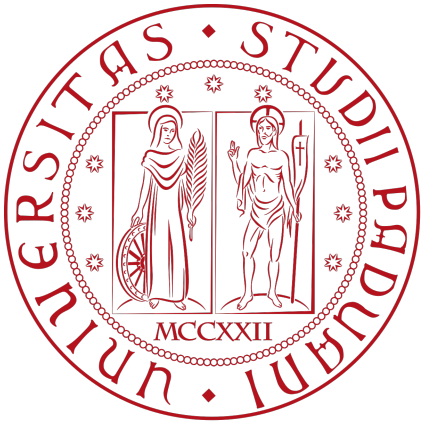
UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA

DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA



ELABORATO DI LAUREA

Design, implementation and deployment of a visual odometry system based on a stereovision module

Relatore

Prof. Damiano Varagnolo

Laureando

Giovanni Giaretta

ANNO ACCADEMICO 2024/2025

Abstract

This thesis presents the development of a stereo visual odometry system designed as part of the Nautilus student project, aimed at supporting navigation tasks in challenging environments. Visual odometry allows estimating the motion of a camera-equipped system based solely on visual information, making it particularly valuable in scenarios where GPS or other localization methods are unavailable. The system was tested on three different datasets: the KITTI benchmark, a custom dataset acquired in a controlled environment, and a third collected along the canals of Chioggia during a Nautilus field mission. The analysis highlights both the strengths and limitations of the system in varying conditions. The thesis concludes by discussing possible future developments for the improvement of the system in terms of robustness and accuracy, considering possible future underwater environments.

Contents

Chapter 1

Research interest

1.1 General introduction

1.1.1 What is visual odometry?

The term odometry is historically associated with the estimation of the position of a wheeled vehicle on the basis of the distance traveled by the latter and the steering angle. The word derives, in fact, from the Greek "odos" (road) + "metron" (measure): to measure the distance traveled on the road.
However, today this definition would be a great reduction. The goal, of course, coincides, but the data can come from various types of sensors. For example, they can be images from cameras, so we talk about visual odometry. For now, we will briefly delve into the latter, which will be the focus of the thesis. An overview of the other types of sensors will be presented in Section 1.1.3.
Visual odometry (VO) is the process of determining the position and orientation of an agent by analyzing only the input of a single or multiple cameras attached to it. The first research on this technology was carried out starting from the 1980s, and a good part of it concerned the use of the VO in the design of modules that trace the trajectory of planetary rovers (research motivated by the NASA Mars exploration program). In fact, an odometry method was wanted that did not suffer from errors due to wheel slippage on "difficult" and not well known terrains. In this sense, visual odometry provided a supplement to calculate the 6-degree-of-freedom (this describes the full extent of motion

possible for an object in a three-dimensional space, encompassing three linear and three angular degrees of freedom).
There are two main approaches to develop a VO system: monocular, which uses a single camera, and stereoscopic, which uses a pair of cameras. In this thesis the stereo approach has been chosen as it allows us to recover information of absolute scale thanks to the knowledge of the distance between the two cameras. A more in-depth discussion of the geometric foundations necessary to understand them is provided in Chapter 2.

1.1.2 Why is important for localization and navigation

In autonomous navigation systems, the ability of an agent to estimate its position and orientation in real time is a fundamental functionality. This information is, in fact, crucial for numerous modules within an autonomous system, including:

- Path planning: planning an optimal path to a destination necessarily requires knowledge of the current position. Without this information, the agent would not be able to determine in which direction to move or how to avoid obstacles.

- Localization and perception of the environment: visual odometry provides a continuous and local estimate of the position of the agent with respect to the surrounding environment. By integrating these data with other perception modules, such as object recognition or semantic segmentation, it is possible to build three-dimensional maps updated in real time (SLAM). In this sense, the VO acts as a sort of dynamic "reference system": it allows to determine where to correctly place each new piece of information collected. For example, if the agent identifies an obstacle, it is thanks to the VO that it is possible to associate a coherent position in space to that object.

1.1.3 Comparison with other odometry systems

Other odometry systems are briefly presented and, subsequently, the pros and cons of VO are listed, comparing it with the others.

Wheel odometry

It is the simplest and most direct method to estimate the displacement of a vehicle. It is based on the use of wheel encoders, devices that measure the number of revolutions of the wheels, converting it into a displacement with respect to the ground. Wheel odometry is a relative positioning technique (i.e. a technique that provides the current position with respect to a known starting point, without absolute references). One of the main weak points, in addition to the fact that it can only be used on vehicles with wheels, is the susceptibility to accumulation error over time due to wheel slippage. It is a simple, inexpensive method and guarantees good accuracy in the short term.

IMU/INS

The Inertial Measurement Unit (IMU) is a device that measures linear acceleration and angular velocity using accelerometers and gyroscopes. While the IMU itself does not compute position or orientation, its measurements can be integrated over time to estimate them.

However, due to the accumulation of small measurement errors and noise during numerical integration, this process leads to a rapid increase in estimation error, a phenomenon known as drift. For this reason, raw IMU data is typically only useful for short-term motion estimation.

To improve accuracy and enable long-term navigation, IMUs are commonly integrated into an Inertial Navigation System (INS), which combines the IMU with additional algorithms (e.g., Kalman filtering) and sometimes external references such as GPS or visual sensors. This allows the system to correct for drift and produce more reliable estimates of position, velocity, and orientation over time.

Like wheel odometry, inertial-based approaches are relative positioning techniques, meaning they estimate motion with respect to an initial known state.

GNSS/GPS

In common language, the term GPS (Global Positioning System) is used to generically indicate any satellite location system. In reality, GPS is the name of the American system, but there are also other operational constellations such as GLONASS(Russia), GALILEO(European Union) and Beidou(China). The correct term to refer to all these systems together is GNSS(Global Navigation Satellite System). Modern GNSS receivers are capable of combining signals from multiple constellations, improving coverage and reliability. For this reason, in the following we will refer to the satellite location system with the term GNSS.
A GNSS receiver provides three main types of information:

- Position: estimate of the absolute location of the receiver on Earth(latitude, longitude and altitude);
- Navigation: real-time tracking of the movement, obtained by comparing successive positions;
- Time: extremely precise synchronization thanks to the atomic clocks on board the satellites, with accuracies of the order of nanoseconds.

The calculation of the position is based on the principle of trilateration: the position in space is determined by knowing the distance to at least three known points. In the case of GNSS, these points are satellites, whose position is known with great precision thanks to the ephemerides, information transmitted in the signal itself. The receiver measures the arrival time of the signal from each satellite and multiplies it by the speed of light c to obtain the distance. Theoretically, knowing the distance to three satellites, it is possible to estimate the three spatial coordinates by solving the intersection of three spheres. However, in practice, this estimate requires that the receiver's clock is perfectly synchronized with those of the satellites, which is not realistic: satellites have atomic clocks, while receivers do not. To overcome this asynchrony, a fourth satellite is needed, which allows us to introduce an additional unknown: the receiver time delay. This results in a system of four nonlinear equations in four

unknowns:

$$\begin{cases} \sqrt{(x-x_1)^2+(y-y_1)^2+(z-z_1)^2} = c(t-t_1+\delta t) \\ \sqrt{(x-x_2)^2+(y-y_2)^2+(z-z_2)^2} = c(t-t_2+\delta t) \\ \sqrt{(x-x_3)^2+(y-y_3)^2+(z-z_3)^2} = c(t-t_3+\delta t) \\ \sqrt{(x-x_4)^2+(y-y_4)^2+(z-z_4)^2} = c(t-t_4+\delta t) \end{cases} \tag{1.1}$$

Dove:

- (x_i, y_i, z_i) sono le coordinate del satellite i;
- t_i è il tempo di trasmissione del segnale dal satellite i;
- δt è l'errore temporale dell'orologio del ricevitore.

This extra measurement is crucial: for example, a phase shift of just 1μs in the time measurement would result in a spatial error of about 300 meters, since $c \approx 3 \times 10^8\,\mathrm{m/s}$

The main advantages of GNSS are absolute position and the absence of error accumulation (it is not a relative technique). A standard GNSS system offers an accuracy of about 10 meters. More advanced technologies such as Differential GNSS or Real-Time Kinematic GNSS(RTK-GNSS) can increase accuracy to the centimeter, but require complex and expensive infrastructure, and are therefore less widespread in general applications.

Sonar/Ultrasonic sensors

Sonar/ultrasonic sensors consist of two main components: the transmitter and the receiver. The transmitter emits a short pulse of ultrasonic sound waves, and the receiver captures the signal that has been reflected by objects in the surrounding environment. The sensor then measures the Time of Flight(TOF), the time it takes for the signal to travel the round trip. Knowing the speed of sound, which in air is approximately $c = 343\,\mathrm{m/s}$ (at room temperature), it is possible to calculate the distance of the object from the sensor.

$$d = \frac{c \cdot \mathrm{TOF}}{2}$$

Dove:

- d è la distanza,
- c è la velocità del suono, che in aria è circa $343\,\mathrm{m/s}$,
- TOF è il tempo di volo, ovvero il tempo che il segnale impiega a percorrere il tragitto di andata e ritorno.

By collecting this data along a path, it is possible, through a process of triangulation, to reconstruct the trajectory of the agent. With the use of multiple acoustic sensors, a detailed map of the surrounding environment can be obtained with high position accuracy.
However, one of the main limitations of this technology concerns the reflection of sound waves: the ability to detect an object strongly depends on the material and orientation of the reflecting surface. Soft or irregular surfaces, for example, tend to absorb or scatter the signal, reducing the reliability of the measurement. Furthermore, the environment can introduce significant background noise, such as sound waves from other acoustic sources (for example, other robots or machines), which interfere with the signal.

