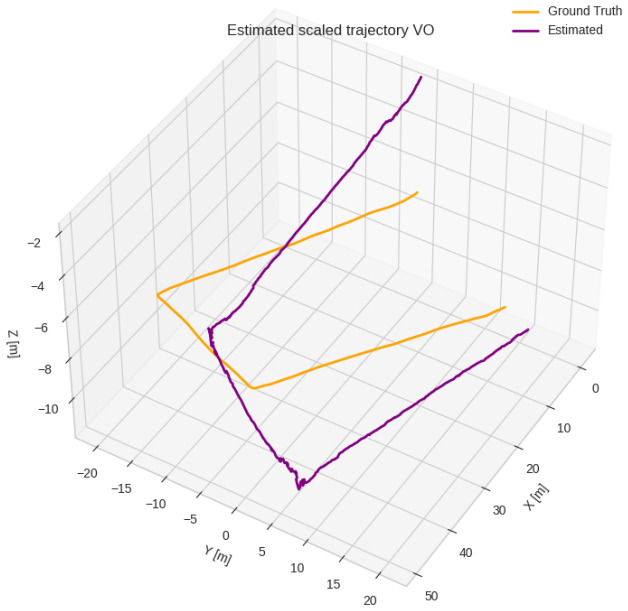


Figure 9.12: Estimated scaled trajectory VO

to objects, hindering its ability to correctly compute camera motion along the vertical direction. When navigating scenes with limited vertical landmarks, small errors in horizontal motion estimation can be misinterpreted as vertical displacements, leading to vertical drift. This problem becomes especially evident in smooth or linear contexts, such as straight roads, where the lack of distinct landmarks for depth correction results in compounded inaccuracies in the predicted trajectory.

9.2 IMU integration

The results of integrating data from IMU are analyzed and discussed in this section.

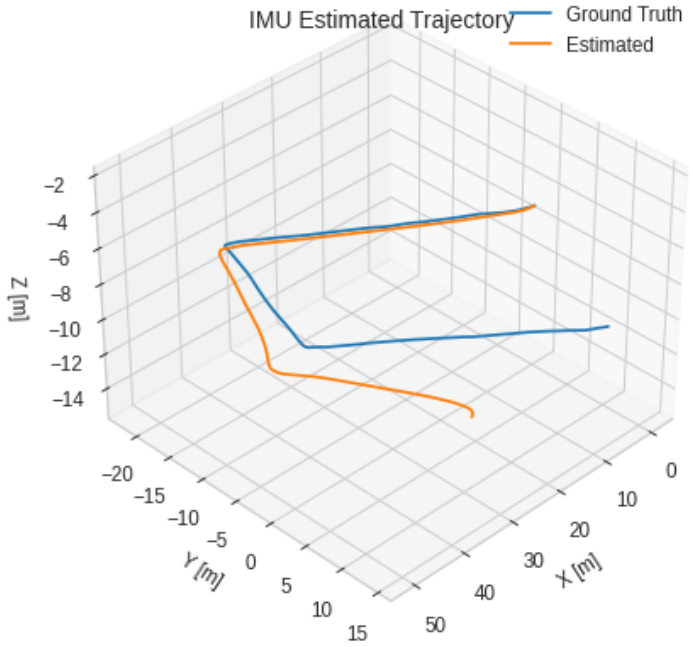


Figure 9.13: Comparison of the trajectory estimated from IMU integration with GT. The plot highlights the drift over time when relying solely on IMU data for trajectory estimation, showing a significant divergence from the ground truth as the system progresses.

Figure 9.13 presents the trajectory estimated using only the IMU data, compared to the GT trajectory. The integration of the IMU provides an estimation of the position and velocity and orientation of UAV, but one of the key challenges when relying solely on the IMU for trajectory estimation is the accumulation of drift over time. As shown in the graph, the estimated trajectory begins to diverge significantly from the GT after a certain period, primarily due to the cumulative error in the IMU measurements, particularly in the accelerometer and gyroscope readings.

This drift is a well-known limitation of inertial navigation systems when no external corrections or additional sensors are used. Even small errors in the sensor readings can accumulate rapidly, leading to significant deviations in the estimated trajectory. The effect is particularly noticeable over longer time periods, where the divergence from the GT becomes more pronounced.

The behavior of the IMU is more clearly highlighted in Figure 9.14, where the graphs show the time evolution of position estimates along the x, y and z axes, derived from the IMU integration, compared to the corresponding GT positions along the same axes. The respective error trends are shown on the right-hand side of the Figures. These graphs clearly illustrate how the IMU-derived position estimate progressively diverges from the GT due to the accumulation of drift over time. A similar phenomenon is observed in the orientation estimates in Figure 9.15.

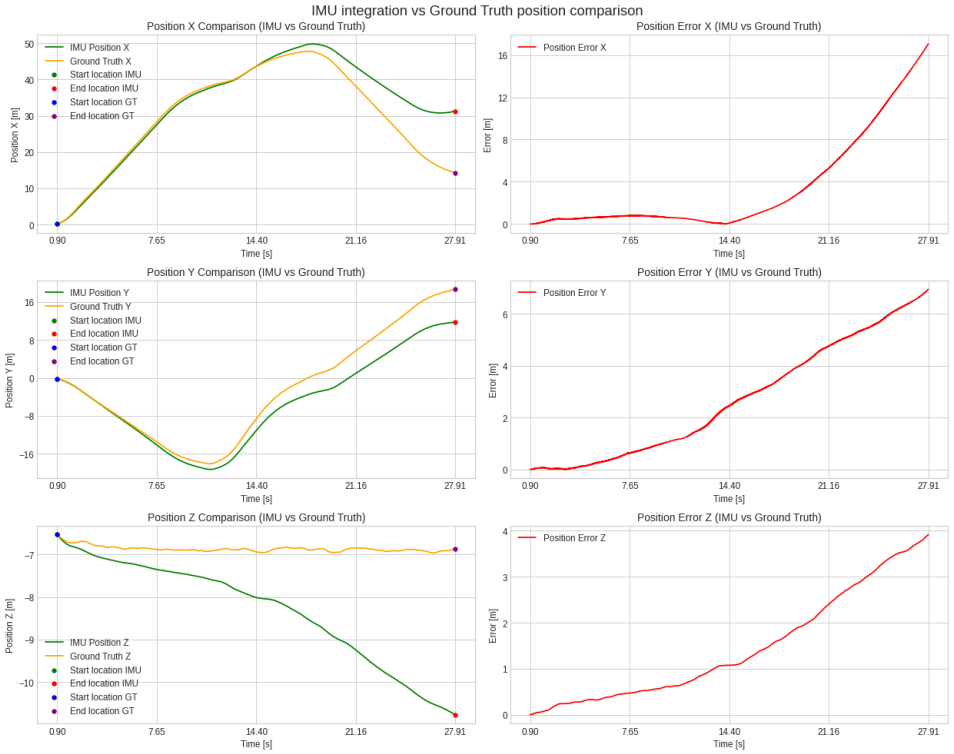


Figure 9.14: Comparison of IMU integration and Ground Truth positions and errors over time

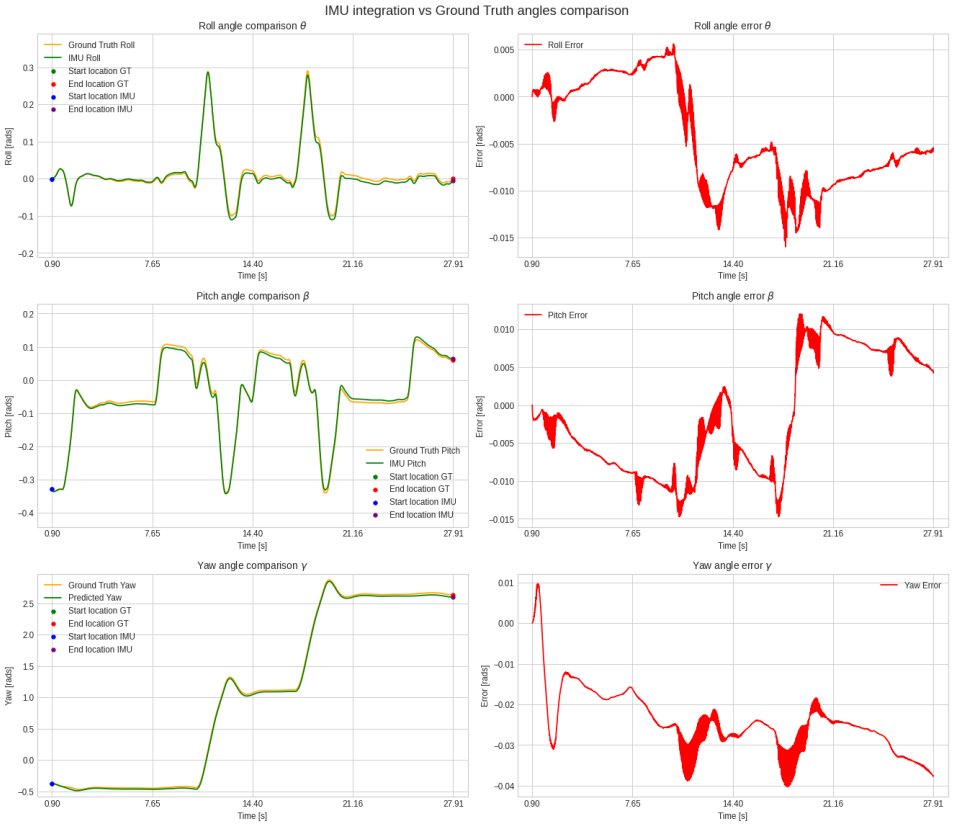


Figure 9.15: Comparison of IMU integration and Ground Truth orientations and errors over time

Another important aspect that emerges from the above graphs is that angular drift is negligible while positional drift is more pronounced. The reason for this is mainly due to the nature of IMU sensors and the integration process of their data. Orientation is estimated by integrating the angular velocity measured by the gyroscope, which tends to accumulate errors at a slower rate than accelerometers. Gyroscopes are generally more accurate and less prone to drift than accelerometers, which limits the effect of angular error over time. Orientation also changes more gradually, and small deviations in angular velocity have less impact on the final result. In contrast, position is calculated by integrating acceleration twice: first to obtain velocity and then to obtain position itself. This double integration process significantly amplifies even small errors in accelerometer readings, such as bias and noise. Errors in acceleration data accumulate over time and cause a progressive and pronounced drift in position estimation.

9.3 Error State Extended Kalman filter results

In this section, the results obtained from the implementation of the ES-EKF are presented. The primary objective of the analysis is to assess the performance of the system under different sensor configurations and understand the individual contributions of each sensor—namely the camera and the IMU—to the final trajectory estimation. Three distinct cases were analyzed, each with different combinations of sensor inputs, to evaluate the advantages and disadvantages of each sensor's contribution in estimating the vehicle's trajectory. In all three cases, careful tuning of the filter parameters was crucial for achieving optimal performance.

The tuning process involved adjusting parameters such as the process noise covariance, measurement noise covariance and the initial state estimates. These parameters play a significant role in the performance of the ES-EKF, as they control how much the filter trusts the IMU measurements compared to the VO or other sensor inputs. Inaccurate tuning can lead to either overconfidence in unreliable sensor data or excessive reliance on potentially noisy measurements. Therefore, obtaining the right balance was fundamental to the success of each configuration. The following sections will describe the specific configurations used in each case, followed by an analysis of the results.

9.3.1 FIRST CASE: 3D position from VO and attitude angles (roll, pitch, yaw) from IMU

In the first case, the sensor fusion is performed using 3D position estimates (x, y, z) from VO combined with attitude angles from IMU.

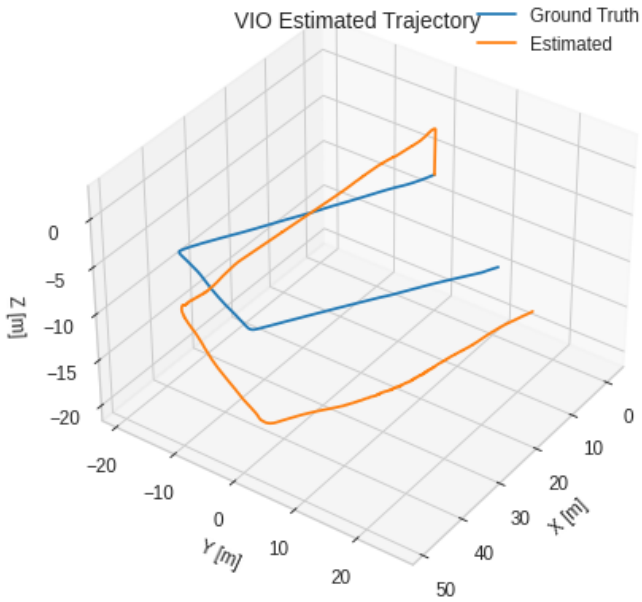


Figure 9.16: 1st CASE: 3D VIO trajectory

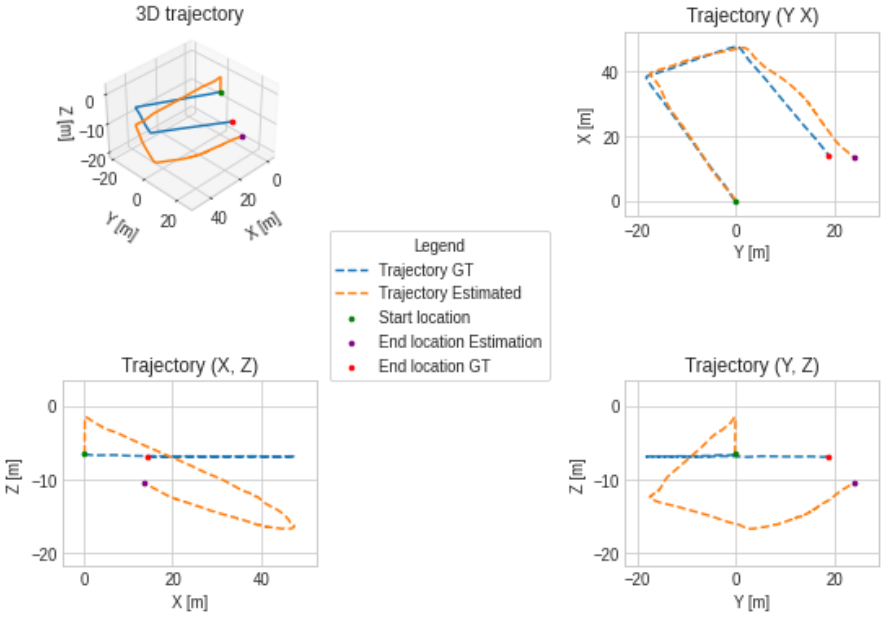


Figure 9.17: 1st CASE: VIO trajectory from different points of view

The main challenge in this configuration, as highlighted in the figures 9.18 and 9.19, is the drift in the z-axis position estimation caused by VO. The camera provides good estimates of x and y positions, but in the absence of accurate depth

information, the z-position is much less reliable. However, the angles estimated using the filter are highly accurate, with very low errors. This is primarily due to the fact that the angles predicted by the IMU in the prediction step were extremely precise, contributing significantly to the overall accuracy of the angle estimation.

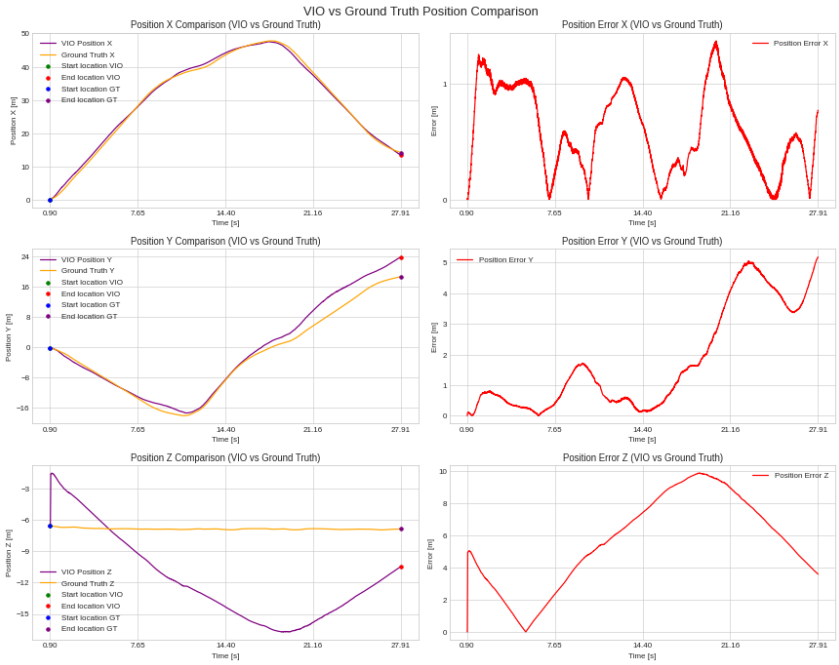


Figure 9.18: 1st CASE: Comparison of VIO and Ground Truth positions and errors over time

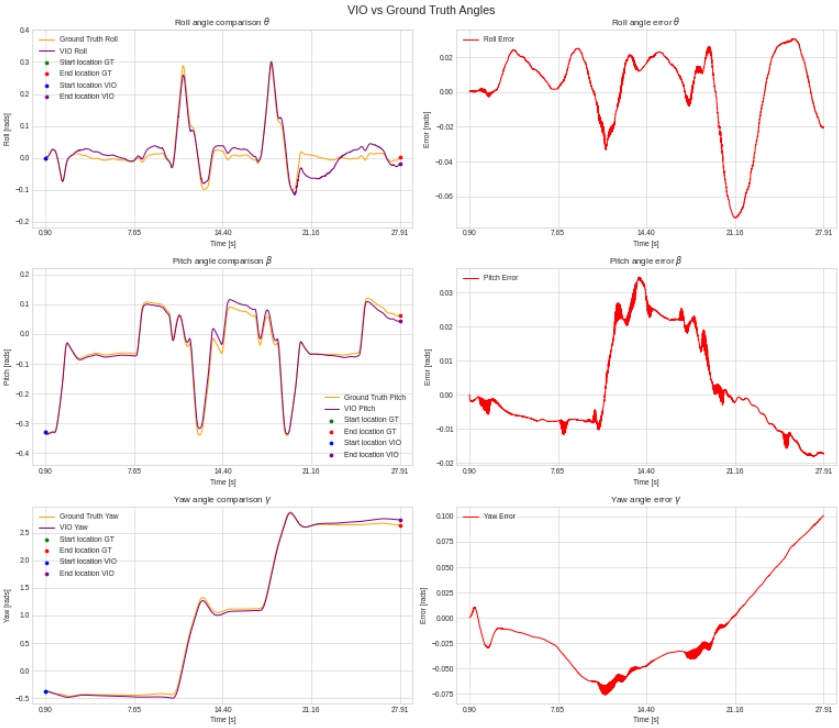


Figure 9.19: 1st CASE: Comparison of VIO and Ground Truth orientations and errors over time

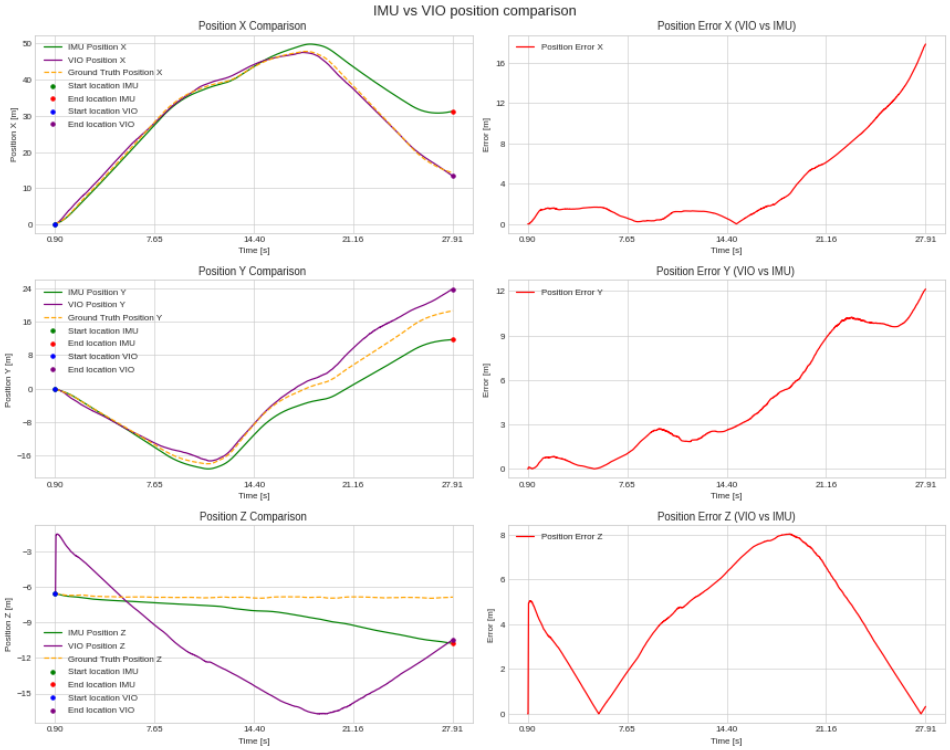


Figure 9.20: 1st CASE: Comparison of positions of VIO, Predicted and Ground Truth, with the error between VIO and Predicted positions shown on the left

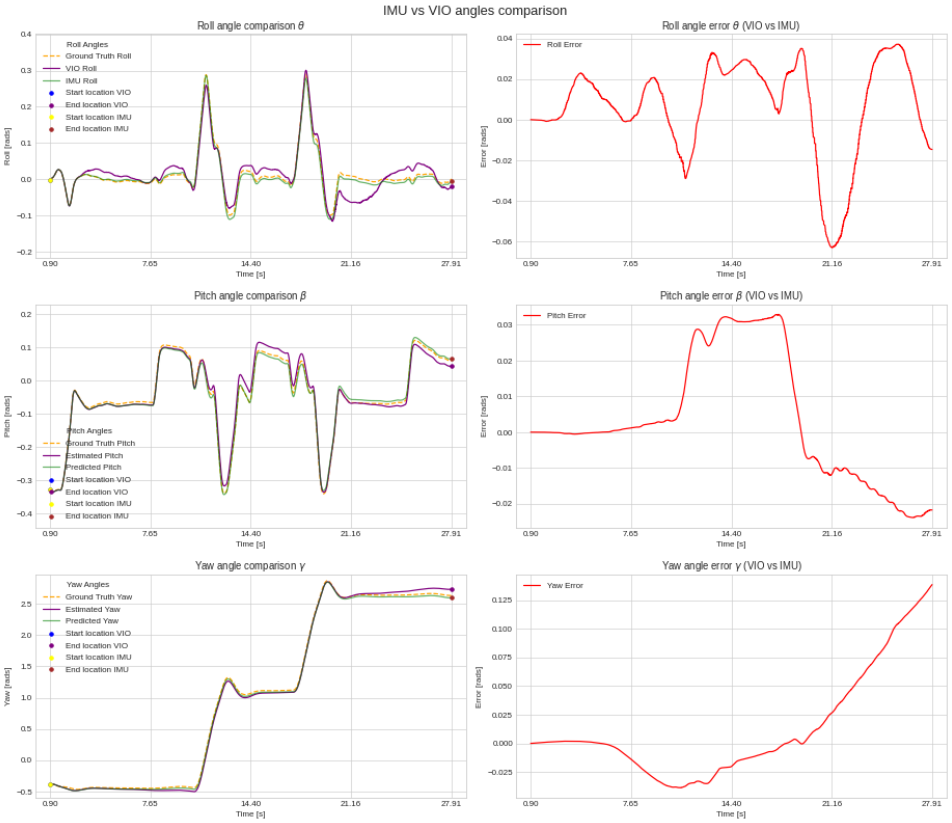


Figure 9.21: 1st CASE: Comparison of angles of VIO, Predicted and Ground Truth, with the error between VIO and Predicted angles shown on the left

In figure 9.20 through the comparison and analysis of positions from the VIO algorithm, the prediction step, and the GT over time, it was found that in terms of position along the x- and y-axis, VIO demonstrates a clear advantage over using the IMU alone, indicating that visual integration improves the accuracy of position estimation. This improvement is likely due to the ability of VIO to correct for drift and errors accumulated by the IMU. Furthermore, the combination of IMU data helps to resolve the scale ambiguity that is typically present in monocular VO, leading to more consistent and reliable position estimates.

Since the fusion process combines only the VO position data with the IMU angle measurements, the alignment between the angle and position estimates may not be perfect, resulting in slightly lower angle accuracy. However, despite this, the angle estimates from VIO are very accurate, with only a slight deterioration compared to the angles obtained from the prediction phase, as shown in Figure 9.21.

9.3.2 SECOND CASE: 2D position from VO, attitude position (z) from IMU and attitude angles (roll, pitch, yaw) from IMU

In the second case, the filter used 2D position from VO, the attitude position (z) from IMU and attitude angles (roll, pitch, yaw) from IMU.

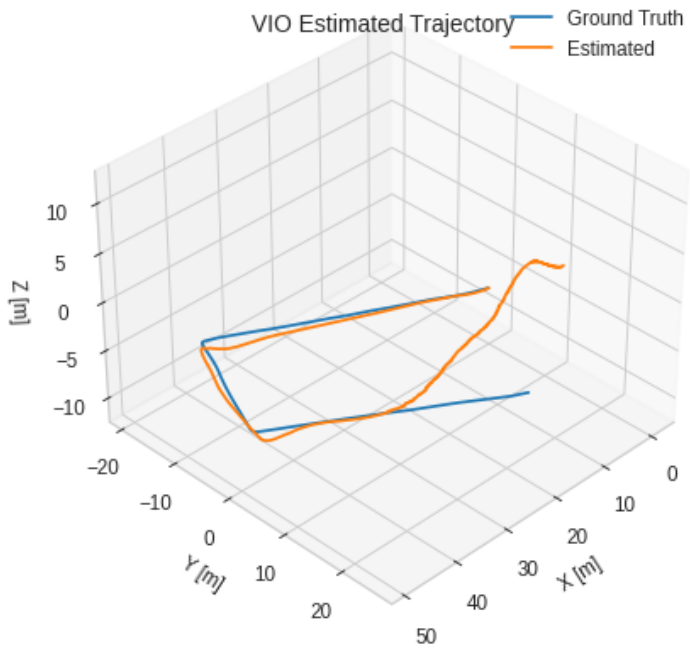


Figure 9.22: 2nd CASE: 3D VIO trajectory

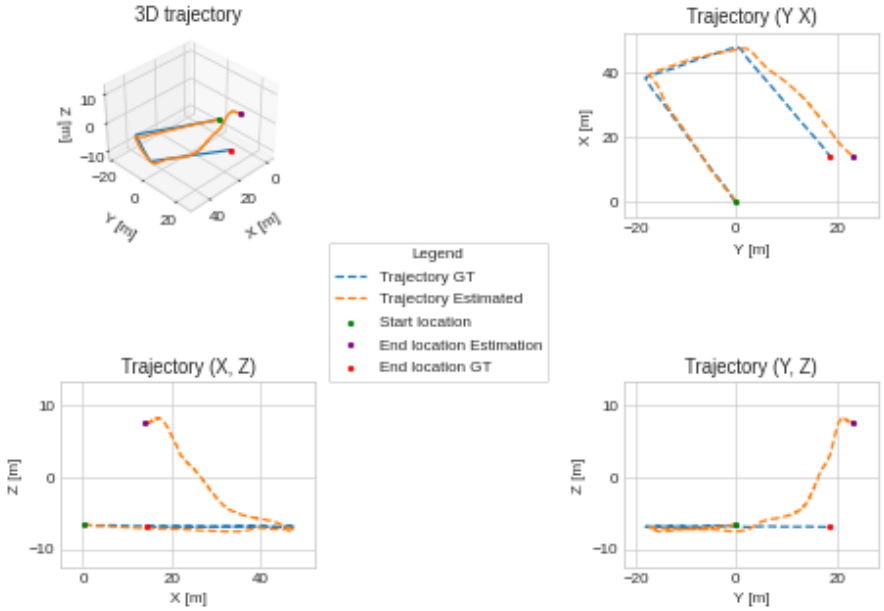


Figure 9.23: 2nd CASE: VIO trajectory from different points of view

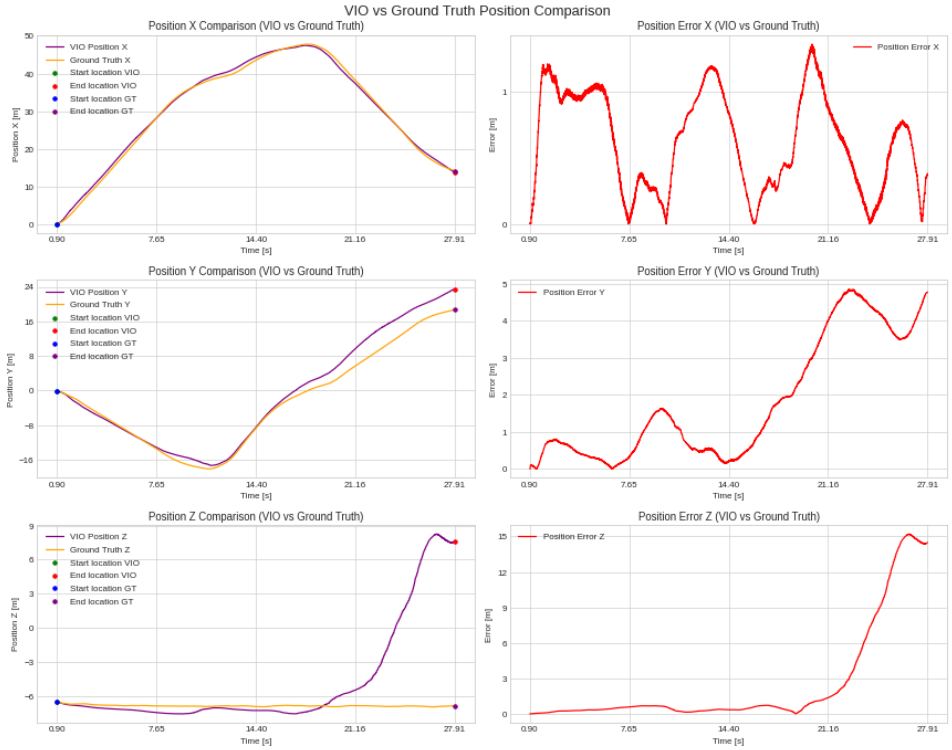


Figure 9.24: 2nd CASE: Comparison of VIO and Ground Truth positions and errors over time

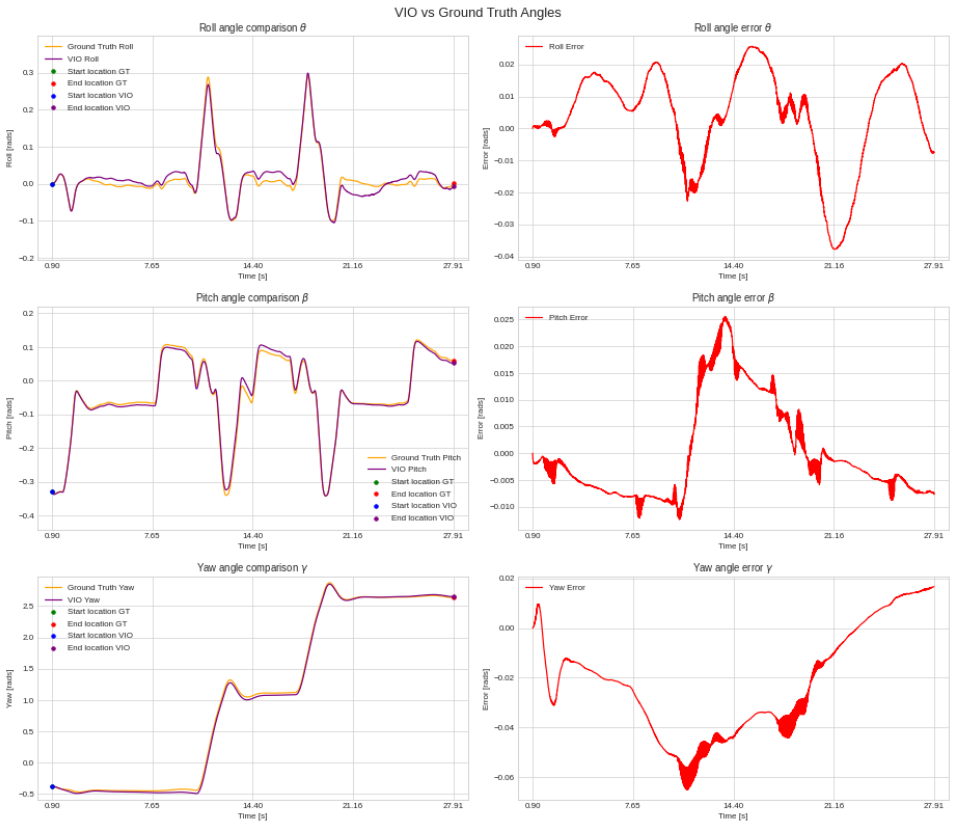


Figure 9.25: 2nd CASE: Comparison of VIO and Ground Truth orientations and errors over time