LiDAR

Laser sensors, commonly known as LiDAR(Light Detection and Ranging), use laser light to measure distances. Unlike acoustic sensors, which use sound waves, LiDAR sensors emit pulses of light that are reflected by nearby objects. The distance between the sensor and the object is calculated by measuring the Time of Flight(TOF), which is the time it takes for the light signal to travel there and back.
Compared to ultrasonic sensors, LiDARs offer much higher resolution and greater precision, allowing them to generate a detailed 3D map of the surrounding environment. They are widely used in applications such as environment mapping, odometry, obstacle detection and, more recently, autonomous navigation. However, LiDAR sensors also have some disadvantages: they are more expensive and require high computational processing, which can affect real-time performance, especially in complex and dynamic environments.

So why VO?

Each odometry technique analyzed has specific advantages, but also important limitations. Wheel odometry, for example, is simple and cheap, but it suffers severely from error accumulation over time due to wheel slippage. IMU/INS systems offer good short-term reactivity, but necessarily require external correction to avoid significant drift. GNSS, on the other hand, is very stable over the long term and immune to error accumulation, but has low local resolution, does not work well underwater, in enclosed spaces, or in highly urbanized environments, and offers limited precision in its standard configuration. Sensors such as sonar and LiDAR can offer very precise environmental maps, but have high costs or operational fragilities (interference, problematic reflections).
Visual odometry fits into this context as a powerful compromise: using simple cameras, often already available on board autonomous vehicles, it is able to provide an accurate estimate of the relative movement between consecutive frames. VO is quite robust, relatively cheap and can work both indoors and outdoors, in urban or natural context. Although it is also a relative positioning technique and is subject to error accumulation, it can be easily integrated with other sources (e.g. GNSS,IMU) in a sensor fusion framework to improve overall performance. The choice of this thesis to study VO arises, therefore, from its balance between cost, accuracy, versatility and from the computational and geometric challenges it poses, making it a fundamental component for autonomous localization in modern robotic systems.

1.2 Applications

Visual odometry has found extensive use in a wide range of robotics and autonomous navigation systems. One of its primary applications is in aerial robotics, particularly in Unmanned Aerial Vehicles (UAVs), where stereo VO, often fused with inertial measurements, enables autonomous flight in GPS-denied environments. For instance, the visual–inertial odometry pipeline developed by George et al. [1] is specifically designed for high-flying drones and

has shown robust performance in aerial surveys at altitude.

In underwater robotics, stereo imagery has been used to support localization where GNSS is unavailable. Eustice et al. [2] demonstrated visually-aided dead reckoning and mapping using stereo cameras for Autonomous Underwater Vehicles (AUVs), showing its effectiveness in challenging visual conditions. In the automotive sector, stereo VO is frequently integrated into multi-sensor perception systems for autonomous driving. The KITTI Vision Benchmark Suite [3] provides a widely used dataset and evaluation framework for stereo VO and visual odometry-based SLAM systems in real urban environments. Such systems are often fused with GNSS and LiDAR data for improved robustness and redundancy. Stereo VO has also proven effective in space exploration. NASA's Mars rovers, such as *Curiosity*, employ stereo camera systems for visual odometry on the Martian surface [4], compensating for unreliable wheel odometry due to terrain-induced slippage.

As seen, in all these domains, stereo VO is rarely used in isolation. Instead, it is typically embedded within sensor fusion frameworks that combine inertial data, GNSS, LiDAR, or wheel encoders to mitigate drift and enable long-term localization in complex, dynamic environments. Moreover, many practical applications adopt multi-camera systems to provide a 360-degree view of the surroundings.

1.3 Challenges

Visual odometry has shown great potential in a variety of indoor applications, but its deployment in real-world scenarios still poses significant challenges. The correct functioning of VO depends in specific environmental conditions: good lightning, sufficient textures on the scene, and static environment. When these conditions are lacking, the accuracy of motion can significantly degrade. Many real-world environments do not satisfy these requirements. Dynamic scenes, homogeneous or reflective surfaces, changes in light due to shadows or directional lighting are common sources of error in displacement estimation. Furthermore, adverse weather condition or interaction with other agents

(people, vehicles) can introduce unpredictable variations of the scene. These difficulties are amplified in extreme environments such as the underwater one, where we would like to extend the developed VO system. In this context, natural light fades rapidly with depth, and images are frequently disturbed by particles suspended in the water(backscatter). Surfaces are often poorly textured and the environment is dynamic , with marine fauna or water flows altering the scene. Added to this is the luminuos variation due to the refraction and selective absorption of the different wavelengths of light.

1.4 Thesis objective

This thesis focuses on the implementation of a stereo visual odometry system to estimate the motion of an agent in static and sufficiently structured environments. The work represents the first step towards a more complete visual localization system, designed for applications in complex scenarios such as underwater environments. Although the current pipeline does not yet include mechanisms for handling dynamic scenes or artificial lighting, it lays the foundations for future extensions that allow adaptation to more challenging conditions. The main contribution lies in the development of the complete VO pipeline, from calibration to motion estimation, with particular attention to the modularity of the system. Moreover, considering the intended deployment in real-time applications with constrained hardware, computationally expensive methods such as deep learning-based approaches are deliberately excluded from this work.

Chapter 2

Theoretical foundations

Imagine we are on a railway line, looking towards the horizon. We know that the tracks are parallel lines that, in the real world, never meet. Yet, with the naked eye (or through a camera), we observe a curious phenomenon: the tracks seem to get progressively closer until they meet at a distant point. How do we explain this apparent paradox?
The answer lies in projective space, a mathematical construct that extends Euclidean space to model visual perception. It is the space in which images, lines to infinity, vanishing points and, more generally, the geometry of artificial vision "live".

2.1 Projective space

2.1.1 What is projective space?

Projective space is an extension of Euclidean space, to which points at infinity are added. This allows lines that live in projective space to always intersect: if two lines are not parallel, they meet at a finite point; otherwise, they meet at a point at infinity. This eliminates the exception of Euclidean space, where parallel lines never meet.
In this way we can model real phenomena such as central projection, in which parallel lines in the real world converge at a point at infinity (called vanishing point) in the image. The photo we took, or an instant of the world seen with

our eyes, lives precisely in this projective space.

A point (x, y) in Euclidean space is represented in projective space by adding an extra coordinate, resulting in the homogeneous coordinates $(x, y, 1)$. It is also stated that the points $(x, y, 1)$ and (kx, ky, k) with $k \neq 0$ represent the same point in projective space. For example:

$$(12, 6, 3) \equiv (4, 2, 1),$$

corresponding to the euclidean point $(4, 2)$ Note that, at the point $(x, y, 0)$, there is no corresponding point in Euclidean space. This is because dividing by zero gives a point at infinity:

$$\left(\frac{x}{0}, \frac{y}{0}\right) \to \infty$$

In this way, the points at infinity are represented in the projective space. The reasoning naturally extends to more dimensions: the Euclidean space $\mathbb{R}^n$ is extended to the projective space $\mathbb{P}^n$. In particular, $\mathbb{P}^2$ can be modeled as the set of rays that start from the origin of $\mathbb{R}^3$, where each point is a direction (the points that differ by a scalar factor k form a ray). In this model:

- the lines in $\mathbb{P}^2$ correspond to planes passing through the origin in $\mathbb{R}^3$;
- two distinct rays determine a plane (therefore a straight line in $\mathbb{P}^2$);
- two planes intersect in a ray (a point in $\mathbb{P}^2$).

The rays passing through the origin of $\mathbb{R}^3$ are projected onto the plane $x_3 = 1$. The points at infinity correspond to the rays parallel to this plane (i.e. those with $x_3 = 0$).

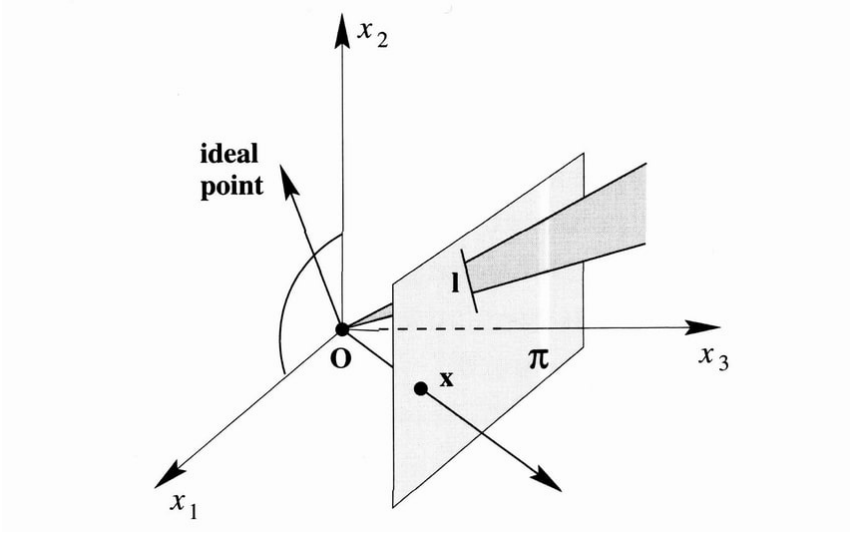


Figure 2.1: A model of the projective space $\mathbb{P}^2$ as the set of rays emanating from the origin 3d Euclidean space. Reproduced from Hartley and Zisserman, *Multiple View Geometry*, 2004, Figure 2.1. Used for academic purposes only.