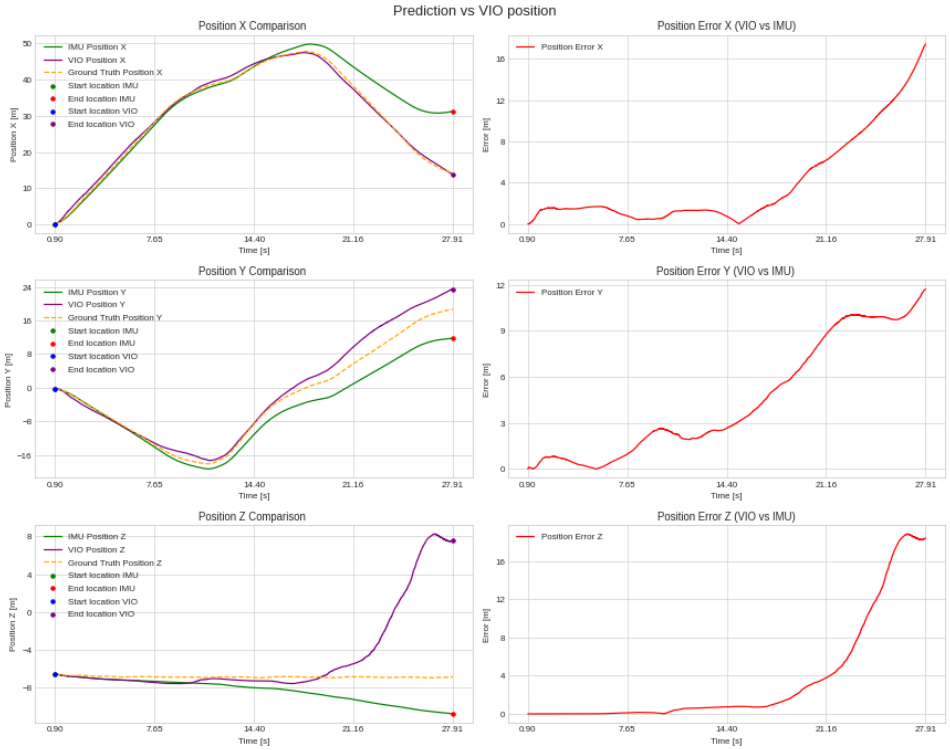


Figure 9.26: 2nd CASE: Comparison of positions of VIO, Predicted and Ground Truth, with the error between VIO and Predicted positions shown on the left

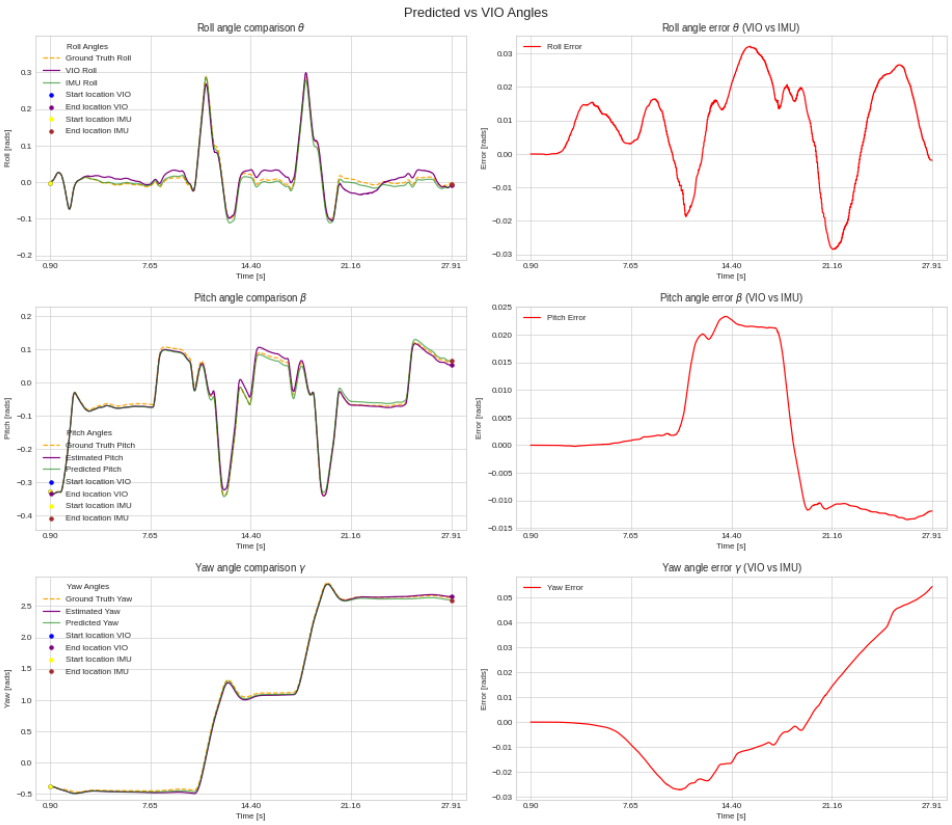


Figure 9.27: 2nd CASE: Comparison of angles of VIO, Predicted and Ground Truth, with the error between VIO and Predicted angles shown on the left

The divergence in the z-axis towards the end of the trajectory, as shown in Figure 9.30, can be attributed to the accumulation of errors from the IMU, as the z-position estimate depends solely on the integration of inertial measurements, which are prone to drift and noise. Additionally, the z-position is less constrained compared to the x and y axes, which benefit from the VO data, making it less observable and more susceptible to errors, as illustrated in Figures 9.24 and 9.20.

9.3.3 THIRD CASE: 2D position from VO, attitude position (z) from barometer and attitude angles (roll, pitch, yaw) from IMU

In the third case, the x and y positions were obtained from VO, the attitude (z) position was derived from a barometer, while attitude angles (roll, pitch, yaw) from IMU. A barometer was integrated to provide a more stable estimate of the vertical position, as the results from the first two cases indicated that neither the VO data nor the IMU-based integration along the z-axis are reliable.

The barometer is simulated using GT data along z-axis, with added noise to replicate the signal typically provided by a real barometer, ensuring a more realistic and accurate representation of its behavior in the system. The signal used to simulate the barometer is shown in the following image.

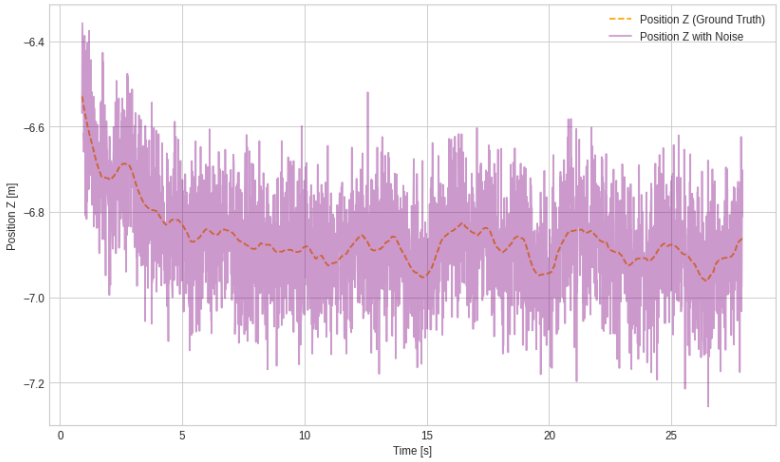


Figure 9.28: Simulated barometer signal based on ground truth data, with added noise to replicate the behavior of a real barometer

Highly accurate and stable z-position estimates are delivered by the barometer, particularly in environments with minimal vertical motion or constant altitude changes. This integration significantly reduced the drift observed in the first two cases, where the z position was solely determined by the VO algorithm and IMU integration, which relied on less stable vertical positioning.

Combining the barometer with the camera for horizontal position estimation and the IMU for orientation led to a more robust overall trajectory estimate, with reduced drift in the vertical dimension.

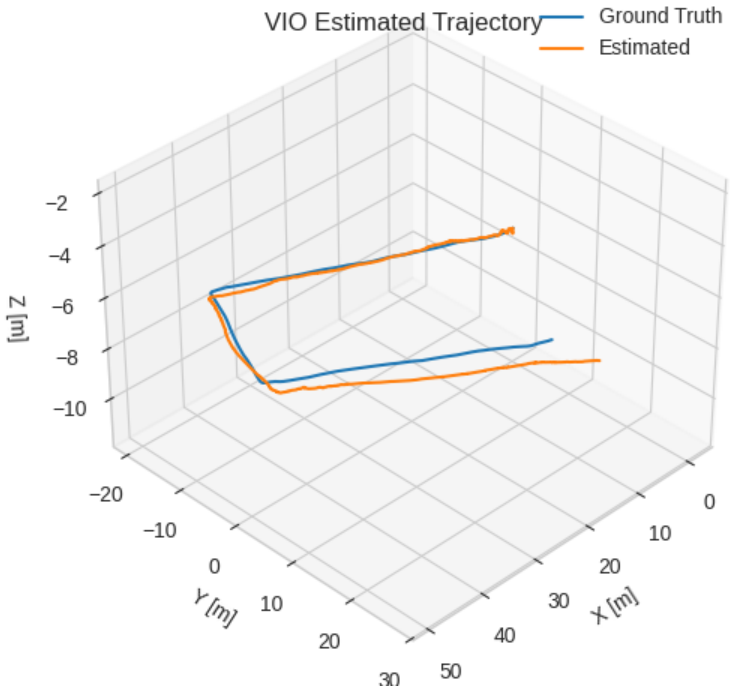


Figure 9.29: 3rd CASE: 3D VIO trajectory

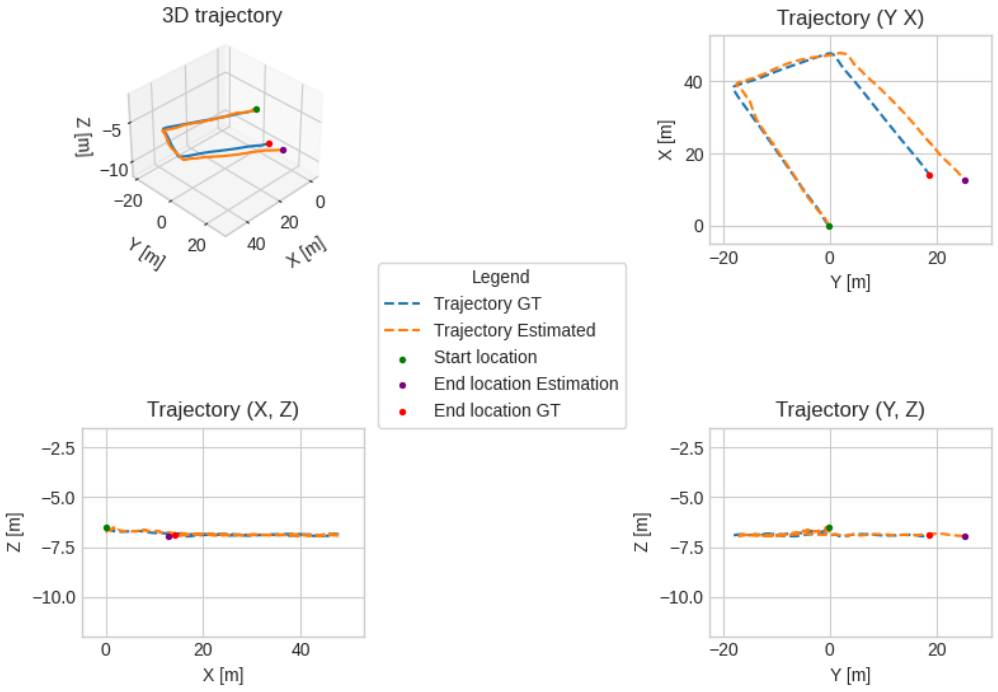


Figure 9.30: 3rd CASE: VIO trajectory from different points of view

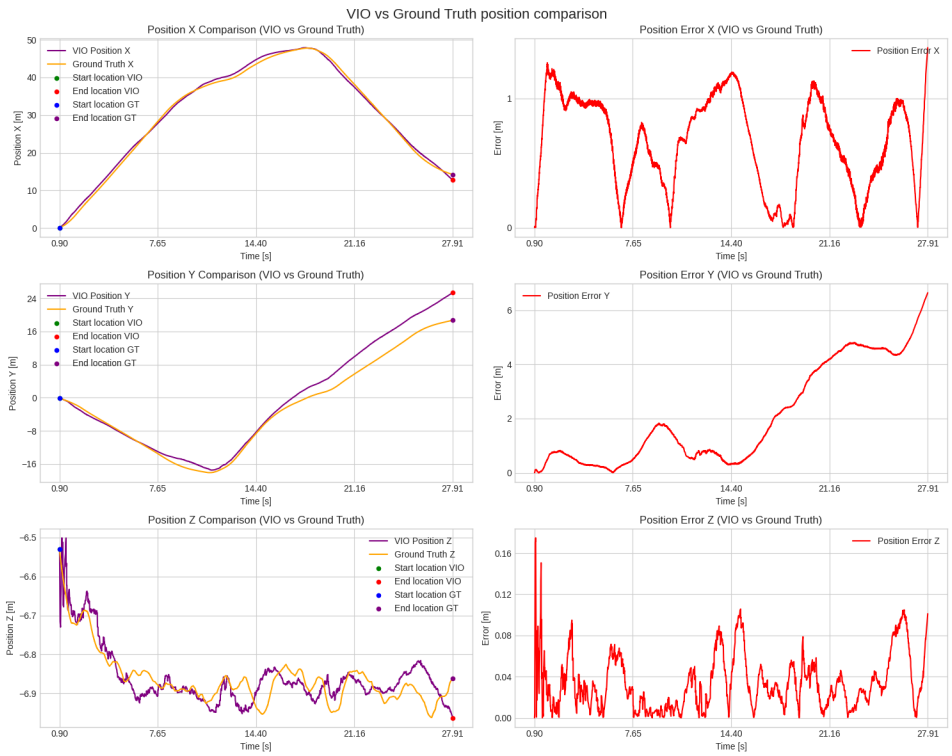


Figure 9.31: 3rd CASE: Comparison of VIO and Ground Truth positions and errors over time

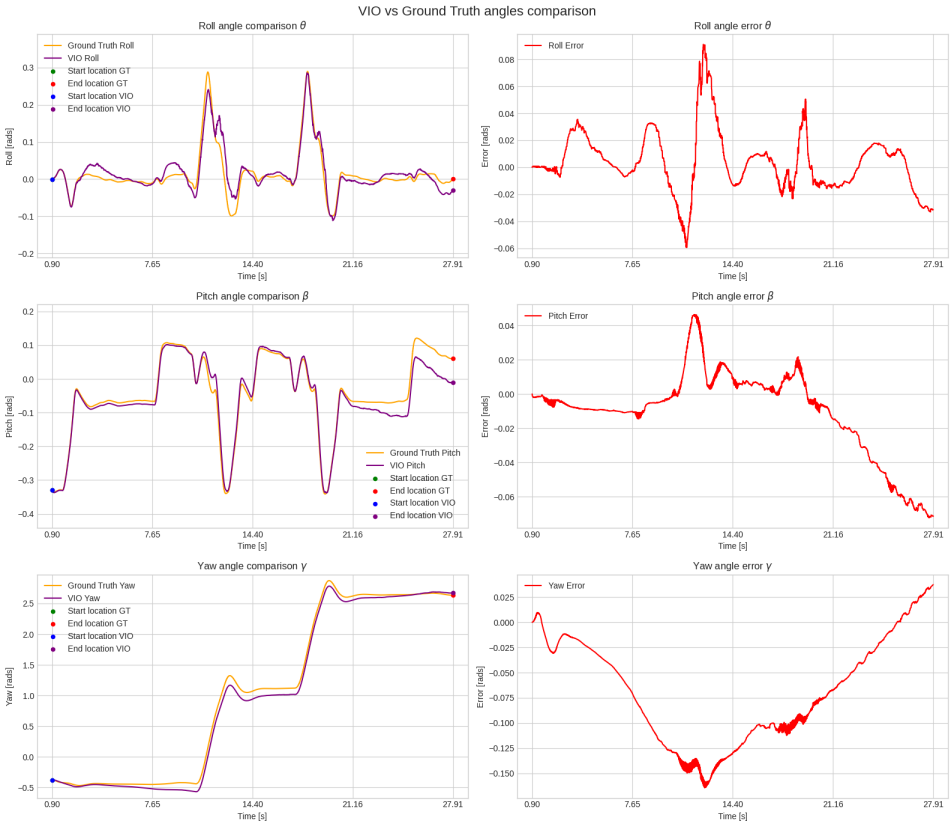


Figure 9.32: 3rd CASE: Comparison of VIO and Ground Truth orientations and errors over time

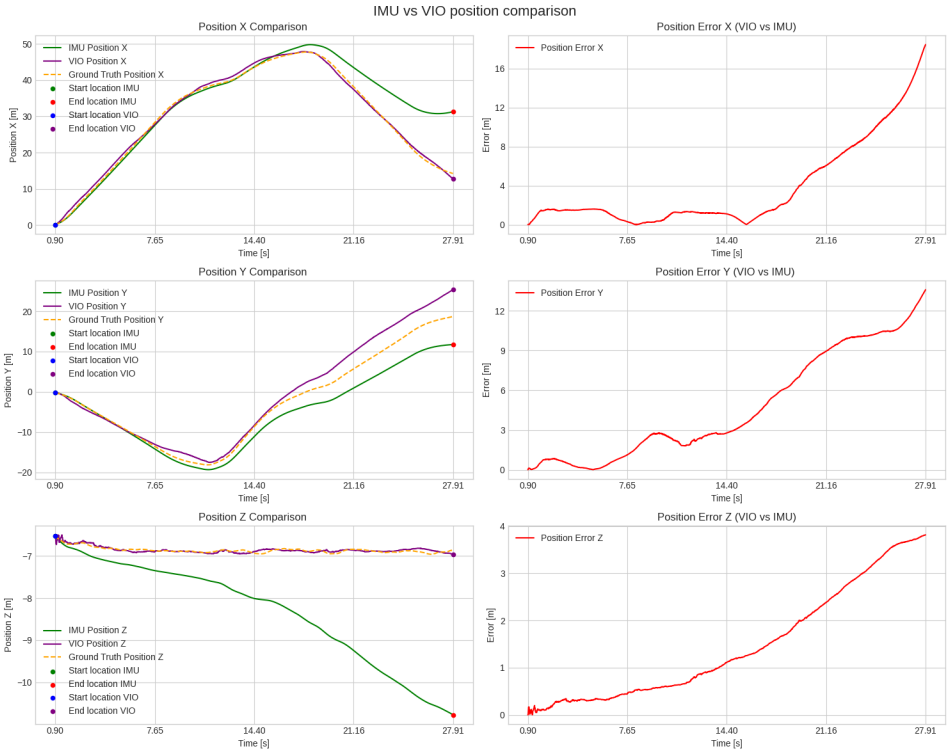


Figure 9.33: 3rd CASE: Comparison of positions of VIO, Predicted and Ground Truth, with the error between VIO and Predicted positions shown on the left

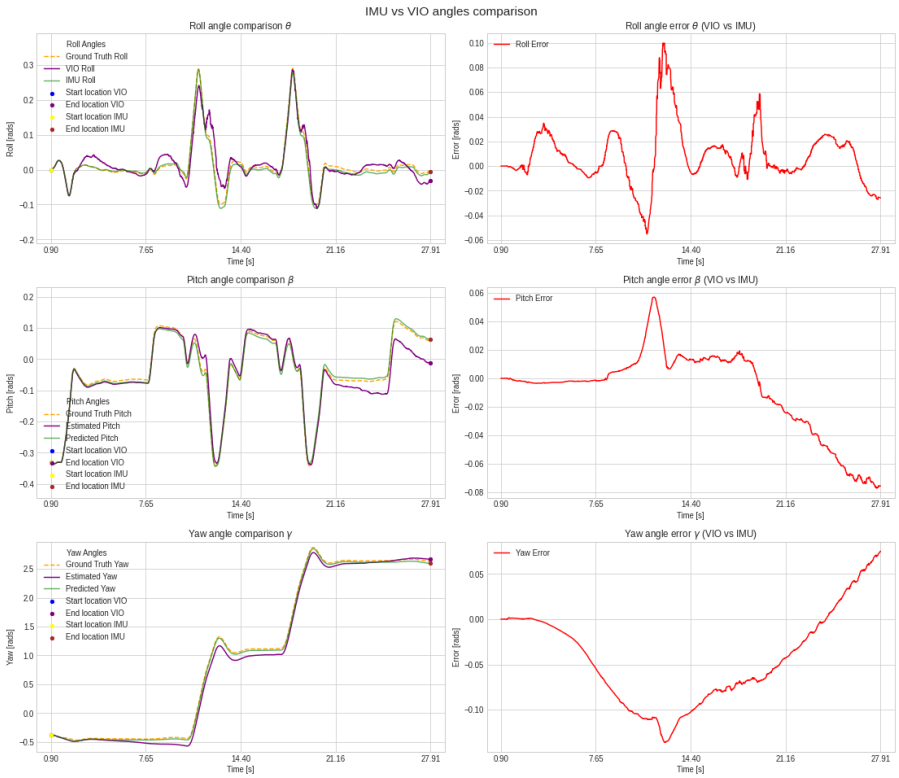


Figure 9.34: 3rd CASE: Comparison of angles of VIO, Predicted and Ground Truth, with the error between VIO and Predicted angles shown on the left

Analyzing the plots above, it is evident that the combination these fused data within the filter yields the best results in terms of trajectory accuracy. By integrating the x and y positions from VO, the z position from the barometer and the IMU integration angles, the system effectively minimizes drift and improves the overall stability of the trajectory estimation, leading to more precise and reliable results.

Across all three components —x, y, and z— the benefits of integrating IMU and VO data are evident. This fusion results in a more accurate trajectory compared to using each sensor individually. Analyzing the errors reveals a deviation of about 1 meter along the x-axis, around 6 meters along the y-axis, and approximately 0.10 meters along the z-axis. Despite these errors, the results remain acceptable in terms of accuracy, especially considering the simplicity of the setup. This demonstrates that even with a basic sensor configuration, combining IMU and VO can lead to better trajectory estimation.

9.3.4 Scale estimation

One of the main objectives of this experiment is to evaluate the ability of the Visual-Inertial ES-EKF framework to estimate scale. The goal is to test the filter's performance in scale estimation under realistic and plausible conditions. Indeed,as highlighted by D.Scaramuzza in [29], the initialization of the visual scale factor correctly, up to about 10% of the true value, is crucial for proper state convergence and for the filter to function correctly. Based on this consideration, the initial value of the scale should be set close to the expected real scale which in this case coincides with the *scale factor*, 6.17, derived from the VO implementation section.

Another important aspect for the proper functioning of the filter is the initialization of the scale's covariance which represents the uncertainty associated with its estimate. Proper initialization of this value is important because it influences how much the filter trusts the initial scale estimate in the early stages of the process. If the covariance is initialized too high, the filter may place too much trust in noisy measurements, leading to incorrect estimates. On the other hand, if it is initialized too low, the filter might ignore important updates, causing slow convergence or even divergence of the scale estimate. Therefore, carefully setting the covariance value ensures that the filter strikes the right balance between trusting the initial estimate and adapting to new data.

Based on the best case scenario among those analyzed so far, namely the third case, several experiments were conducted to investigate how the initialization of the scale and its initial covariance value affected the final estimated trajectory.

SCALE INITIALIZATION: 3

- **Initial covariance:** 0.00003

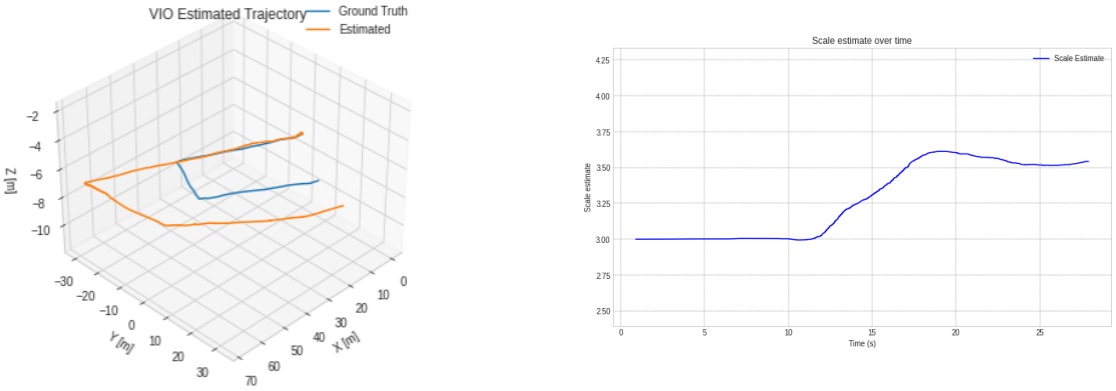


Figure 9.35: Estimated trajectory and scale over time with initial scale equal to 3 and initial covariance to 0.00003

- **Initial covariance:** 0.001

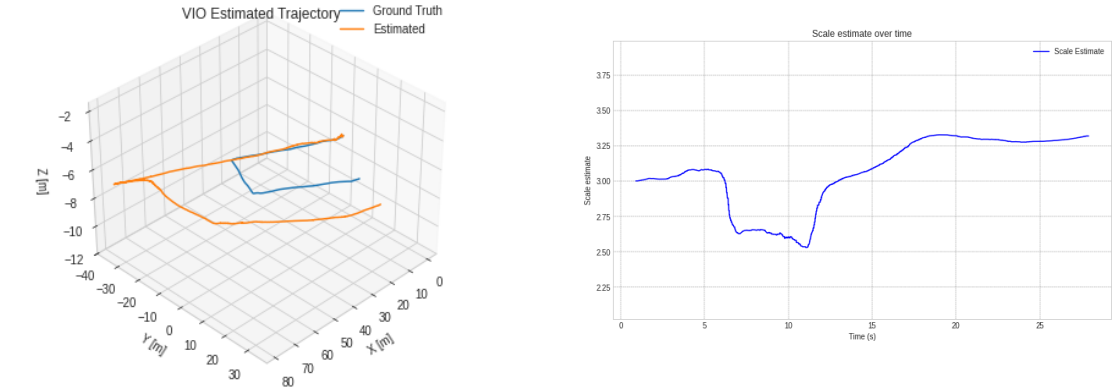


Figure 9.36: Estimated trajectory and scale over time with initial scale equal to 3 and initial covariance to 0.001

- **Initial covariance:** 1

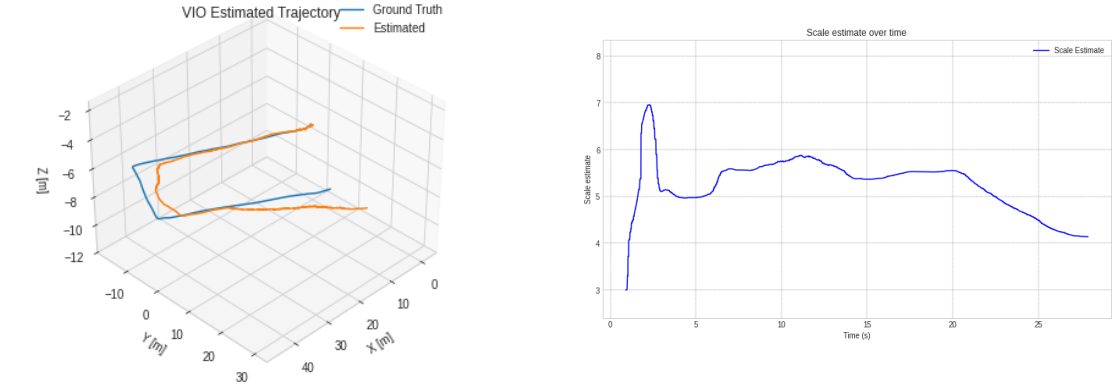


Figure 9.37: Estimated trajectory and scale over time with initial scale equal to 3 and initial covariance to 1

SCALE INITIALIZATION: 5

- **Initial covariance:** 0.00003

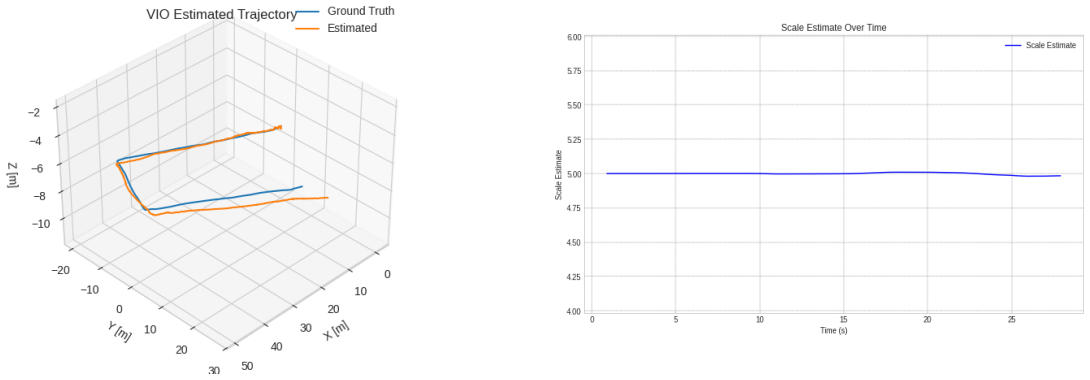


Figure 9.38: Estimated trajectory and scale over time with initial scale equal to 5 and initial covariance to 0.00003

- **Initial covariance:** 0.001

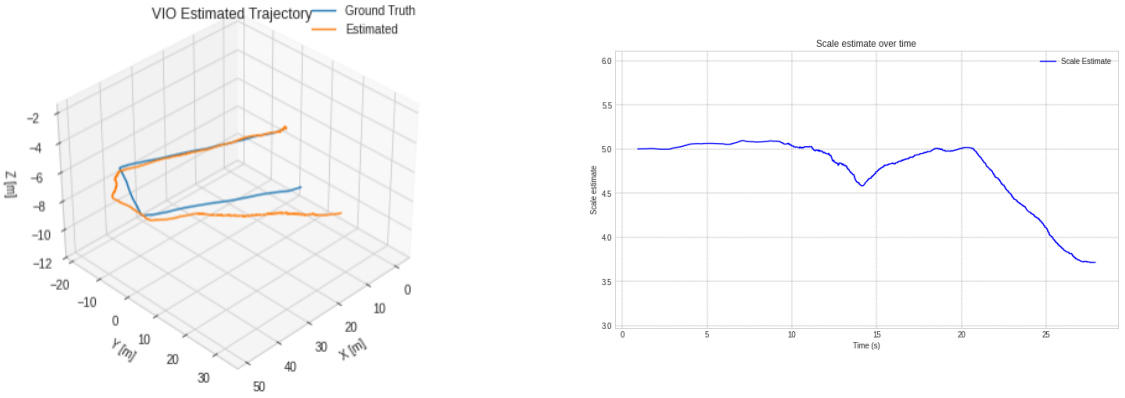


Figure 9.39: Estimated trajectory and scale over time with initial scale equal to 5 and initial covariance to 0.001

- **Initial covariance:** 1

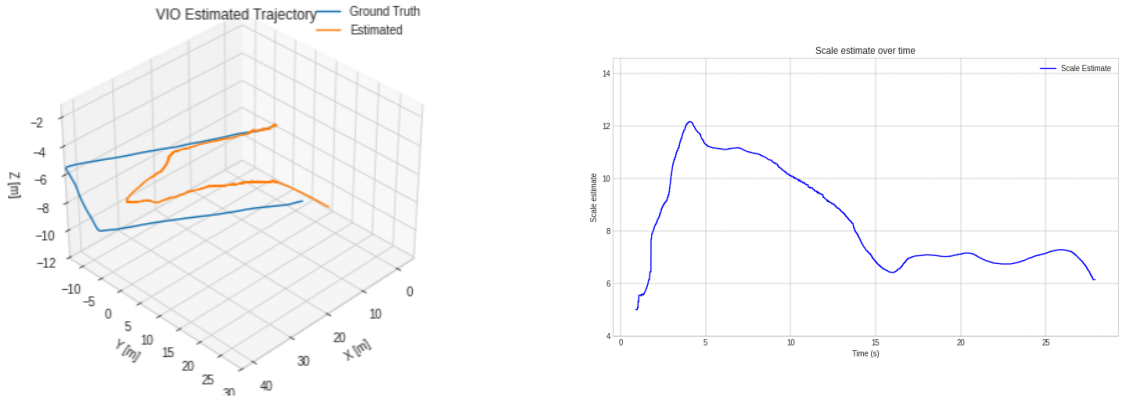


Figure 9.40: Estimated trajectory and scale over time with initial scale equal to 5 and initial covariance to 1