2.1.2 Straight lines in projective space

In projective space, a line is also represented in homogeneous form by a vector $\mathbf{l} = (a, b, c)^\top$, which corresponds to the line equation $ax + by + c = 0$. This can be compactly written using the dot product as $\mathbf{l}^\top \mathbf{x} = 0$, where $\mathbf{x}$ is a point in homogeneous coordinates.

Given a point $\mathbf{x}$ and a line $\mathbf{l}$, one can:

- Determine whether $\mathbf{x}$ lies on $\mathbf{l}$ by checking if $\mathbf{l}^\top \mathbf{X} = 0$;
- Find the line passing through two points $\mathbf{x}_1$ and $\mathbf{x}_2$ using the cross product: $\mathbf{l} = \mathbf{x}_1 \times \mathbf{x}_2$;
- Find the intersection point of two lines $\mathbf{l}_1$ and $\mathbf{l}_2$ via the cross product: $\mathbf{x} = \mathbf{l}_1 \times \mathbf{l}_2$.

As an example, consider two horizontal lines $y = 1$ and $y = 2$, which are parallel in Euclidean space. These correspond to projective lines

$$\mathbf{l}_1 = \begin{pmatrix} 0 \\ -1 \\ 1 \end{pmatrix}, \quad \mathbf{l}_2 = \begin{pmatrix} 0 \\ -1 \\ 2 \end{pmatrix}.$$

Their intersection point is given by the cross product:

$$\mathbf{x} = \mathbf{l}_1 \times \mathbf{l}_2 = \begin{pmatrix} 0 \\ -1 \\ 1 \end{pmatrix} \times \begin{pmatrix} 0 \\ -1 \\ 2 \end{pmatrix} = \begin{pmatrix} (-1)(2) - (1)(-1) \\ (1)(0) - (0)(2) \\ (0)(-1) - (-1)(0) \end{pmatrix} = \begin{pmatrix} -2+1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix}.$$

This point $\mathbf{x} = (-1, 0, 0)^T$ lies on the line at infinity and corresponds to the vanishing point of horizontal lines. This example illustrates how in projective space, even parallel lines intersect — at a point at infinity.

2.1.3 Conics in projective space

A conic is the locus of points that satisfy a second-degree equation in two variables. In the Euclidean plane, this is typically written as:

$$ax^2 + bxy + cy^2 + dx + ey + f = 0,$$

and it represents curves such as ellipses, parabolas, and hyperbolas.
In projective geometry, this equation is homogenized by introducing homogeneous coordinates. The general conic equation becomes:

$$\mathbf{x}^\top \mathbf{C} \mathbf{x} = 0,$$

where $\mathbf{x} = (x, y, 1)^\top$ is a point in homogeneous coordinates, and $\mathbf{C}$ is a 3×3 symmetric matrix that encodes the parameters of the conic. This compact representation has several advantages: it unifies all types of conics, including degenerate ones (such as pairs of lines), and it fits into the projective framework where points at infinity play a central role. Conics are fundamental tools in computer vision, especially in camera calibration, application which will be explored in later chapters.

2.1.4 Projective transformation

Imagine capturing a photograph of a vase from two different viewpoints. The second image can be seen as the result of a projective transformation applied

to the first one. By working in projective space and using homogeneous coordinates, we can represent such transformations, which are generally complex and do not preserve angles or distances but do preserve straight lines, as linear operations on vectors.

This is one of the main advantages of projective space: complex geometric transformations become linear manipulations, greatly simplifying analysis and computation in computer vision applications. Formally, a planar projective transformation (also called a homography) is a linear transformation from the projective plane $\mathbb{P}^2$ to itself, applied to a 2D point expressed in homogeneous coordinates, and represented by a non-singular 3×3 matrix H:

$$\mathbf{x}' = H\mathbf{x}$$

where $\mathbf{x}, \mathbf{x}' \in \mathbb{P}^2$ are homogeneous vectors representing corresponding points before and after the transformation. Since we work with homogeneous coordinates, the transformation is defined up to scale, only the ratio between the components matters. Lines transform differently: given a line $\mathbf{l}$, the transformed line $\mathbf{l}'$ is given by:

$$\mathbf{l}' = H^{-\top}\mathbf{l}$$

Similarly, conics follow the transformation rule:

$$C' = H^{-\top}CH^{-1}$$

These relations allow for manipulation of geometric primitives under perspective projection.

2.2 Camera Models

In this section, we aim to define a model that approximates the behavior of a physical camera. A camera performs a mapping from the three-dimensional world to a two-dimensional image plane. Several classes of camera models exist, depending on factors such as the number of projection centers, the direction

of light rays, and their geometric configuration. The model of interest in this thesis is the one most commonly adopted in computer vision, as it represents the behavior of real-world cameras: the central projection model. In this case, all rays pass through a single projection center, resulting in what is referred to as a canonical projection. From this point onward, whenever we refer to camera models, we will implicitly refer to this class.

2.2.1 Simplified pinhole model

The pinhole camera model is the most basic and specific type of camera model. In this setting, all rays of light converge at a single projection center. We assume this center lies at the origin of a Euclidean coordinate system, and that the image plane is located at $z = f$, where f denotes the focal length. Now, consider a 3D point in space $\mathbf{X} = (X, Y, Z)^T$ and the ray connecting it to the projection center. In the pinhole model, the point $\mathbf{X}$ is projected onto the image plane at the point where the ray intersects it.

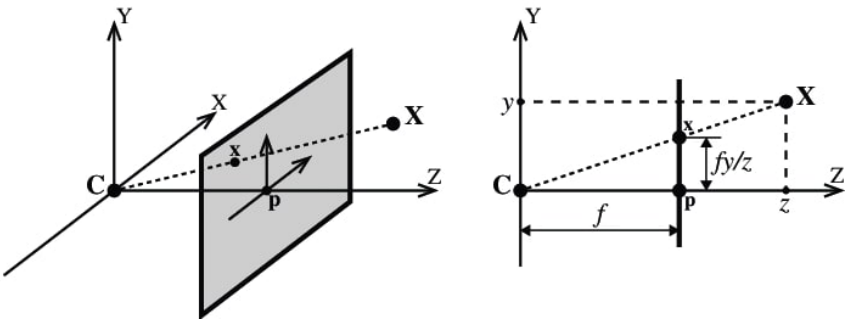


Figure 2.2: Pinhole model

From the similarity of triangles, we obtain the following relationships:

$$\frac{x}{X} = \frac{f}{Z}, \quad \frac{y}{Y} = \frac{f}{Z} \quad \Rightarrow \quad x = \frac{fX}{Z}, \quad y = \frac{fY}{Z}.$$

Thus, a 3D point $\mathbf{X} = (X, Y, Z)^T$ is mapped to a point on the image plane at coordinates $(x, y, f)^T$. Since the image plane lies at $z = f$, we can write the image coordinates as:

$$\mathbf{x} = \left(\frac{fX}{Z}, \frac{fY}{Z}\right)^T.$$

This defines a projection from $\mathbb{R}^3$ to $\mathbb{R}^2$. However, to express this mapping algebraically in a linear form and generalize it, it is convenient to move to projective geometry. As introduced in Section 2.1, we can use homogeneous coordinates, which allow us to describe this nonlinear mapping linearly. In homogeneous form, we define the projection matrix:

$$P = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix},$$

and represent a 3D point as $\tilde{\mathbf{X}} = (X, Y, Z, 1)^T$. The projection becomes:

$$\tilde{\mathbf{x}} = P\tilde{\mathbf{X}} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} fX \\ fY \\ Z \end{bmatrix} \sim \begin{bmatrix} \frac{fX}{Z} \\ \frac{fY}{Z} \\ 1 \end{bmatrix}.$$

This corresponds exactly to the desired projection in Euclidean coordinates. Therefore, the equation $\tilde{\mathbf{x}} = P\tilde{\mathbf{X}}$ defines a first simplified version of the pinhole model. In the following sections, we will introduce further generalizations leading to the complete standard pinhole model.

2.2.2 Standard pinhole model

A first generalization of the basic pinhole model involves accounting for the principal point offset. The principal point is defined as the intersection between the image plane and the line that passes through the projection center and is orthogonal to the image plane.

In the simplified model, this point is assumed to coincide with the origin of the image coordinate system. However, this is rarely the case in practice. Typically, the principal point lies slightly off-center, and this offset must be taken into account. Thus, the mapping becomes:

$$(X, Y, Z)^T \mapsto \left(\frac{fX}{Z} + p_x,\ \frac{fY}{Z} + p_y \right)^T,$$

where (p_x, p_y) are the coordinates of the principal point in the image reference frame. In homogeneous coordinates, the projection can be expressed as:

$$\tilde{\mathbf{x}} = K \begin{bmatrix} I & \mathbf{0} \end{bmatrix} \tilde{\mathbf{X}},$$

where $\tilde{\mathbf{X}} = (X, Y, Z, 1)^T$, and the calibration matrix $K \in \mathbb{R}^{3\times3}$ is:

$$K = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}.$$

The matrix K is called the camera calibration matrix, and it plays a central role in the definition of the standard pinhole model.