SCALE INITIALIZATION: 6.17

- **Initial covariance:** 0.00003

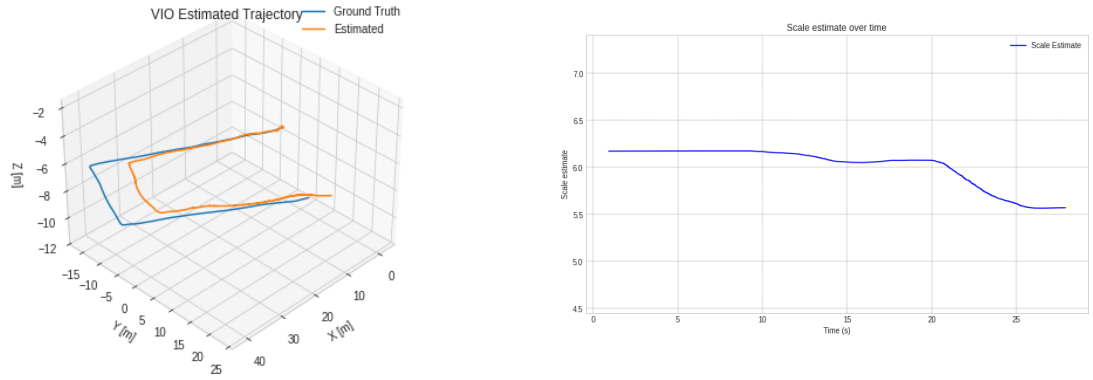


Figure 9.41: Estimated trajectory and scale over time with initial scale equal to 6.17 and initial covariance to 0.00003

- **Initial covariance:** 0.001

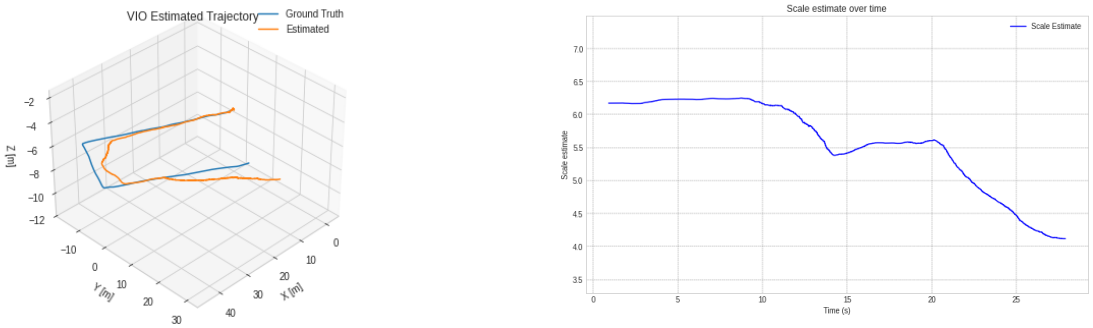


Figure 9.42: Estimated trajectory and scale over time with initial scale equal to 6.17 and initial covariance to 0.001

- **Initial covariance:** 1

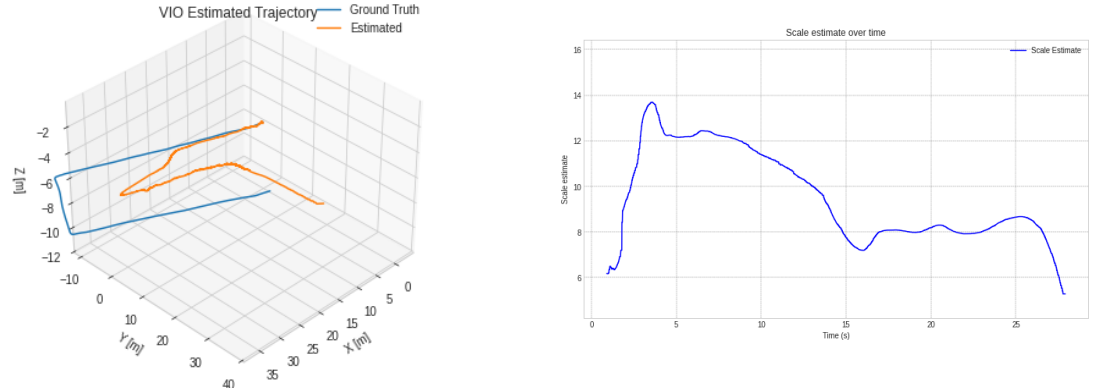


Figure 9.43: Estimated trajectory and scale over time with initial scale equal to 6.17 and initial covariance to 1

From the graphs above, we can observe that when the scale is initialized at 3, a value much smaller than the scale ratio, the filter, starting with a very low initial covariance, struggles to recover the correct scale and settles on a much lower value, around 3.5. On the other hand, initializing with a higher initial covariance (set to 1) causes the system to place less trust in the initial scale value, allowing the filter to incorporate more measurements and recover the scale more effectively. This results in a better alignment between the estimated trajectory and the GT trajectory.

Now, consider the case where the scale is initialized with the scale ratio of 6.17 and a low covariance, indicating high confidence in the initial value. In this scenario, the filter closely tracks the GT trajectory, but the estimated scale tends to be underscaled. When the covariance is increased, the estimated trajectory becomes unstable, and the scale value fluctuates significantly between 14 and 5. This instability occurs because the filter is overcompensating in its attempt to adjust the scale, unable to settle on an optimal value. Despite initializing the filter with the correct scale, the results are suboptimal, even when experimenting with different initial variance settings.

Through a process of trial and error, an optimal scale initialization value of 5 was determined. This value is close to the calculated scale factor but slightly smaller. With this initialization, the filter achieves optimal results, particularly when the initial covariance is set to 0.00003. In this case, the scale is successfully recovered and the VIO trajectory closely matches the GT trajectory, aside from inherent errors in the localization algorithm. Furthermore, the scale stabilizes and converges toward the initialized value. In contrast, increasing the initial covariance further significantly degrades performance. In conclusion, a scale value of 5 provides a good trade-off between recovering the scale and correcting the error in the trajectory estimate obtained through VIO.

9.4 Error metrics

In this section various error metrics are calculated to evaluate the final performance of the VIO system using the ES-EKF filter. The goal was to understand whether and to what extent the combination of VO data with IMU data led to improvements in both position and angle estimates compared to using IMU data alone.

The error metrics used are as follows:

1. **Root Mean Square Error (RMSE)**: This matrix quantifies the average magnitude of the error between the estimated and GT values. It gives an indication of how closely the estimated position or orientation matches the actual values, with smaller values representing better accuracy. The RMSE is particularly useful for assessing the overall precision of the system [50].

2. **Mean Absolute Error (MAE)**: MAE calculates the average of the absolute differences between the estimated and true values. Unlike RMSE, which penalizes larger errors more heavily, MAE provides a more straightforward and intuitive measure of average error magnitude, useful for identifying consistent biases or deviations [50].

3. **Error Percentage**: This matrix expresses the ratio of the final position error to the total travel distance. It provides a normalized measure of the overall accuracy relative to the distance traveled, allowing for better comparison of performance across different scales and scenarios.

These metrics were computed for both the VIO system with ES-EKF and for the IMU-only system. By comparing the results, we can assess how the fusion of IMU data and VO improves the accuracy and reliability of the system, both in terms of positional and angular estimates.

Position metrics

Metric	ESKF	IMU
RMSE (m)	3.48	7.14
MAE (m)	10.02	18.82
Error Percentage (%)	11.13	11.17

Table 9.1: Error Metrics for ESKF and IMU Systems

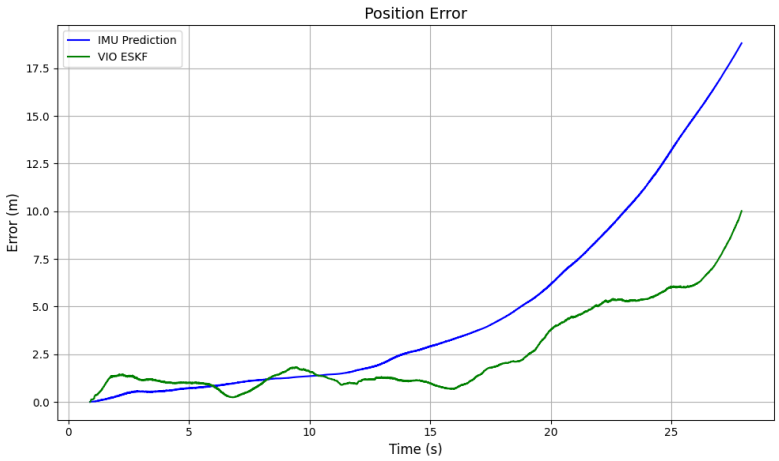


Figure 9.44: Position error comparison between IMU integration and VIO ESKF over time.

The results in Table 9.1 show that the ES-EKF system outperforms the IMU in position accuracy. The ES-EKF achieves a lower RMSE of 3.48 meters, compared to 7.14 meters for the IMU and a smaller MAE of 10.02 meters versus 18.82 meters, suggesting that the ES-EKF model has smaller deviations from the true position. Although the error percentages for both systems are similar, 11.13% for ES-EKF and 11.17% for IMU, thus the ES-EKF still demonstrates a slight edge in terms of absolute error. These findings highlight that incorporating VO information with the ES-EKF improves position estimation accuracy compared to using the IMU alone, as further illustrated in Figure 9.45.

Angles metrics

Metric ROLL	ESKF	IMU
RMSE (rad)	0.0299	0.0070
MAE (rad)	0.0968	0.0160
Error Percentage (%)	2.08	0.60

Table 9.2: Roll error metrics comparison between ESKF and IMU integration.

Metric PITCH	ESKF	IMU
RMSE (rad)	0.0655	0.0078
MAE (rad)	0.1792	0.0147
Error Percentage (%)	4.14	0.72

Table 9.3: Pitch error metrics comparison between ESKF and IMU integration.

Metric YAW	ESKF	IMU
RMSE (rad)	0.0745	0.0256
MAE (rad)	0.1501	0.0403
Error Percentage (%)	6.15	2.46

Table 9.4: Yaw error metrics comparison between ESKF and IMU integration

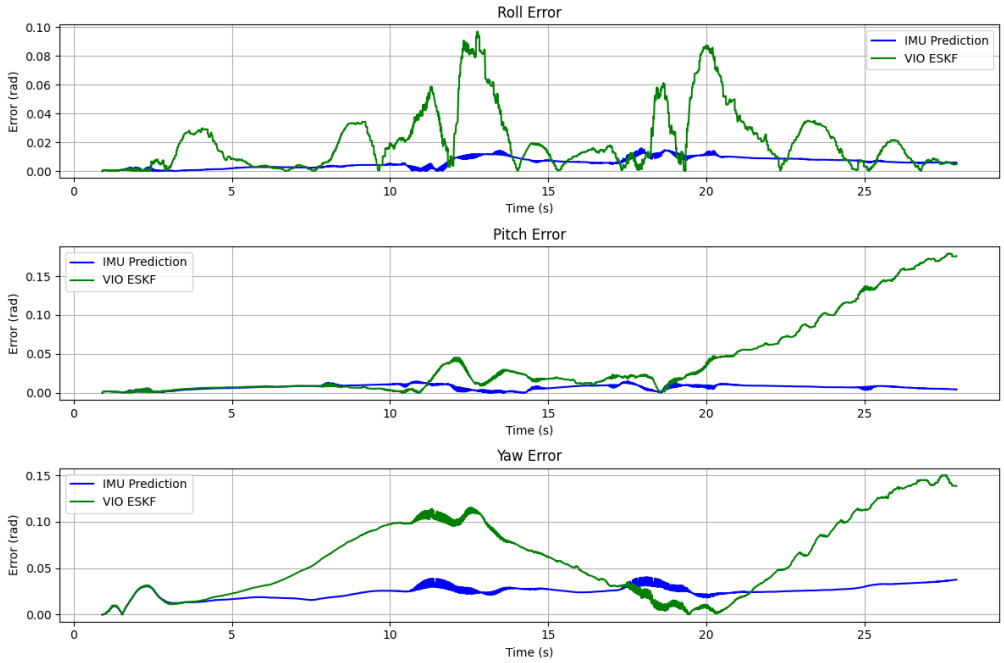


Figure 9.45: Angles error comparison between IMU integration and VIO ESKF over time

The results presented in the tables for roll, pitch and yaw errors show that the IMU alone performs slightly better than the ES-EKF integration for angle estimation. For all three angles, the RMSE, MAE and error percentages for the IMU are lower than those for the ES-EKF. In particular, the RMSE for roll, pitch and yaw are all smaller with the IMU, suggesting that the IMU alone provides more accurate angle estimates in this case. While the ES-EKF integration improves position estimation, it appears to introduce small errors when estimating orientation relative to the IMU. This behavior is further emphasized by the visual comparison in Figure 9.45.

Chapter 10

VIO enhanced with a depth camera: preliminary implementation

Until now, an ES-EKF approach has been used to integrate information from monocular VO and IMU, with the goal of achieving a more accurate trajectory compared to the individual sensors. This approach addressed one of the main challenges of monocular VO, namely the scale ambiguity. The results obtained from this implementation demonstrated that the approach works effectively, yielding good results in terms of accuracy.

At this point, a slightly different approach was chosen to see if the limitations of the initial method could be overcome and if better results can be obtained. A depth camera, specifically the X500 Depth Camera in Gazebo, shown in the image 10.1, was introduced, which features two grayscale sensors for stereo depth and a single central color sensor.

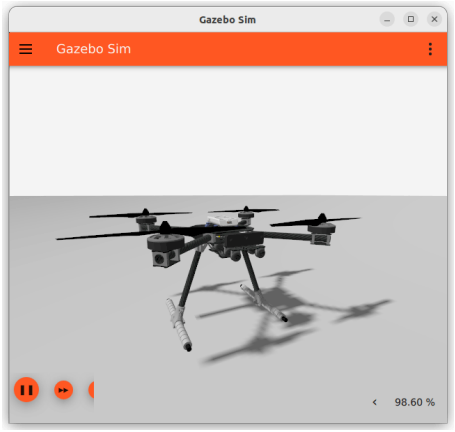


Figure 10.1: X500 Depth Camera UAV [27]

With this new sensor, the scale will no longer be estimated by the filter but instead provided as input to the VO algorithm, which will output a scaled trajectory directly. To implement this new configuration, the VO algorithm was modified: instead of using 2D-2D motion estimation, 3D-2D motion estimation was used and the SIFT method was replaced with ORB. The rest of the system, including the IMU

data and filter structure, remained unchanged, except for the removal of the scale from the state vector, as the scale is now provided directly by the depth camera.

10.1 Depth details

A depth camera with a maximum range of 17.1 meters and a minimum range of 0.2 meters is used. These values represent the maximum and minimum depth distances that the camera is capable of perceiving. The depth data captured by the camera is then published to the ROS2 topic `/depth_camera`, allowing for processing of depth information within the system. To facilitate communication between ROS2 and PX4, an additional bridge is required to handle depth camera-related topic. The following command is used to run the bridge and stream the depth camera data:

```
ros2 run ros_gz_image image_bridge /depth_camera
```

Below is the image captured by the depth camera along with the corresponding depth map.

Figure 10.2: Image captured by the depth camera

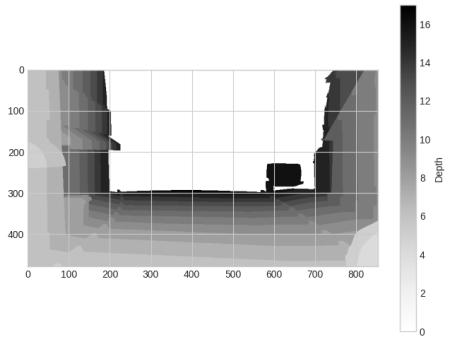


Figure 10.3: Depth map associated with the captured image

10.2 VO implementation and results

In the motion estimation process, a 3D-2D approach is employed to integrate depth information and estimate camera motion between two subsequent image frames. The algorithm works by matching keypoints between two consecutive images, which are extracted using the ORB feature detector and descriptor. This choice was made because, experimentally, it was observed that ORB provided a more accurate final trajectory compared to SIFT.