To fully define this model, we must account for the fact that in practice, 3D points are not expressed in the camera coordinate frame, but rather in an arbitrary reference system known as the world coordinate frame. This is the case, for instance, during camera calibration or visual odometry. In the latter, the left camera is usually chosen as the origin of the world frame. In order to project a 3D point onto the image plane, the point must first be transformed from the world frame to the camera frame. Let $\mathbf{X}_w \in \mathbb{R}^3$ be a 3D point in the world coordinate system, and let $\mathbf{X}_c$ be the same point expressed in the camera frame. Then:

$$\mathbf{X}_c = R(\mathbf{X}_w - \mathbf{C}),$$

where $R \in \mathbb{R}^{3\times3}$ is a rotation matrix representing the camera orientation, and $\mathbf{C} \in \mathbb{R}^3$ is the camera center in world coordinates. This transformation can be written in homogeneous coordinates as:

$$\tilde{\mathbf{X}}_c = \begin{bmatrix} R & -R\mathbf{C} \\ \mathbf{0}^T & 1 \end{bmatrix} \tilde{\mathbf{X}}_w,$$

where the transformation matrix is 4×4. This representation makes the transformation linear in homogeneous space. Combining the coordinate trans-

formation with the camera projection, we obtain:

$$\tilde{\mathbf{x}} = K \begin{bmatrix} R & -R\mathbf{C} \end{bmatrix} \tilde{\mathbf{X}}_w,$$

which defines the full standard pinhole projection model. The 3 parameters in K are called *intrinsic parameters*, as they are specific to the camera itself. The 6 parameters defining R and $\mathbf{C}$ are known as *extrinsic parameters*, since they describe the camera's position and orientation with respect to the world frame. Sometimes, it is convenient to avoid explicitly using the camera center $\mathbf{C}$, and instead use a translation vector $\mathbf{t}$. In this case, the relation becomes:

$$\mathbf{X}_c = R\mathbf{X}_w + \mathbf{t},$$

and therefore the full camera matrix can be written as:

$$P = K \begin{bmatrix} R & \mathbf{t} \end{bmatrix},$$

where $\mathbf{t} = -R\mathbf{C}$.

2.2.3 Camera model extensions

The ideal pinhole model assumes square pixels and a uniform scale across both image axes. However, real cameras, such as the Raspberry Pi Camera Module 3 used in this thesis, deviate from this ideal due to physical sensor characteristics. In particular, modern CCD cameras often exhibit anisotropic pixel scaling, requiring a more realistic camera model. To account for this, we adopt an extended pinhole model that incorporates the intrinsic calibration matrix:

$$K = \begin{bmatrix} \alpha_x & 0 & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \text{where} \quad \alpha_x = fm_x, \quad \alpha_y = fm_y.$$

Here, f is the focal length, m_x and m_y are the pixel densities along the horizontal and vertical axes, and (x_0, y_0) is the principal point in pixel coordinates. The parameter α_x (resp. α_y) represents the focal length in terms of horizontal (resp. vertical) pixels. This model captures the essential internal characteristics of the camera and is widely used in computer vision. Importantly, we assume a zero skew parameter, $s = 0$, which is standard for nearly all modern cameras. The full camera projection model used throughout this thesis is thus:

$$P = K[R \mid t],$$

where R is a rotation matrix, t is a translation vector, and K is the intrinsic matrix described above. This formulation balances physical realism and mathematical tractability, making it well suited for applications such as calibration, structure from motion, and visual odometry. More general projective camera models exist (e.g., allowing for skew or using arbitrary 3×4 matrices), but are not required for our setting and are therefore not considered further.

2.2.4 From the projective camera to the camera

It is worth noting that the projection matrix P encodes information about the camera's geometry, such as the projection center and the principal plane. From these considerations, the matrix P can be decomposed into the camera's intrinsic and extrinsic parameters. This procedure will be discussed in detail in the section dedicated to the decomposition of the projection matrix, and later applied in the implementation of the visual odometry system.

Camera center

The matrix P is of size 3×4 and has rank 3. Consequently, its null-space is a one-dimensional vector space. It can be shown that this null-space is generated by a unique vector C, which represents the camera center in homogeneous coordinates within the world reference frame. As a brief reminder, a single vector in homogeneous coordinates, defined up to scale, generates a one-dimensional vector space, since all scalar multiples represent the same point in projective

space. The camera center is the only point in 3D space for which the image is undefined, since:

$$PC = \mathbf{0}$$

Intuitively, this makes sense as it represents the point from which all projection rays emanate.

Column vectors

We denote by $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4$ the 3-dimensional column vectors of the matrix P. These vectors correspond to image points with particular geometric meaning. Specifically,

$$\mathbf{p}_1 = P\tilde{X}, \quad \text{where} \quad \tilde{X} = (1, 0, 0, 0)^T,$$

which represents the direction at infinity along the X-axis in the world coordinate system. The image $\mathbf{p}_1$ is thus the vanishing point corresponding to the X direction. Similarly, $\mathbf{p}_2$ and $\mathbf{p}_3$ correspond to the vanishing points for the Y and Z directions respectively. Finally,

$$\mathbf{p}_4 = P\tilde{X}, \quad \text{with} \quad \tilde{X} = (0, 0, 0, 1)^T,$$

which is the image of the origin of the world coordinate system.

Row vectors

In $\mathbb{P}^3$, a plane is defined as the locus of points X satisfying a linear equation:

$$\pi^T X = 0,$$

where $\pi \in \mathbb{R}^4$ is a nonzero vector representing the plane. Specifically, the plane $\pi = (a, b, c, d)^T$ represents the equation:

$$aX_1 + bX_2 + cX_3 + dX_4 = 0.$$

In the affine case, setting $d = 1$ reduces to the classical Cartesian equation of a plane. Therefore, a plane in $\mathbb{P}^3$ is defined by four homogeneous variables. The rows of the matrix P can be interpreted as plane vectors. We denote by

P_1^T, P_2^T, P_3^T the first, second, and third row of P, respectively. Consider the set of points X lying on the plane defined by P_1^T, i.e., satisfying:

$$P_1^T X = 0.$$

These points are mapped to image points of the form $(0, y, w)^T$. Since $PC = \mathbf{0}$, it follows that C also lies on P_1. Thus, the plane P_1 passes through the projection center C and intersects the x-axis of the image plane. An analogous argument holds for P_2 with respect to the y-axis. The plane P_3, on the other hand, contains points satisfying

$$P_3^T X = 0,$$

which are mapped to points of the form $(x, y, 0)^T$, i.e., points at infinity on the image plane. Hence, P_3 is the principal plane, passing through C and parallel to the image plane.

Note an important distinction: while P_1 and P_2 depend on the image reference frame (shifting the image origin also shifts these planes), P_3 does not, as it is intrinsically linked to the camera's geometry.

2.3 Epipolar geometry

This section introduces epipolar geometry, the geometric framework that governs the relationship between two images of the same 3D scene, acquired from different viewpoints. These viewpoints may correspond either to two physically distinct cameras (as in stereo systems), or to a single moving camera observing the scene at different time instants.

In this work, we focus on a stereo configuration, where the two images are captured simultaneously by a pair of spatially separated cameras.

Geometric constraints

Epipolar geometry plays a fundamental role in stereo vision, as it simplifies the search for point correspondences, that is, identifying points in two different

images that correspond to the same 3D point in space.

Consider two cameras, each with its own projection matrix. The line connecting their projection centers is called the baseline, and it intersects the two image planes at points known as the epipoles.

Any plane containing the baseline is referred to as an epipolar plane. Let M be a 3D point observed by both cameras, with projections m_1 and m_2 on their respective image planes. It can be shown that m_1, m_2, the projection centers, and the point M lie on the same epipolar plane π, defined by the baseline and the back-projection of m_1. Since m_2 lies on π, it must also lie on the intersection of π with the second image plane. This intersection forms a line l', known as the epipolar line corresponding to m_1.

Consequently, to find the corresponding point m_2, one needs to search only along l', rather than across the entire image. Symmetrically, the back-projection of m_2 defines an epipolar line l in the first image, which must contain the corresponding point m_1.

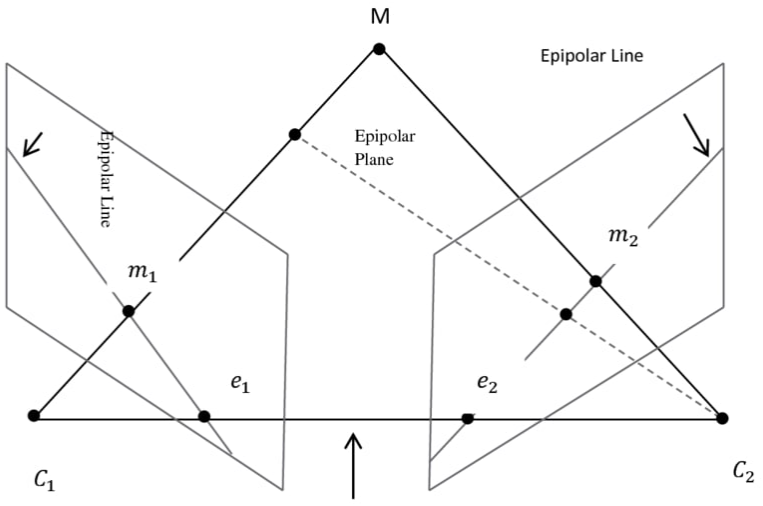


Figure 2.3: Epipolar plane. m1 and m2 are the images of M, e1 and e2 are the epipoles. Note that all epipolar lines intersect at the epipole

2.3.1 The fundamental matrix F

The fundamental matrix $\mathbf{F}$ is the algebraic representation of the epipolar geometry of a stereo system. As previously discussed, in the context of stereo matching, the search for a corresponding point in the second image is restricted to a line, the epipolar line. Algebraically, this defines a mapping:

$$\mathbf{x} \mapsto \ell'$$

where $\mathbf{x} \in \mathbb{P}^2$ is a point in the first image, and $\ell' \in \mathbb{P}^2$ is the corresponding epipolar line in the second image. This type of mapping, from points to lines, is known as a correlation in projective geometry and is represented by a 3×3 rank-2 matrix called the fundamental matrix $\mathbf{F}$. More specifically, for any point $\mathbf{x}$ in the first image, the corresponding epipolar line in the second image is:

$$\ell' = \mathbf{F}\,\mathbf{x}$$

Dually, for a point $\mathbf{x}'$ in the second image, the epipolar line in the first image is $\ell = \mathbf{F}^\top \mathbf{x}'$. The fundamental matrix encapsulates all the epipolar geometry between the two views and can be estimated directly from a set of image correspondences $(\mathbf{x}, \mathbf{x}')$.

Geometric derivation

We want to map a point $\mathbf{x}$ in the first image to the corresponding epipolar line $\mathbf{l}'$ in the second image. The process is divided into two steps:

Step 1: Transfer via a plane.

We select a plane π in 3D space that does not intersect any of the two camera centers. Conceptually, this plane contains some 3D points (though not all) that we want to project. Consider the ray passing through the camera center and the image point $\mathbf{x}$: it intersects π in a 3D point $\mathbf{X}$. Projecting $\mathbf{X}$ onto the second image yields a point $\mathbf{x}'$. By epipolar geometry, since $\mathbf{X}$ lies on the back-projection of $\mathbf{x}$, then $\mathbf{x}'$ lies on the epipolar line $\mathbf{l}'$. This reasoning holds for all $\mathbf{x}_i$ and corresponding $\mathbf{x}'_i$, implying the existence of a 2D homography

H_π that maps points from one image to the other:

$$\mathbf{x}' = H_\pi \mathbf{x}$$

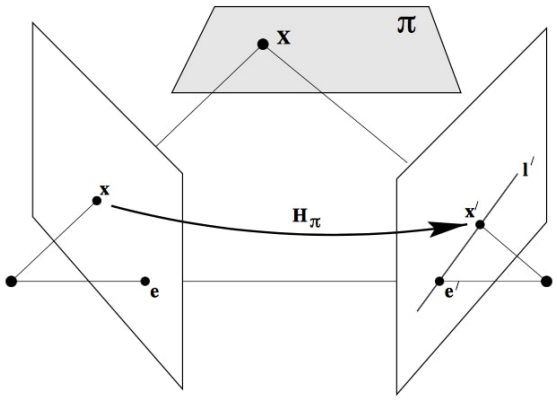


Figure 2.4: Transfer via a plane

Step 2: Line from two points.

Given $\mathbf{x}'$ and the epipole $\mathbf{e}'$ in the second image, the epipolar line $\mathbf{l}'$ is the line through these two points:

$$\mathbf{l}' = \mathbf{e}' \times \mathbf{x}' = \mathbf{e}' \times H_\pi \mathbf{x}$$

This can be rewritten as:

$$\mathbf{l}' = [\mathbf{e}']_\times H_\pi \mathbf{x}$$

where $[\mathbf{e}']_\times$ is the skew-symmetric matrix of the vector $\mathbf{e}' \in \mathbb{R}^3$, defined as:

$$[\mathbf{e}']_\times = \begin{bmatrix} 0 & -e'_3 & e'_2 \\ e'_3 & 0 & -e'_1 \\ -e'_2 & e'_1 & 0 \end{bmatrix}$$

This identity holds:

$$\mathbf{e}' \times \mathbf{v} = [\mathbf{e}']_\times \mathbf{v}, \quad \forall \mathbf{v} \in \mathbb{R}^3$$

which transforms a vector cross product into a matrix multiplication — easier to implement and manipulate. We define the fundamental matrix as:

$$\mathbf{F} = [\mathbf{e}']_\times H_\pi$$

so that:

$$\mathbf{l}' = \mathbf{F}\mathbf{x}$$

Geometrically, $\mathbf{F}$ maps points in $\mathbb{P}^2$ to lines in $\mathbb{P}^2$, and has rank 2. This matrix plays a central role in stereo reconstruction and motion estimation, as it encodes the epipolar constraints, discussed in the next paragraph.

2.3.2 Properties of F

Several key properties of the fundamental matrix F help us better understand its role in visual odometry.

Epipolar Constraint

The fundamental matrix satisfies the following constraint for any pair of corresponding points $\mathbf{x} \leftrightarrow \mathbf{x}'$ in two images:

$$\mathbf{x}'^\top F \mathbf{x} = 0$$

This follows directly from the relation $\mathbf{l}' = F\mathbf{x}$. Since $\mathbf{x}'$ lies on its corresponding epipolar line $\mathbf{l}'$, the dot product must vanish:

$$\mathbf{x}'^\top \mathbf{l}' = 0 \Rightarrow \mathbf{x}'^\top F \mathbf{x} = 0$$

Although simple, this constraint is of crucial importance: it allows the computation of F using only point correspondences between two images, without requiring the explicit camera projection matrices. Therefore, estimating F constitutes the first geometric step in the motion estimation stage of the visual odometry pipeline. It is typically estimated using robust algorithms like RANSAC.

Transpose Property

If F is the fundamental matrix for the camera pair (P, P'), then $F^\top$ is the fundamental matrix for the reversed pair (P', P).

Epipoles

Each epipolar line in an image passes through the corresponding epipole. Given $\mathbf{l}' = F\mathbf{x}$, and since all lines must pass through the epipole $\mathbf{e}'$, we have:

$$\mathbf{e}'^\top \mathbf{l}' = \mathbf{e}'^\top F\mathbf{x} = 0 \quad \forall \mathbf{x}$$

This implies:

$$\mathbf{e}'^\top F = \mathbf{0}^\top \quad \text{and} \quad F\mathbf{e} = \mathbf{0}$$

That is, $\mathbf{e}'$ lies in the left null-space (kernel) of F, and $\mathbf{e}$ lies in the right null-space. This constraint highlights a key geometric property: although F maps a 2D point to a line (which has two degrees of freedom), all of these lines must pass through a common point — the epipole. For this to hold, F must have rank 2, and therefore:

$$\det(F) = 0$$

Being a homogeneous 3×3 matrix with rank 2, F has 7 degrees of freedom. This fact also explains why at least 7 point correspondences are required to estimate F uniquely.

2.3.3 The Essential Matrix E

Up to this point, using the fundamental matrix F, we have remained in the "camera space": points were expressed in pixels and were tied to the internal parameters of the camera — focal length, skew, and the principal point offset. However, our ultimate goal is to estimate the relative rotation and translation between the two cameras. To do so, we must decouple from the pixel-based representation. As discussed in the pinhole model, the matrix K maps a 3D point (actually a 3D ray) to an image point using the camera's internal parameters. We now aim to perform the inverse operation. Given $x = PX$, if

K is known, we can compute:

$$\hat{x} = K^{-1}x$$

where $\hat{x}$ is the image point x expressed in normalized coordinates.

Example: Suppose $x = \begin{bmatrix} 800 \\ 600 \\ 1 \end{bmatrix}$ (a pixel near the top-right region of a 1280x960 image), and assume the intrinsic matrix:

$$K = \begin{bmatrix} 800 & 0 & 640 \\ 0 & 800 & 480 \\ 0 & 0 & 1 \end{bmatrix}$$

Then the normalized coordinates $\hat{x}$ are obtained as:

$$\hat{x} = K^{-1}x = \begin{bmatrix} 1/800 & 0 & -640/800 \\ 0 & 1/800 & -480/800 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 800 \\ 600 \\ 1 \end{bmatrix} = \begin{bmatrix} (800-640)/800 \\ (600-480)/800 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.2 \\ 0.15 \\ 1 \end{bmatrix}$$

This result means that the direction of the ray starting at the camera center and passing through pixel $(800, 600)$ can be described by the vector $\begin{bmatrix} 0.2 \\ 0.15 \\ 1 \end{bmatrix}$ in normalized camera coordinates. These values are expressed in units of focal length, meaning that the x and y components represent the offset from the principal point, normalized by the focal length (here assumed to be 1). The resulting vector defines a viewing ray originating from the optical center of the camera and pointing in the direction of the corresponding image point. It does not represent a specific 3D point in the scene, but rather the entire set of 3D points lying along that ray. For this reason we do not know the absolute translation.

Though lens distortion is not encoded in K, it is assumed to be corrected be-

forehand via appropriate distortion coefficients. We are now effectively working with an ideal pinhole camera. Thus, the camera matrix becomes:

$$K^{-1}P = [R \mid t]$$

This is called the normalized camera matrix. The fundamental matrix that describes the epipolar geometry between two normalized camera matrices is called the essential matrix, denoted by E. It can be considered a calibrated version of F. Just like F, the essential matrix satisfies the epipolar constraint. Given corresponding points $\hat{x}$ and $\hat{x}'$ in normalized coordinates:

$$\hat{x}'^T E \hat{x} = 0$$

Since $\hat{x}'^T = x'^T K'^{-T}$ and $\hat{x} = K^{-1}x$, we can write:

$$x'^T K'^{-T} E K^{-1} x = 0$$

Given that $x'^T F x = 0$, we obtain the relation:

$$F = K'^{-T} E K^{-1} \quad \Rightarrow \quad E = K'^T F K$$

This equation establishes the direct link between F and E.