The algorithm begins by detecting and matching keypoints between two images. For each matched keypoint, the algorithm extracts the 2D pixel coordinates from both images and, if a depth map is provided for the first image, it retrieves the depth information corresponding to each keypoint. The 2D points from the first image are transformed into normalized camera coordinates by applying the inverse of the camera calibration matrix K. This operation maps the 2D pixel coordinates

to 3D rays in the camera frame. The depth values are then used to scale these 3D points, yielding the corresponding 3D positions in the camera's coordinate system.

Once the 3D-2D correspondences are established, the algorithm proceeds to estimate the camera's motion using PnP, specifically employing the RANSAC method to robustly solve for the camera's rotation and translation vectors. The RANSAC-based approach is utilized to filter out outliers and improve the accuracy of the motion estimation, ensuring that only inliers are used in the final estimation. The camera's rotation and translation are obtained by solving for the pose that minimizes the reprojection error, which measures the discrepancy between the observed 2D keypoints in the second image and the corresponding projected 3D points in the camera frame. The steps outlined above are summarized in the table below:

Step	Description
1.	Detect and match keypoints using the ORB feature detector.
2.	Retrieve depth values for each matched keypoint of the first image.
3.	Convert 2D points to normalized camera coordinates using the inverse of K
4.	Scale 3D rays by depth to obtain 3D points in the camera frame.
5.	Estimate camera motion using PnP and RANSAC.
6.	Use RANSAC to filter out outliers and refine the estimate.
7.	Minimize reprojection error to finalize the camera pose.

Table 10.1: Steps of the 3D-2D motion estimation algorithm with provided depth values

The results obtained by the implementation of this algorithm are illustrated in the following Figures:

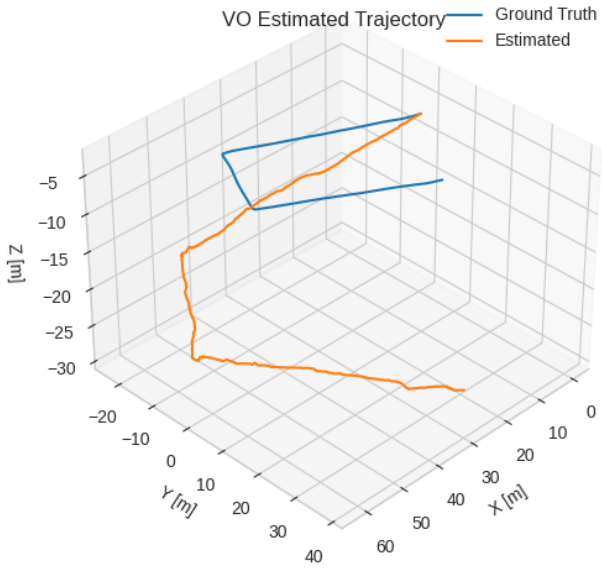


Figure 10.4: Depth camera: VO 3D-2D final trajectory

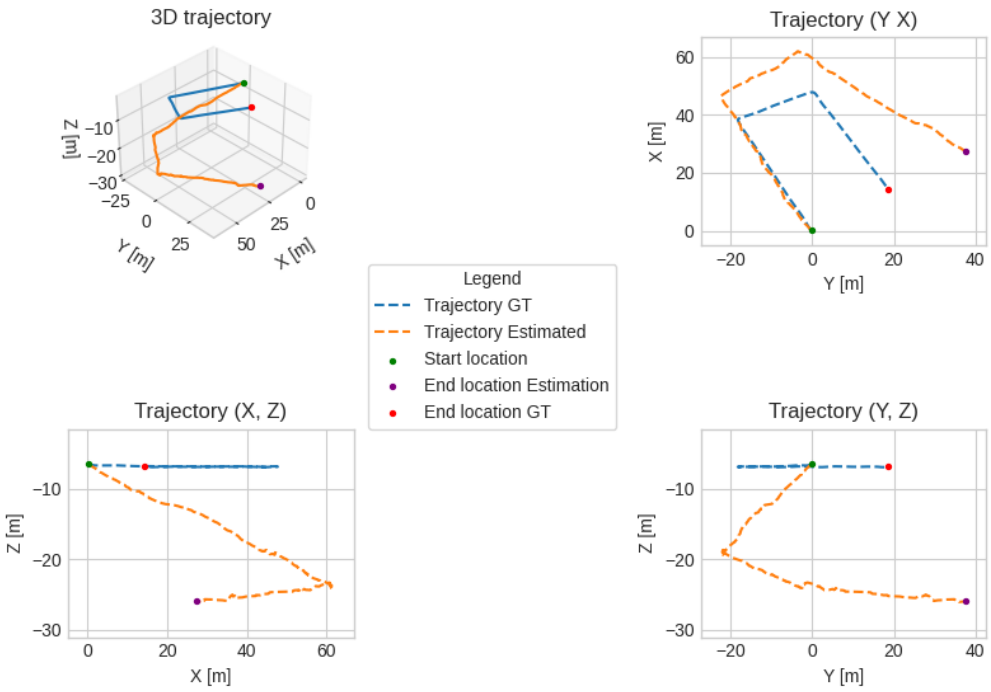


Figure 10.5: Depth camera: VO 3D-2D final trajectory from different points of view

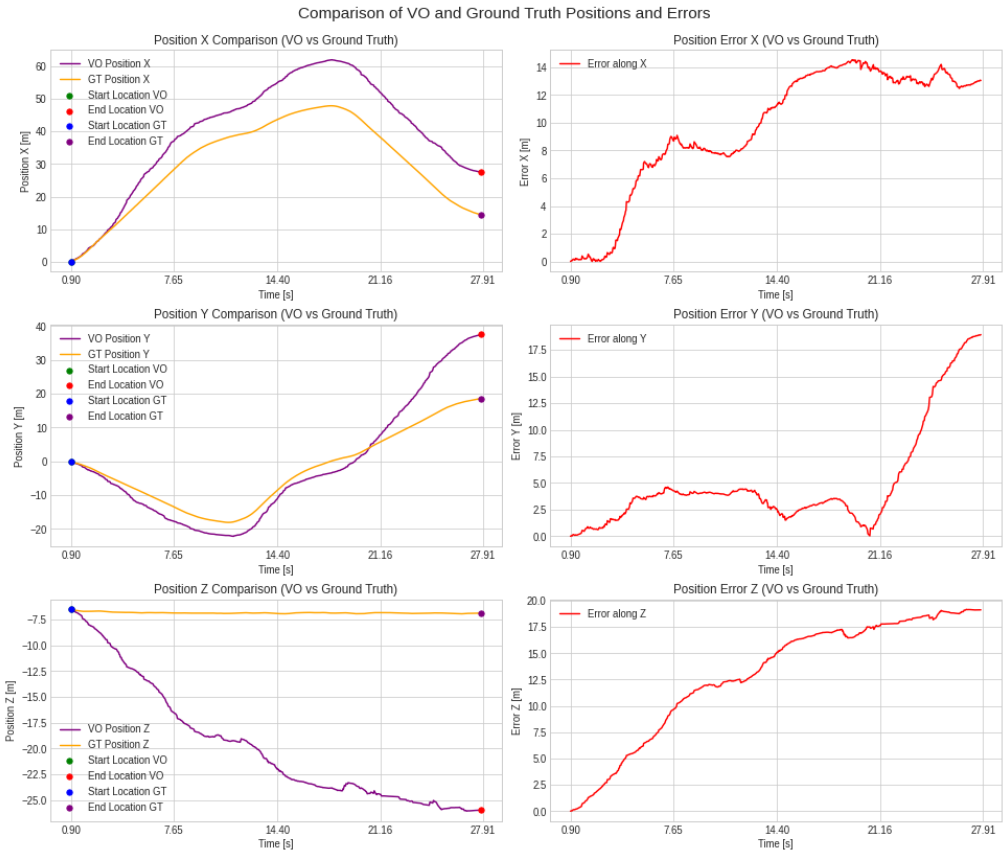


Figure 10.6: Depth camera: Comparison of VO and Ground Truth positions and errors over time

Figure 10.7: Depth camera: Comparison of VO and Ground Truth angles and errors over time

This new approach appeared promising, as it incorporated additional depth information, which was expected to improve the accuracy of the motion estimation. However, in the end it did not meet expectations. Specifically, the scale of the estimated movement appears not to have been fully recovered. Significant errors are observed along the X-axis (approximately 14 meters), the Y-axis (around 17 meters), and, similarly, along the Z-axis. This indicates that the depth information, which plays an important role in the 3D-2D correspondence and motion estimation, is not accurately captured or utilized.

Several factors may explain the discrepancies, including potential inaccuracies in the depth data, such as incorrect calibration of the depth camera or poor quality depth maps. These issues could arise from improper camera settings or errors in the simulation's .sdf file parameters. Additionally, suboptimal settings in the PnP-RANSAC solver, especially for outlier rejection, may have led to an imperfect estimation of the camera's motion.

10.3 Error State Extended Kalman filter results

As mentioned at the beginning of the chapter, the structure of the filter stays the same, only the scale has been removed from the state vector. The results obtained by the filter are summarized in the following Figures:

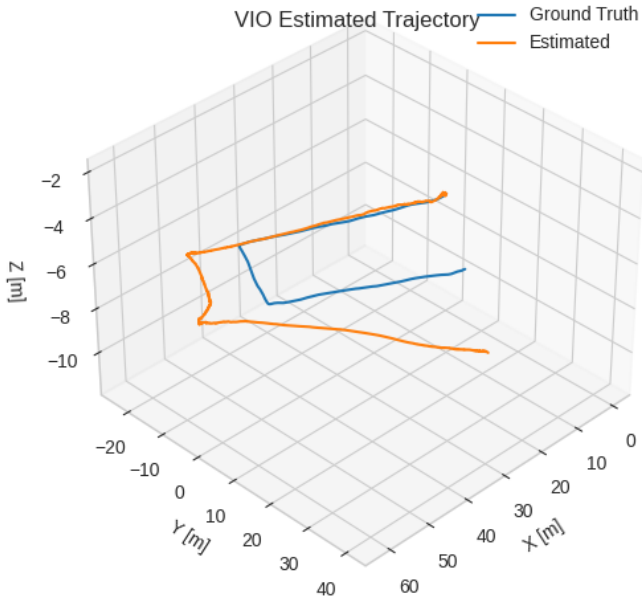


Figure 10.8: Depth camera: 3D VIO trajectory

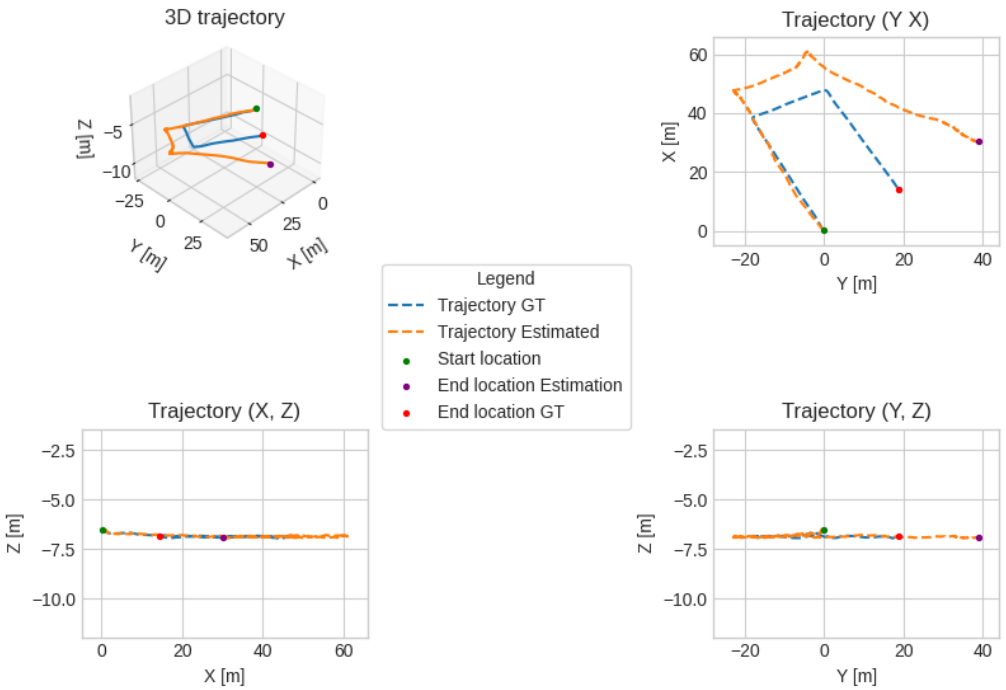


Figure 10.9: Depth camera: VIO trajectory from different points of view

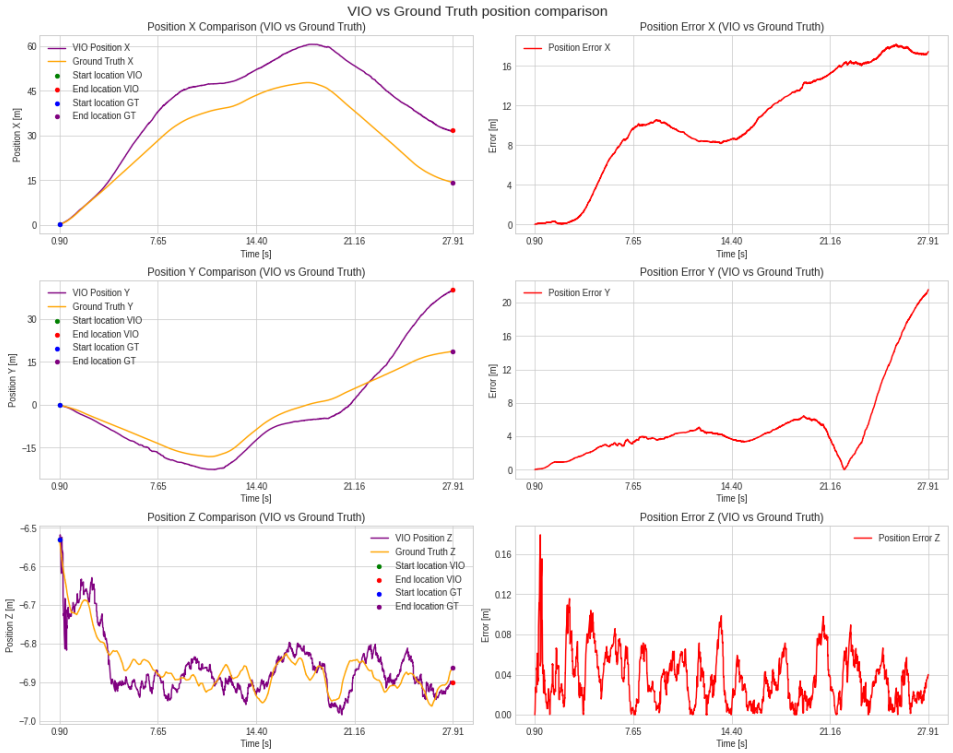


Figure 10.10: Depth camera: Comparison of VIO and Ground Truth positions and errors over time

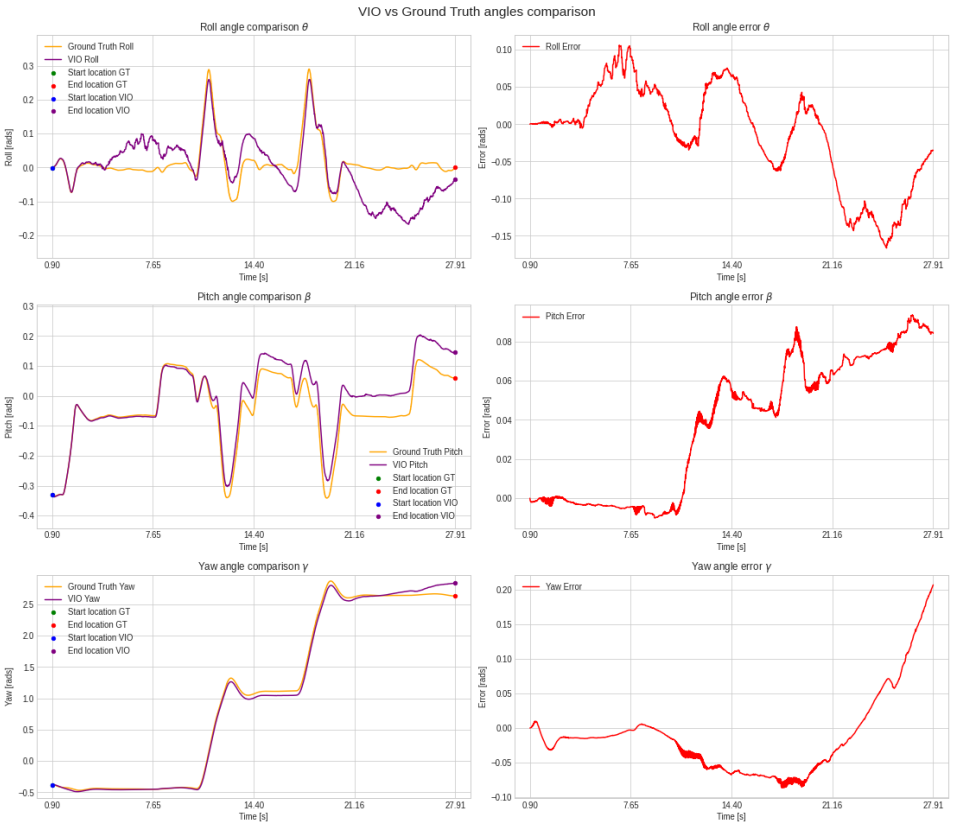


Figure 10.11: Depth camera: Comparison of VIO and Ground Truth angles and errors over time