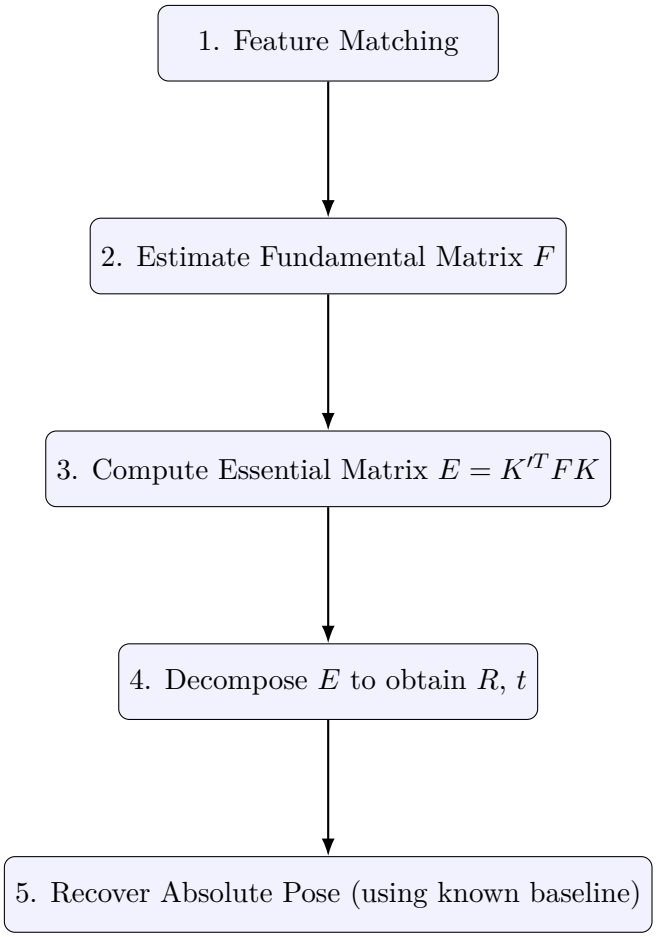


Figure 2.5: Pipeline of motion estimation using the essential matrix

2.3.4 Practical considerations

Although the matrices **F** and **E** are not used in the implemented system, where depth is computed directly via triangulation from a calibrated stereo pair, their inclusion is valuable for understanding the geometric foundations of visual odometry.

This knowledge may prove useful in future developments, especially in challenging scenarios like underwater robotics, where accurate calibration is often impractical. In such cases, estimating **F** or **E** from image correspondences can help improve robustness and enable relative pose estimation without prior calibration.

Chapter 3

Setup and preprocessing

This chapter covers the initial stages of the visual odometry pipeline, including the configuration of a fixed stereo camera system and image preprocessing. Specifically, we cover the setup of the system, camera calibration, and stereo rectification.

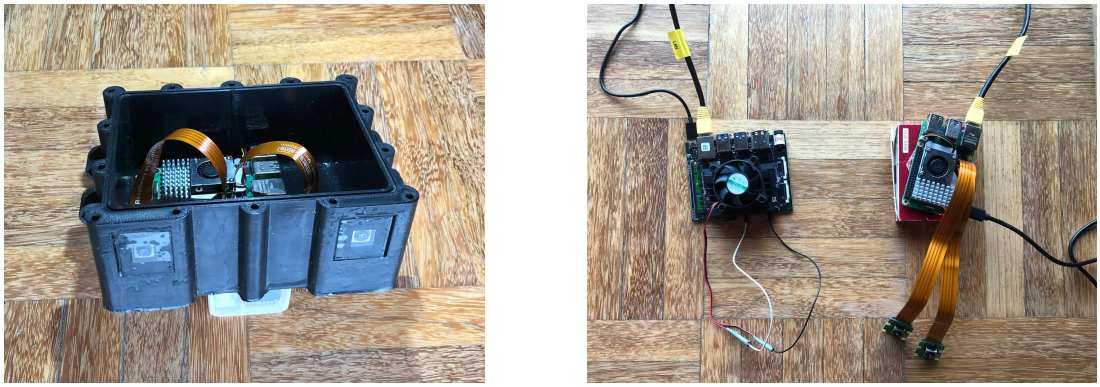
In addition to selected sequences from the KITTI dataset, introduced in Chapter 1 and used as a professional benchmark, we also employ a custom Lab Dataset acquired using our stereo rig in a controlled indoor environment. The rig was manually moved along a table to minimize vertical offset between the cameras while capturing scenes that present visual challenges not typically found in KITTI, such as textureless areas and limited features. This second dataset is particularly valuable as it requires full calibration and rectification, providing a realistic testbed for evaluating the system's robustness under non-ideal and less structured conditions. Accordingly, Chapters 3 and 4 will present results obtained on both datasets to assess the performance and limitations of the proposed pipeline.

3.1 Hardware setup

The following section describes the hardware setup used for acquire the Lab dataset. Each stereo module includes the following components:

- One Raspberry Pi 5

- Two Raspberry Pi Camera Module 3
- Dedicated camera connectors
- Power bank for mobile power supply
- microSD card for on-board data storage
- Cooling fan mounted on the Raspberry Pi 5
- Custom 3D-printed enclosure containing all components



(a) External view of the enclosure (b) components

Figure 3.1: Stereo module with 3D-printed housing

The baseline between the two cameras is approximately 12 cm, with a more accurate value of 12.22 cm estimated through calibration. This value represents a balanced trade-off: it is sufficiently wide to achieve accurate depth estimation even at medium-to-long distances, while still compact enough to keep the system portable and mechanically stable. A wider baseline improves depth resolution for distant objects, which is particularly beneficial for visual odometry applications where the camera needs to track motion over several meters.

3.2 Calibration

In this section, we explore the concept of camera calibration, a fundamental step in the VO pipeline implemented in this work. All subsequent stages of

the pipeline depend heavily on the intrinsic parameters of the stereo camera system.

The calibration process allows us to introduce a metric structure into the image plane, enabling the measurement of angles and ratios of lengths. Although this does not provide an absolute metric reconstruction, it does yield a reconstruction that is metric up to an unknown scale factor.

We begin by discussing the theoretical foundations of camera calibration, with a particular emphasis on the role of the absolute conic in transitioning from the projective to the metric space. Following this, we provide a briefly description of the algorithms used for both monocular and stereo calibration, focusing in particular on the methods employed in this thesis using the OpenCV library.

3.2.1 The absolute conic

The absolute conic, denoted as Ω_∞, is a purely theoretical object that lies entirely on the plane at infinity π_∞. In 3D projective space, the points belonging to Ω_∞ satisfy the following two constraints:

$$X_1^2 + X_2^2 + X_3^2 = 0, \quad X_4 = 0$$

This formulation shows that the conic is made up only of points at infinity, and geometrically encodes the notion of circularity, that is, it characterizes which transformations preserve angles and shapes such as circles in the metric space. We can express Ω_∞ in matrix form, as a conic defined by a symmetric matrix C such that a point $X \in \mathbb{P}^3$ lies on the conic if and only if:

$$X^\top C X = 0$$

In the canonical metric frame, the absolute conic can thus be described by:

$$(X_1, X_2, X_3) I (X_1, X_2, X_3)^\top = 0$$

This leads us to associate Ω_∞ with the identity matrix $C = I$. A key geometric result is that Ω_∞ is invariant under a projective transformation if and only if

that transformation is a similarity, meaning it preserves angles and length ratios (up to scale). A similarity transformation in $\mathbb{R}^3$ has the form:

$$x = sRX + t$$

where R is a rotation matrix, $s > 0$ is a global scale, and t is a translation vector.

Proof: Let us assume a general projective transformation H acting on homogeneous coordinates $X \in \mathbb{P}^3$. If we require that the plane at infinity π_∞ remains fixed (i.e., mapped onto itself), the transformation must be affine. An affine transformation can be represented in homogeneous coordinates by a 4×4 matrix of the form:

$$H = \begin{bmatrix} A & t \\ 0 & 1 \end{bmatrix}$$

where A is a 3×3 matrix and t is a 3×1 translation vector. The fact that the last row is $(0, 0, 0, 1)$ ensures that points at infinity (with $X_4 = 0$) remain on the plane at infinity. Let us now consider how such a transformation affects the absolute conic. Under a transformation H, a point X becomes HX, and the transformed conic Ω'_∞ is given by:

$$\Omega'_\infty = H^{-\top} C H^{-1}$$

To preserve the conic (i.e., $\Omega'_\infty = C = I$), the following condition must hold:

$$H^{-\top} I H^{-1} = \lambda I$$

Substituting the affine form of H, we obtain that this is equivalent to requiring:

$$A^\top A = \lambda I$$

This condition implies that A must be a scaled rotation matrix, i.e. a similarity transformation. Therefore, we conclude that only similarity transformations preserve the absolute conic.

Summary of Transformation Hierarchy

To better understand the role of the absolute conic, we summarize the hierarchy of geometric transformations in 3D:

Transformation	DOF	Preserves	Form
Projective	15	Incidence relations	$x' = HX$
Affine	12	Parallelism	$x' = AX + t$
Similarity	7	Angles, length ratios (up to scale)	$x' = sRX + t$
Euclidean	6	Angles and lengths	$x' = RX + t$

Table 3.1: Geometric transformation classes in 3D space.

Euclidean transformations correspond to rigid body motions, which are the ultimate goal in visual odometry, as they describe how the camera moves through space. However, in the context of single camera calibration, having a metric reconstruction up to scale is generally sufficient for most computer vision applications.

By introducing the absolute conic, we are enforcing 5 geometric constraints on the reconstruction, which correspond to the following degrees of freedom:

- 2 DOF: enforcing orthogonality between the coordinate axes (angles),
- 2 DOF: enforcing equal scale along the axes (isotropy),
- 1 DOF: fixing the overall scale factor (up to an unknown constant).

In this way, the absolute conic introduces the necessary structure to elevate a purely projective reconstruction to a metric one (up to scale).

3.2.2 Better understanding of K

In Chapter 2, we discussed the importance of the intrinsic matrix K in the projection of the 3D world onto the image plane. However, it is also possible to project points using $K = \mathbf{I}_{3\times3}$, i.e., without knowing the intrinsic parameters that define K. What changes between these two cases? To answer this, we delve deeper into the concept of K. As introduced in Chapter 2, a point $\mathbf{x}$ on the image plane backprojects to a ray defined by the image point $\mathbf{x}$ and the

camera center. Suppose the points on this ray are expressed as $\tilde{\mathbf{X}} = \lambda\mathbf{d}$ in the Euclidean coordinate system of the camera. These points map onto the image plane as:

$$\mathbf{x} = K[\mathbf{I}|\mathbf{0}] \begin{pmatrix} \lambda\mathbf{d} \\ 1 \end{pmatrix} = \lambda K\mathbf{d} \quad \text{(up to scale)}.$$

Thus, K identifies the affine transformation between the image point $\mathbf{x}$ and the direction of the backprojected ray $\mathbf{d} = K^{-1}\mathbf{x}$, measured in the Euclidean coordinate system of the camera (i.e., in metric units rather than pixels).
This means that K essentially establishes the metric within the image plane: it allows us to recover meaningful geometric quantities such as angles between rays. For instance, given two image points $\mathbf{x}_1$ and $\mathbf{x}_2$, the angle between their corresponding rays $\mathbf{d}_1 = K^{-1}\mathbf{x}_1$ and $\mathbf{d}_2 = K^{-1}\mathbf{x}_2$ can be computed via:

$$\cos\theta = \frac{\mathbf{d}_1^\top \mathbf{d}_2}{\|\mathbf{d}_1\|\|\mathbf{d}_2\|}.$$

Without knowledge of K (i.e., assuming $K = \mathbf{I}$), the image plane coordinates lack this intrinsic metric information, and thus angles and length ratios cannot be correctly interpreted in a physical sense.
Furthermore, this metric role of K connects closely with the concept of the Image of the Absolute Conic (IAC), introduced in the next section. The IAC encodes the intrinsic calibration parameters in a projective invariant form and is a fundamental tool to recover K from images. Hence, the intrinsic matrix K and the absolute conic are deeply related: K can be derived by enforcing the preservation of the conic under suitable transformations

3.2.3 The Image of the Absolute Conic

We now aim to project the Absolute Conic, defined earlier in the 3D projective space, onto the image plane. To do so, consider how a point $\mathbf{X}_\infty = (\mathbf{d}^\top, 0)^\top$ lying on the plane at infinity π_∞ is mapped onto the image plane by a generic

camera matrix P. In general, the projection is given by:

$$\mathbf{x} = P\mathbf{X}_\infty = KR[\mathbf{I} \mid -\mathbf{C}] \begin{pmatrix} \mathbf{d} \\ 0 \end{pmatrix} = KR\mathbf{d}.$$

The mapping from π_∞ to the image plane is therefore a homography $H = KR$, which depends only on the intrinsic matrix K and the rotation matrix R. Importantly, this mapping is independent of the camera center C. Applying this homography H to the Absolute Conic Ω_∞, we obtain its image on the plane, the so-called Image of the Absolute Conic (IAC), denoted by ω:

$$\omega = (KR) \cdot \Omega_\infty \cdot (KR)^\top.$$

Since Ω_∞ is mapped to the identity matrix in canonical coordinates (as seen in the previous section), and R is an orthonormal matrix such that $RR^\top = \mathbf{I}$, the IAC simplifies to:

$$\omega = K^{-\top} K^{-1}.$$

This formulation highlights a crucial fact: ω depends exclusively on the intrinsic matrix K and is completely independent of the camera's orientation and position in space. Therefore, if ω can be estimated from image data, it becomes possible to recover K, up to scale, by performing a Cholesky or inverse decomposition.

The IAC thus provides a gateway to introducing metric properties into the image: once ω is known, one can interpret angles and distances (up to scale) directly on the image plane. This makes the IAC a key concept in several self-calibration and autocalibration methods, where camera intrinsics are recovered from multiple views without using a known calibration target.

Although the IAC is not directly employed in the calibration algorithm implemented in this thesis, it is introduced for the same reason as the Fundamental and Essential matrices to provide a solid theoretical foundation for understanding the geometry underlying visual odometry. In particular, in challenging contexts such as underwater environments, where prior calibration is difficult,

the IAC plays a crucial role in autocalibration, enabling the recovery of intrinsic parameters from image data alone. The calibration approach adopted here is based on a known planar calibration pattern (i.e., a checkerboard), allowing the direct estimation of K and distortion parameters.

3.2.4 Lens distorsion

Real-world cameras require lenses to focus incoming light rays onto the optical sensor, thereby enabling the formation of sharp images. However, the presence of lenses introduces various distortions, due to the physical refraction of light. The most significant types of distortion are:

- **Radial distortion**: This is caused by the shape of the lens itself. Straight lines, especially those farther from the image center, appear curved either outward (barrel distortion) or inward (pincushion distortion).
- **Tangential distortion**: This occurs when the lens is not perfectly aligned perpendicular to the image sensor plane. As a result, certain parts of the image may appear closer or farther than they actually are.

These distortions are commonly modeled using a set of intrinsic distortion parameters. Radial distortion is typically expressed as a polynomial function of the distance from the optical center. Let r be the radial distance from the principal point; the corrected coordinates $(x_{\text{corr}}, y_{\text{corr}})$ can be expressed as:

$$x_{\text{corr}} = x\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right), \quad y_{\text{corr}} = y\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right)$$

where k_1, k_2, k_3 are the radial distortion coefficients.
Tangential distortion is modeled as:

$$x_{\text{corr}} = x + \left[2p_1 xy + p_2(r^2 + 2x^2)\right], \quad y_{\text{corr}} = y + \left[p_1(r^2 + 2y^2) + 2p_2 xy\right]$$

where p_1 and p_2 are the tangential distortion coefficients.
These parameters (k1, k2, k3, p1, p2) are estimated during the camera cal-

ibration process and are used later to undistort the images before further processing.

3.2.5 Calibration Algorithms

In this section, we provide a high-level overview of the monocular and stereo camera calibration algorithms employed in this work. The calibration procedure relies on a known planar pattern, specifically a checkerboard with fixed square size and known geometry.

The method used for monocular calibration is based on the technique introduced by Zhang [5], which allows for calibration using multiple images of a planar target taken from different, unknown orientations. This approach estimates both the intrinsic parameters K and the distortion coefficients of each camera by minimizing the reprojection error of the known 3D points (checkerboard corners) onto the image plane.

Stereo calibration builds upon the individual monocular calibration of both cameras and additionally estimates the relative pose (rotation and translation) between them. The implementation used in this thesis leverages the functions provided by the OpenCV library, in particular:

- `cv2::findChessboardCorners` for detecting the corner points of the checkerboard;
- `cv2::calibrateCamera` for monocular calibration;
- `cv2::stereoCalibrate` for joint estimation of stereo parameters.

For more details, refer to the OpenCV documentation[1].

Monocular calibration

The first step in the monocular calibration process is the definition of the checkerboard pattern used for calibration. In our case, the pattern consists of 13×9 inner corners, with each square having a side length of 0.04 meters. The checkerboard is black and white, a standard choice due to the high contrast,

[1] `https://docs.opencv.org/4.x/`

which facilitates corner detection.

Given these characteristics and assuming the checkerboard lies on a planar surface (with $z = 0$ for simplicity), we can generate the corresponding 3D coordinates of the inner corners in metric units. For example:

$$\mathbf{X}_{\text{world}} = \{(0, 0, 0), (0.04, 0, 0), (0.08, 0, 0), \ldots\}$$

The core idea is to compare these known 3D coordinates with the corresponding 2D image points detected in the camera frames, in order to estimate the intrinsic parameters K, distortion coefficients, and, for each image, the extrinsic parameters R and T. To detect corners in the images, the OpenCV function `cv2::findChessboardCorners` is used. This function returns the pixel coordinates of the detected corners. It is important to note that depending on the viewing angle, corner detection accuracy may vary, oblique poses often yield poorer initial localization.

To refine this localization, especially for applications such as visual odometry where precision is critical, we apply `cv2::cornerSubPix`. This function refines the detected pixel coordinates to subpixel accuracy by searching for the actual corner location (e.g., the intersection of two black squares) within each pixel neighborhood.

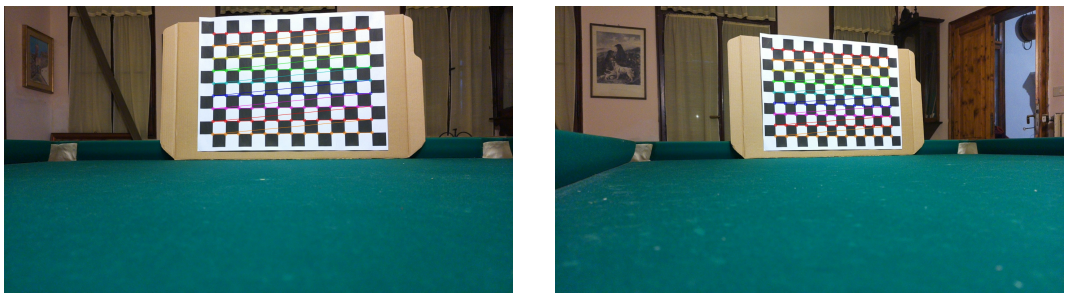


Figure 3.2: Chessboard pattern images

After collecting 2D–3D correspondences from all calibration images, the function `cv2::calibrateCamera` is called. This function performs a nonlinear optimization to jointly estimate the following:

- Intrinsic matrix K

- Distortion coefficients
- Rotation and translation vectors (R_i, T_i) for each image

The function takes as input:

- The 3D object points for each image
- The corresponding 2D image points
- Image size (width and height)

It returns the estimated parameters and the reprojection error, which quantifies the discrepancy between observed image points and those reprojected from the estimated 3D geometry. Since this metric reflects the overall quality of the calibration process, it will be analyzed in detail in the following paragraph.