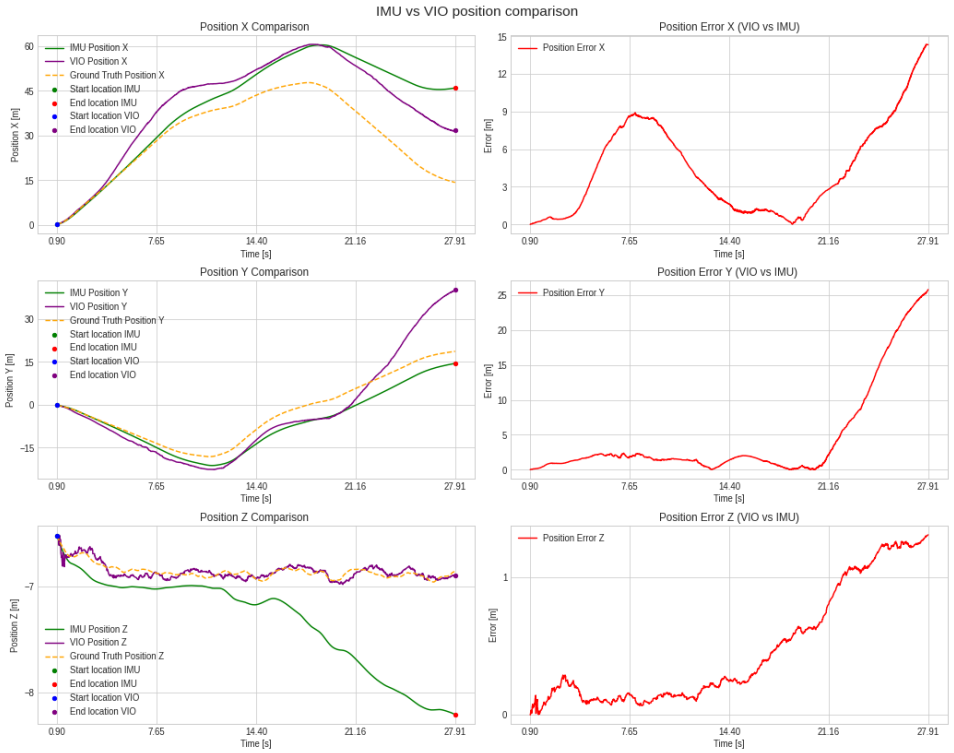


Figure 10.12: Depth camera: Comparison of positions of VIO, Predicted and Ground Truth, with the error between VIO and Predicted positions shown on the left

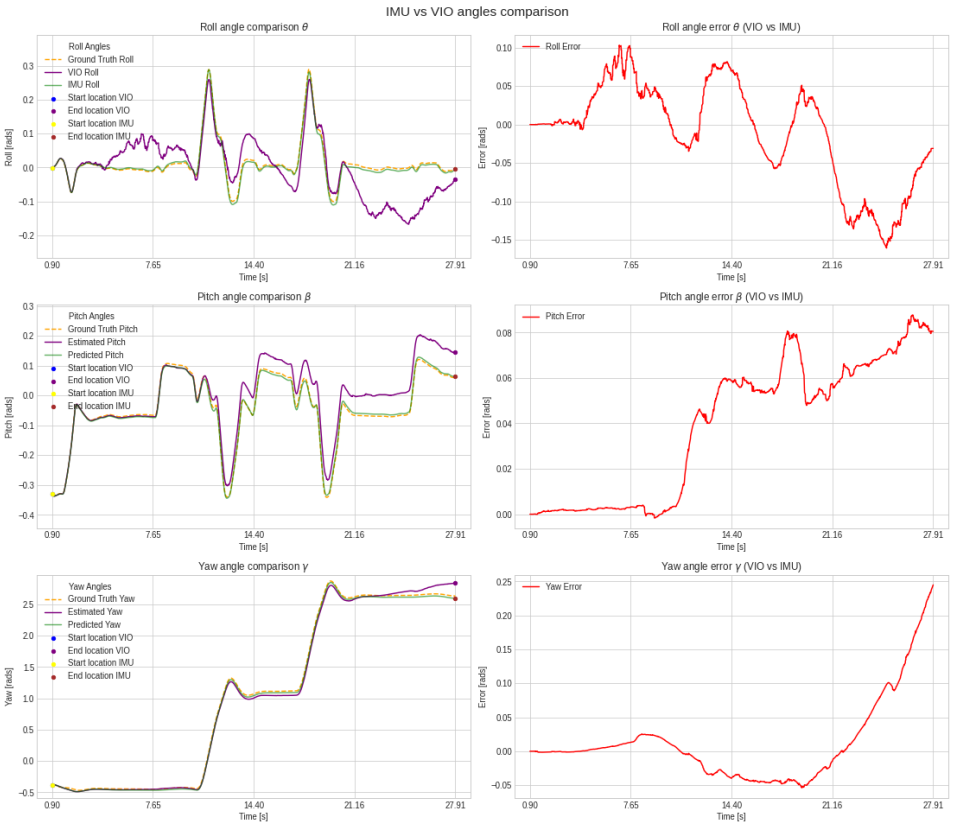


Figure 10.13: Depth camera: Comparison of angles of VIO, Predicted and Ground Truth, with the error between VIO and Predicted positions shown on the left

The filter also produces disappointing results, with errors larger than those observed in the previous scenario where the depth camera was not used. In fact, the trajectory estimated with the filter in this configuration is almost worse than the one calculated using only the IMU. This becomes particularly evident at position Y, where the error is significantly higher. For position X, while the filter provides a better estimate than the IMU alone, it still deviates significantly from the ground truth, in fact the error increases from about 1 meter in the previous case without depth to almost 17 meters in this configuration. Considering the barometer for the Z-axis, the trajectory estimated along Z continues to produce reliable results, showing good consistency with the real motion.

Chapter 11

Conclusions and Future developments

The main objective of this thesis was to analyze and implement a basic algorithm of VIO with a minimal set of sensors, consisting of monocular camera and IMU, capable of enabling the localization of a drone in a GPS-denied environment. The research aimed to understand how far it is possible to push this type of implementation while maintaining minimal hardware and reducing system complexity, yet ensuring accurate estimates of position and orientation.

Various approaches were tested for data fusion using an ES-EKF filter, evaluating the advantages and challenges of combining VO and IMU data for different spatial components. The best combination was achieved by fusing the x and y components from the VO, the attitude angles from the IMU and using a barometer for the z component. This choice was made because both the IMU and VO provided unreliable estimates along the z-axis, which significantly degraded the final performance of the filter, while using the barometer ensured more accurate and stable results.

A key aspect addressed in the study was the recovery of the relative scale, essential for obtaining consistent estimates in a context without absolute references. The scale was recovered by initializing the ES-EKF filter not with the value calculated between VO and GT but with a slightly lower optimal value. This approach compensated for both VO errors and initial scale estimation errors, improving the overall precision of the system.

Despite the promising results, the method has some limitations. The quality of the estimates heavily depends on the availability of reliable VO data, which can degrade in environments with poor visibility or texture. Moreover, IMU drift, if not properly modeled or compensated for, can negatively impact estimates. Lastly, managing the relative scale remains a key challenge, as initial errors in its determination can propagate over time and compromise the system's overall reliability. However, satisfactory results were achieved, as it was demonstrated that integrating the camera and IMU through the filter does indeed improve the final localization of the drone. Despite this improvement, significant errors remain when compared to the GT, which indicates that further enhancements are necessary to reduce these

discrepancies and improve overall accuracy.

Several improvements could be explored to enhance system performance. A more in-depth analysis of the algorithm with depth camera method represents a promising direction, as depth cameras provide additional spatial information that could complement the existing VO and IMU data, especially for improving the accuracy of depth estimation and scale recovery. By integrating depth data, the system could potentially obtain more reliable estimates of the environment's structure, which would help in reducing drift and contribute to more stable localization.

However, the use of depth cameras introduces new challenges that need further exploration. First, the environmental conditions play a crucial role in determining the effectiveness of depth sensors. For instance, depth cameras typically struggle in low-light conditions or when faced with transparent or reflective surfaces. Additionally, outdoor environments with large open spaces or complex, dynamic scenes can complicate depth perception. Therefore, understanding how environmental factors impact depth sensor performance is critical. Second, the usage parameters of depth cameras need to be carefully defined. These include the camera's resolution, field of view and range which directly affect the quality of depth information provided. For instance, a camera with a limited range might not capture sufficient depth information in large-scale environments, leading to poor localization estimates. Other parameters that impact the final result include those contained in the cv2. PnP RANSAC solver, such as the number of iterations and the reprojection error. These parameters must be carefully defined to ensure optimal performance and accuracy in the estimation process.

Moreover, effective fusion algorithms need to be developed to combine depth data with IMU and VO data in a way that optimizes the system's robustness and minimizes errors.

Lastly, integrating depth cameras into the existing system may offer a promising approach for updating scale estimates. One of the critical challenges in monocular VO is the ambiguity of scale, which is difficult to resolve without external references. Depth cameras could provide a direct way to recover scale by offering more precise depth measurements, thus allowing for better estimation of distances and more accurate positioning. However, while depth cameras have the potential to significantly enhance the system, their integration must be carefully optimized and further studies are necessary to fully understand their benefits and limitations in this context.

Further improvements could also come from implementing a Bundle Adjustment, which would simultaneously optimize position estimates and the intrinsic parameters of VO, reducing accumulated errors and enhancing overall consistency. Additionally, adopting a Tightly Coupled Approach instead of the Loosely Coupled Approach could result in greater accuracy. A Tightly Coupled Approach allows for closer integration between VO and IMU data, leveraging both sources more effectively and imposing stricter constraints, which reduce drift and enhance system robustness, particularly in complex scenarios or under unfavorable conditions for

one of the sensors.

This thesis demonstrated that promising results can be achieved using a minimal sensor set in a GPS-denied environment, leveraging VO and IMU data fusion via an ES-EKF filter. However, challenges remain regarding inertial drift, scale management and sensor consistency. The proposed approach provides a solid foundation but requires further optimizations to tackle more complex scenarios and improve system robustness and accuracy, particularly by exploring the potential of advanced sensors such as depth cameras.

Appendix A: Unscented Kalman filter

The Unscented Kalman Filter (UKF) proposed by Julier and Uhlman is an extension of the traditional KF. It is designed for state estimation in systems characterized by nonlinear dynamics and allows to get the best Gaussian approximaton of the input gaussian through a non-linear function f.

A fundamental operation of the KF is the propagation of a Gaussian random variable through system dynamics. In the EKF, the state distribution is represented by a Gaussian random variable and propagated via a first-order linearization of the nonlinear system. This method can introduce significant errors in the actual posterior mean and covariance of the transformed variable, possibly resulting in suboptimal performance and, in some cases, filter divergence.

The UKF addresses these issues by using a deterministic sampling technique called the unscented transformation. This approach seeks to approximate the mean and covariance of the state distribution using a set of carefully chosen sample points. The samples in the UKF are generated through a specific deterministic algorithm rather than being randomly selected. This allows to capture an high-order information about the distribution with a relatively small number of points. These sample points are chosen to accurately reflect the true mean and covariance of the Gaussian random variable and when processed through the actual nonlinear system they precisely capture the posterior mean and covariance up to the third order (in terms of Taylor series expansion) for any degree of non-linearity. In comparison, the EKF offers only first-order accuracy. Additionally, the computational complexity of the UKF is comparable to that of the EKF [51].

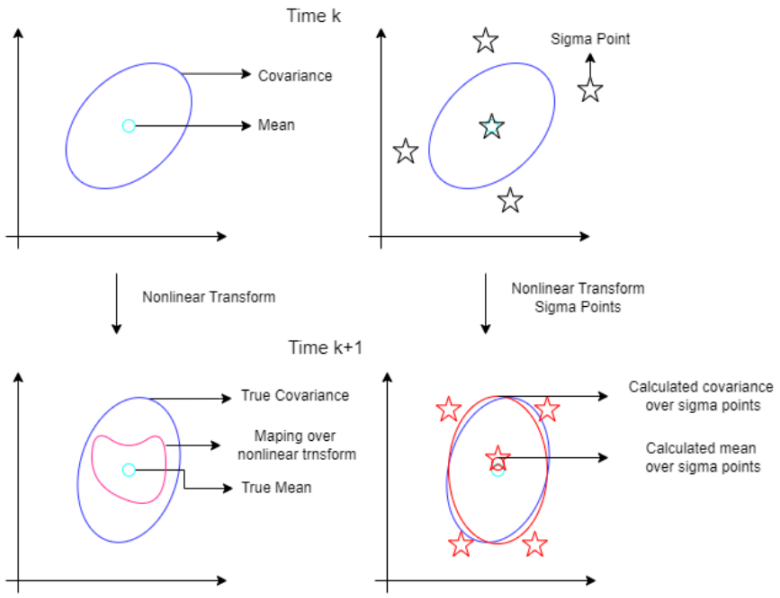


Figure 1: Unscented transform process: selecting sigma points and calculating posterior covariance and mean after transformation.

The problem of propagating a random variable x through a nonlinear function $y = g(x)$ is considered. It is assumed that x has a mean $\bar{x}$ and a covariance matrix P_x. For an L-dim Gaussian, 2L+1 sigma points with corresponding sigma weights W_i are computed according to the following:

$$
\begin{aligned}
\chi_0 &= \bar{x} \\
\chi_i &= \bar{x} + \left(\sqrt{(L+\lambda)P_x}\right)_i && i = 1, \ldots, L \\
\chi_i &= \bar{x} - \left(\sqrt{(L+\lambda)P_x}\right)_{i-L} && i = L+1, \ldots, 2L \\
W_0^{(m)} &= \frac{\lambda}{L+\lambda} \\
W_0^{(c)} &= \frac{\lambda}{L+\lambda} + (1 - \alpha^2 + \beta) \\
W_i^{(m)} &= W_i^{(c)} = \frac{1}{2(L+\lambda)} && i = 1, \ldots, 2L
\end{aligned}
$$

The scaling parameter $X = \alpha^2(L + \kappa) - L$ is introduced, where α determines the spread of the sigma points and is usually set to a small positive value. κ is a secondary scaling parameter, which is typically set to 0, and β is used to incorporate prior knowledge of the distribution of x (for Gaussian distributions, $\beta = 2$ is optimal). These sigma vectors are then propagated through the nonlinear function:

$$y_i = g(X_i), \quad i = 0, \ldots, 2L$$

and the mean and covariance for y are approximated using a weighted sample mean and covariance of the posterior sigma points:

$$\bar{y} \approx \sum_{i=0}^{2L} W_i^{(m)} y_i$$
$$P_y \approx \sum_{i=0}^{2L} W_i^{(c)} \{y_i - \bar{y}\} \{y_i - \bar{y}\}^T$$

Unscented Kalman Filter algorithm

The UKF is a recursive algorithm that shares the same fundamental structure as the standard KF, consisting of two main steps: the prediction step and the correction step. For the UKF to be effective, both the state-transition model and measurement model must be differentiable, allowing for accurate propagation of uncertainties through the system.

In the prediction step, the algorithm employs the unscented transformation, which generates a set of sigma points from the current state estimate. By applying the state-transition model to these sigma points, the UKF predicts the next state estimate, effectively capturing the non-linear dynamics of the system.

Unscented_Kalman_filter(μ_{t-1}, Σ_{t-1}, u_t, z_t):
$\chi_{t-1} = (\mu_{t-1} \quad \mu_{t-1} + \gamma\sqrt{\Sigma_{t-1}} \quad \mu_{t-1} - \gamma\sqrt{\Sigma_{t-1}})$
Prediction
$\bar{\chi_t^*} = g(\chi_{t-1}, u_t)$
$\bar{\mu}_t = \sum_{i=0}^{2n} w_m^{[i]} \bar{X_t^{*[i]}}$
$\bar{\Sigma}_t = \sum_{i=0}^{2n} w_c^{[i]} (\bar{X_t^{*[i]}} - \bar{\mu}_t)(\bar{X_t^{*[i]}} - \bar{\mu}_t)^T + R_t$
$\bar{\chi_t} = (\bar{\mu}_t \quad \bar{\mu}_t + \gamma\sqrt{\Sigma_t} \quad \bar{\mu}_t - \gamma\sqrt{\Sigma_t})$

Table 1: Prediction step Unscented Kalman filter algorithm

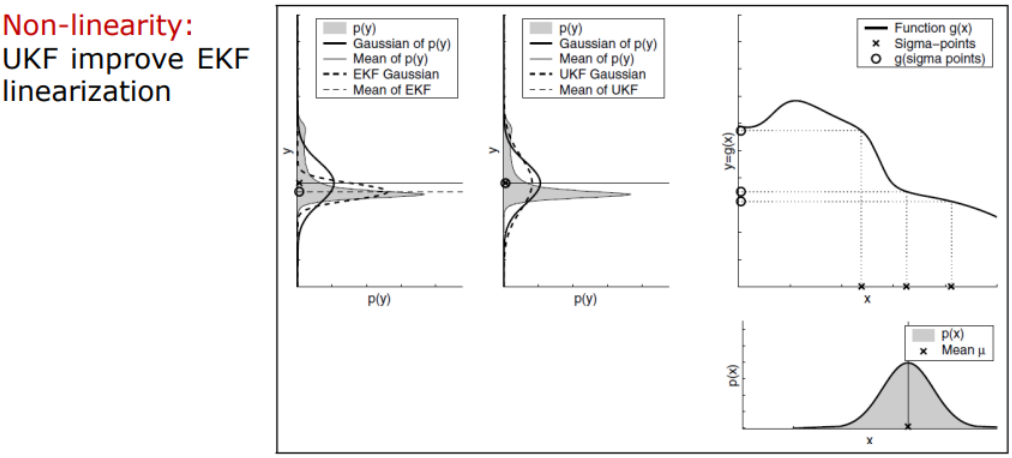
The correction step follows the prediction, where the predicted state estimate is used to derive predicted measurements through the measurement model. The UKF again applies the unscented transformation to the sigma points of the predicted state, generating a corresponding set of predicted measurements. These predicted measurements are then compared to the actual measurements obtained from the system. By evaluating the discrepancy between the predicted and actual measurements, the UKF corrects the state estimate, incorporating the new information to refine its predictions. This iterative process of prediction and correction allows the UKF to maintain an accurate and updated estimate of the system state, even in the presence of uncertainties and non-linearities.

Correction
$\bar{Z}_t = h(\bar{\chi}_{t-1})$
$\hat{z}_t = \sum_{i=0}^{2n} w_m^{[i]} \bar{Z}_t^{*[i]}$
$S_t = \sum_{i=0}^{2n} w_c^{[i]} (\bar{Z}_t^{[i]} - \hat{z}_t)(\bar{Z}_t^{[i]} - \hat{z}_t)^T + Q_t$
$\bar{\Sigma}_t^{x,z} = \sum_{i=0}^{2n} w_c^{[i]} (\bar{\chi}_t^{[i]} - \bar{\mu}_t)(\bar{Z}_t^{[i]} - \hat{z}_t)^T$
$K_t = \bar{\Sigma}_t^{x,z} S_t^{-}1$
$\mu_t = \bar{\mu}_t + K_t^{(} z_t - \hat{z}_t)$
$\Sigma_t = \bar{\Sigma}_t - K_t S_t K_t^T$
Return μ_t, Σ_t

Table 2: Correction step Unscented Kalman filter algorithm

Positive and negative aspects of Unscented Kalman filter

The UKF offers several advantages and some limitations compared to the EKF, especially when dealing with significant uncertainty and pronounced non-linearity in system behavior. A key strength of the UKF is its ability to deliver more accurate estimates in non-linear scenarios. Unlike the EKF, which depends on linearizing non-linear functions, the UKF utilizes the unscented transformation, allowing it to capture uncertainty propagation more effectively. This approach improves performance in generating Gaussian estimates, especially in regions with strong non-linearity, as illustrated in Figure 2. The UKF employs a set of sigma points that cover a broader range of input uncertainty, thus using more information during the transformation process.



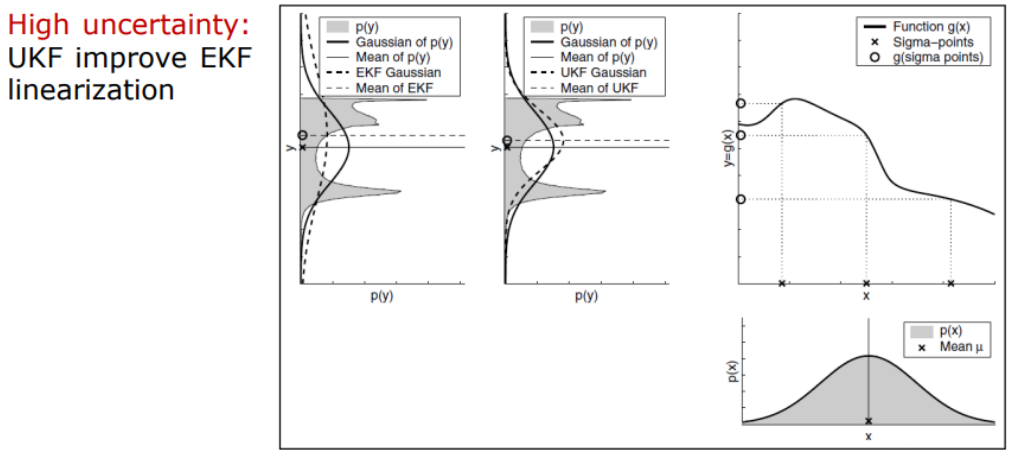


Figure 2: Comparison of state estimation under high non-linearity and uncertainty: The UKF effectively captures the propagation of uncertainty, leading to more accurate results than the EKF

However, despite generally outperforming the EKF under non-linear conditions, the UKF does come with increased computational demands, although the asymptotic complexity of the UKF algorithm is comparable to that of the EKF. The EKF can often execute slightly faster due to its simpler computations.

Furthermore, both the UKF and EKF face limitations in solving position tracking problems for global navigation due to their assumption of Gaussian distribution, which restricts their use in scenarios that require multi-modal distributions. Another important aspect is the choice of appropriate sigma points and weights, it is critical to the UKF's performance and requires careful tuning and optimization for optimal results.

Bibliography

[1] Yusra Alkendi, Lakmal Seneviratne, and Andy Yahyazweiri. State of the art in vision-based localization techniques for autonomous navigation systems. *IEEE Access*, 9:67999–68018, 2021. Received May 1, 2021; accepted May 17, 2021; date of publication May 21, 2021; date of current version June 2, 2021.

[2] Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry: Part i - the first 30 years and fundamentals. *IEEE Robotics and Automation Magazine*, 18(4), 2011.

[3] Kenji Hata and Silvio Savarese. Cs231a course notes 1: Camera models. `https://web.stanford.edu/class/cs231a/course_notes/01-camera-models.pdf`.

[4] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, Cambridge, UK, 2nd edition, 2003.

[5] De Jong Yeong, Gustavo Velasco-Hernandez, John Barry, and Joseph Walsh. Sensor and sensor fusion technology in autonomous vehicles: A review. *Sensors*, 21:2140, 03 2021.

[6] Furrukh Sana. *Modeling and Estimation for Maneuvering Target Tracking with Inertial Systems using Interacting Multiple Models*. PhD thesis, 07 2010.

[7] Renaud Hage, Christine Detrembleur, Frédéric Dierick, Laurent Pitance, Laurent Jojczyk, Wesley Estievenart, and Fabien Buisseret. Dyskimot: An ultra-low-cost inertial sensor to assess head's rotational kinematics in adults during the didren-laser test. *Sensors*, 20(3):833, February 2020.

[8] Mohammad O. A. Aqel, Mohammad H. Marhaban, M. Iqbal Saripan, and Napsiah Bt. Ismail. Review of visual odometry: Types, approaches, challenges, and applications. *SpringerPlus*, 2016.

[9] H. Jung, Yun Hee Lee, Dong Suk Kim, and P. Yoon. Stereo vision based advanced driver assistance system. *Computer Science and Engineering*, December 2005.

[10] Noah Snavely. Lecture 4: Harris corner detection. Accessed: 2024-11-24.

[11] Andry Maykol Pinto, Maria Inês Pereira, et al. A practical survey on visual odometry for autonomous driving in challenging scenarios and conditions. *IEEE Access*, 10:84941–84966, August 2022.

[12] Asmat Zahra. What is computer vision and its applications in the real world? a friendly guide to what is computer vision, its applications, and associated challenges. *Medium*, September 2021. 4 min read.

[13] Robert Collins. Lecture 06: Harris corner detector, 2024. CSE486, Penn State, Reading: TV Section 4.3.

[14] Oluibukun Ajayi. Performance analysis of selected feature descriptors used for automatic image registration. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLIII-B3-2020:559–566, 08 2020.

[15] Chuan Luo, Wei Yang, Panling Huang, and Jun Zhou. Overview of image matching based on orb algorithm. *Journal of Physics: Conference Series*, 2019.

[16] Friedrich Fraundorfer and Davide Scaramuzza. Visual odometry: Part ii - matching, robustness, and applications. *IEEE Robotics and Automation Magazine*, 2012.

[17] Eugene Khvedchenya. Comparison of the opencv's feature detection algorithms – ii. `https://computer-vision-talks.com/2011-07-13-comparison-of-the-opencv-feature-detection-algorithms/`, 2011.

[18] Davide Scaramuzza. Tutorial on visual odometry. Online Tutorial.

[19] Kai Cao, Xuemeng Yang, Song Gao, Chaobo Chen, Jiaoru Huang, and Xiaoru Song. Visual odometry based on 3d-3d and 3d-2d motion estimation method. In *2018 Chinese Automation Congress (CAC)*, pages 3643–3648, 2018.

[20] Camcalib Team. What is the reprojection error, 2022.

[21] Abiel Aguilar-González, Miguel Arias-Estrada, and François Berry. Dense feature matching core for fpga-based smart cameras. pages 41–48, 09 2017.

[22] Davide Scaramuzza and Zichao Zhang. Visual-inertial odometry of aerial robots. In *Springer Encyclopedia of Robotics*. Springer, 2019. Accepted for publication.

[23] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. Stanford University, University of Freiburg, University of Washington, Stanford, CA, Freiburg, Germany, Seattle, WA, 1999–2000. Early draft—Not for distribution.

[24] Joan Solà. Quaternion kinematics for the error-state kalman filter. Technical report, University of Barcelona, November 2017. Accessed: November 8, 2017.

[25] PX4. Px4 software overview, 2024. Accessed: Nov. 21, 2024.

[26] Yafei Hu. Visual odometry tutorial, July 2020. Presentation.

[27] PX4 Development Team. Px4 gazebo simulation: Vehicles.

[28] Syed Agha Hassnain Mohsan, Muhammad Asghar Khan, Fazal Noor, Insaf Ullah, and Mohammed H. Alsharif. Towards the unmanned aerial vehicles (uavs): A comprehensive review. *Drones*, 6(6):147, June 2022. This article belongs to the Special Issue Security, Privacy and Reliability of Drone Communications for beyond 5G Networks.

[29] Davide Scaramuzza, Michael Achtelik, Friedrich Fraundorfer, et al. Vision-controlled micro flying robots from system design to autonomous navigation and mapping in gps-denied environments. *IEEE Robotics & Automation Magazine*, 21(3):30–39, September 2014.

[30] Stephan M. Weiss. *Vision Based Navigation for Micro Helicopters*. Doctoral dissertation, ETH Zurich, Zurich, Switzerland, 2012. Accepted on the recommendation of Prof. Roland Siegwart, Prof. Vijay Kumar, and Dr. Agostino Martinelli.

[31] Cyrill Stachniss. What do cameras actually measure explained in 5 minutes, 2021. Series: 5 Minutes with Cyrill. Special thanks to Olga Vysotska and Igor Bogoslavskyi. Partial image courtesy Brunox983@pixabay. Intro music by The Brothers Records.

[32] Advanced Navigation. Inertial measurement unit (imu): An introduction, 2024. Last edited: June 7, 2024.

[33] Oliver J. Woodman. An introduction to inertial navigation. Technical Report Research Report 696, University of Cambridge, August 2007. Cited on pp. 38, 39, 42.

[34] Mohammad Aqel, Mohammad Hamiruce Marhaban, M Iqbal Saripan, and Napsiah Ismail. Review of visual odometry: types, approaches, challenges, and applications. *SpringerPlus*, 5, 12 2016.

[35] Akshay Kumar Burusa. Visual-inertial odometry for autonomous ground vehicles, 2017. Degree Project in Computer Science and Engineering, Second Cycle, 30 Credits, Stockholm, Sweden.

[36] L. Frédéric. The visual compass: Performance and limitations of an appearance-based method. *Journal of Field Robotics*, 33(1):1–17, 2006.

[37] Ahmed El Amin and Ahmed El-Rabbany. Monocular vo scale ambiguity resolution using an ultra low-cost spike rangefinder. *Open Journal of Civil Engineering*, 10(4):27–38, 2020. Department of Civil Engineering, Ryerson University.

[38] Ali Ismail Awad and Mahmoud Hassaballah, editors. *Image Feature Detectors and Descriptors: Foundations and Applications*. Springer International Publishing, Switzerland, 1st edition, 2016.

[39] Khalid Yousif, Alireza Bab-Hadiashar, and Reza Hoseinnezhad. An overview to visual odometry and visual slam: Applications to mobile robotics. *Intelligent Industrial Systems*, 1:289–311, 2015. Received: 4 July 2015 / Revised: 30 October 2015 / Accepted: 1 November 2015 / Published online: 13 November 2015.

[40] Ebrahim Karami, Siva Prasad, and Mohamed Shehata. Image matching using sift, surf, brief and orb: Performance comparison for distorted images. 11 2015.

[41] Natasha Govender. Evaluation of feature detection algorithms for structure from motion. *Mobile Intelligent Autonomous Systems*, 2009.

[42] H. Goldstein. *Classical Mechanics*. Addison-Wesley, Reading, MA, 1981.

[43] D. Nister, O. Naroditsky, and J. Bergen. Visual odometry . In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages I–652–I–659 Vol.1, Los Alamitos, CA, USA, July 2004. IEEE Computer Society.

[44] V. Madyastha, V. Ravindra, S. Mallikarjunan, and A. Goyal. Extended kalman filter vs. error state kalman filter for aircraft attitude estimation. In *AIAA Guidance, Navigation, and Control Conference*. American Institute of Aeronautics and Astronautics, June 2011.

[45] Zhu Guo, Zhou and Bai. Low-cost sensors state estimation algorithm for a small hand-launched solar-powered uav. *Sensors*, 19:4627, October 2019.

[46] W. Youn and S. Andrew Gadsden. Combined quaternion-based error state kalman filtering and smooth variable structure filtering for robust attitude estimation. *IEEE Access*, 7:148989–149004, 2019.

[47] Lovro Markovic, Marin Kovač, Robert Milijaš, Marko Car, and Stjepan Bogdan. Error state extended kalman filter multi-sensor fusion for unmanned aerial vehicle localization in gps and magnetometer denied indoor environments, 09 2021.

[48] PX4 Development Team. Gazebo simulation with px4, 2021. Accessed: 2024-11-21.

[49] David G. Lowe. Distinctive image features from scale-invariant keypoints. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 1150–1157, Vancouver, B.C., Canada, January 2004. lowe@cs.ubc.ca.

[50] Arize AI. Root mean square error (rmse): What you need to know, 2024. Accessed: 2024-11-22.

[51] Eric A. Wan and Rudolph van der Merwe. The unscented kalman filter for nonlinear estimation. In *Proceedings of the 2000 American Control Conference*, 20000 NW Walker Rd, Beaverton, Oregon 97006, 2000.