The reprojection error

A key quantity in camera calibration is the reprojection error, which provides a numerical measure of how well the estimated camera parameters explain the observed 2D image points.
Formally, given a 3D point X_i on the calibration target, and its observed 2D projection x_i, the reprojection error e_i for that point is defined as:

$$e_i = \|x_i - \hat{x}_i\|_2$$

where $\hat{x}_i$ is the reprojected point computed by projecting X_i using the estimated intrinsic and extrinsic parameters:

$$\hat{x}_i = \pi(K[R|t]X_i)$$

Here, π denotes the pinhole projection function (including distortion), K is the intrinsic matrix, and $[R|t]$ are the extrinsics for the image under consideration. The mean reprojection error is commonly used as a global measure of calibration accuracy:

$$\bar{e} = \frac{1}{N}\sum_{i=1}^{N} e_i$$

where N is the total number of detected points across all calibration images.

Il valore di ritorno di `cv2::calibratecamera`, tuttavia, è il RMS (root mean square). This metric expresses, in pixels, the average distance between the detected points and the ideal projections according to the estimated camera model. Lower values indicate a better fit; typical values for good calibration are usually below 1 pixels.

Stereo calibration

Once the intrinsic parameters of each camera have been estimated, it is crucial in a stereo setup to determine the relative pose between the two cameras. Specifically, we aim to recover the Euclidean transformation, represented by a rotation matrix R and a translation vector t, that allows us to express 3D points from the left camera reference frame (chosen as the origin by convention) in the right camera's frame.

This step is fundamental, as it enables the computation of the baseline between the cameras and allows us to transition from a projective model with metric defined up to scale to an absolute metric reconstruction.

For this task, the `cv2.stereoCalibrate` function from OpenCV was employed. As with `cv2.calibrateCamera`, this function solves a nonlinear optimization problem by minimizing the overall reprojection error across both cameras and all chessboard views. The objective function is defined as:

$$E_{\text{total}} = \sum_i \sum_j \left(\left\| \mathbf{x}_{ij}^{(L)} - \hat{\mathbf{x}}_{ij}^{(L)}(K_1, D_1, R_i, t_i) \right\|^2 + \left\| \mathbf{x}_{ij}^{(R)} - \hat{\mathbf{x}}_{ij}^{(R)}(K_2, D_2, R_i R, t_i + T) \right\|^2 \right) \tag{3.1}$$

Here:

- $\mathbf{x}_{ij}^{(L)}$ and $\mathbf{x}_{ij}^{(R)}$ are the observed image points for the j-th corner in the i-th image pair.
- $\hat{\mathbf{x}}_{ij}^{(L)}$ and $\hat{\mathbf{x}}_{ij}^{(R)}$ are the projected 3D points using the estimated parameters.
- K_1, K_2, D_1, and D_2 are the intrinsics and distortion coefficients for the left and right cameras, respectively.

- R_i and t_i are the pose (rotation and translation) of the calibration target in the left camera for the i-th image.

- R and T are the global rotation and translation between the left and right camera frames—i.e., the stereo rig's relative pose.

It is important to note that only the poses (R_i, t_i) of the left camera are estimated per view. The corresponding poses of the right camera are derived as $(R_i R, t_i + T)$ using the relative transformation estimated through the algorithm.

The function also returns the fundamental matrix F and essential matrix E, which encode the epipolar geometry of the stereo setup. These matrices are internally used by the algorithm to estimate the relative pose R and T between the cameras. Although they will not be used explicitly in the subsequent parts of this thesis, having access to them can still be useful for diagnostic purposes and consistency checks during calibration.

Finally, `cv2::stereoCalibrate` returns an RMS reprojection error across all views. As specified in the OpenCV documentation, this is provided as a vector containing the per-view RMS values. These serve as indicators of calibration accuracy. The parameters retained for the subsequent stages of the stereo VO pipeline are:

$$K_1, K_2, D_1, D_2, R, T$$

Results.

In our setup, using the chessboard described previously and a total of 30 image pairs, the stereo calibration procedure gives a reprojection error of 0.251 px. While a low reprojection error, well below the 1 pixel threshold, is generally indicative of a good calibration, it does not by itself guarantee the generalization of the estimated parameters. The calibration process might still fail to capture the full geometry of the system if the input data lacks sufficient variability. For this reason, it is crucial to collect calibration images from a wide range of viewpoints to ensure that the computed parameters generalize well to arbitrary scenes.

3.3 Rectification

The goal of stereo rectification is to transform the images acquired from two cameras so that corresponding points are projected onto the same horizontal line. This alignment of epipolar lines greatly simplifies the correspondence search, reducing it to a one-dimensional (horizontal) problem. In particular, it facilitates the computation of disparity, defined as the horizontal shift between corresponding points in the rectified stereo images:

$$d = x'_1 - x'_2$$

Disparity is a fundamental quantity in stereo vision, as it enables depth inference by triangulation. A more rigorous definition and usage of disparity will be presented in Chapter 4, where it serves as the starting point for 3D reconstruction.

To achieve rectification, two planar homographies H_1 and H_2 are computed for the left and right images, respectively. These are defined as:

$$H_i = K' R_i K_i^{-1}, \quad i = 1, 2$$

where K_i is the intrinsic matrix of camera i, R_i is the rotation matrix that aligns the image to the rectified reference frame, and K' is the desired intrinsic matrix for the rectified images.

The geometric intuition behind this transformation is as follows: the matrix K_i^{-1} maps a pixel coordinate back into the normalized camera coordinate system (i.e., its internal metric space), R_i rotates the point to align the image plane with the new rectified reference frame, and K' reprojects the point onto the new virtual image plane. The result of applying H_i to the original image points is that the visual rays of both cameras become parallel, and corresponding points lie along the same scanline.

In practice, it is common to choose the left camera as the reference for the rectified frame to minimize distortion applied to the first image. In this case,

we set $R_1 = I$, meaning no rotation is applied to the left camera, and R_2 is computed as the rotation needed to align the right camera to the rectified frame defined by the left.

The function `cv2::stereoRectify` is used to compute the rectification transforms. Following that, `cv2::initUndistortRectifyMap` is applied to jointly handle undistortion and rectification. This function computes the transformation and represents it as a remapping grid. The remapping process consists of taking each pixel from the original image and assigning it a new location in the rectified image. Since this mapping does not always correspond to integer pixel positions, interpolation must be used to estimate the values at non-integer coordinates. While the mathematical and algorithmic details of this process are beyond the scope of this thesis, it is worth noting that this step is crucial to obtaining distortion-free and rectified images.

Finally, the `cv2::remap` function is used to apply the computed map and produce the final rectified images. An example showing images before and after rectification is shown below.

Figure 3.3: Epipolar lines on a KITTI dataset frame.

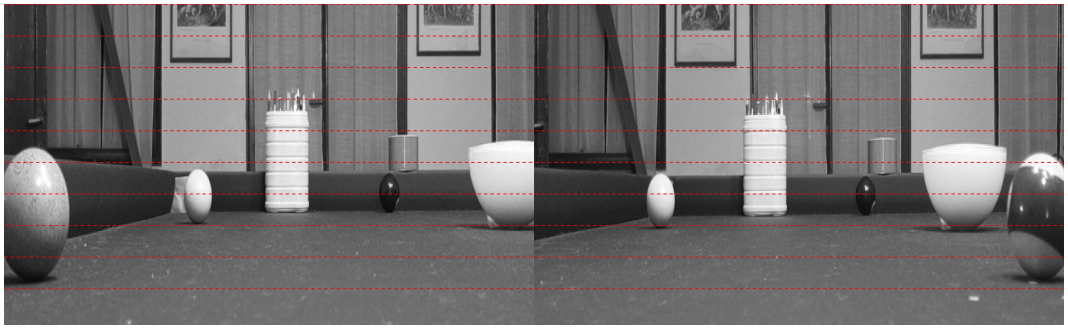


Figure 3.4: Epipolar lines on a not rectified Lab dataset frame.

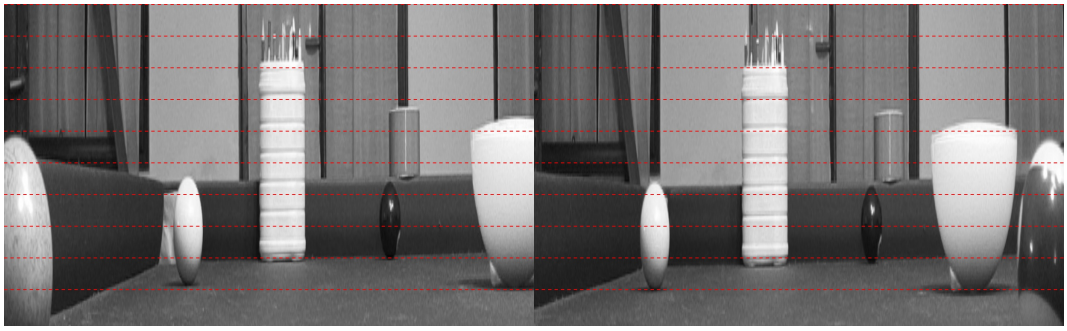


Figure 3.5: Epipolar lines on a rectified Lab dataset frame.

From the results, it is immediately evident that the frames from the KITTI dataset are perfectly rectified. The same objects in the left and right views differ only along the x-axis, without vertical disparity. Conversely, in the Lab dataset, a small amount of vertical disparity is still present in the unrectified images, and the rectification does not seem to completely eliminate this imperfection. In some cases, the rectified image appears even degraded compared to the original.

Nevertheless, the disparity map computed using the rectified images from the Lab dataset turns out to be more detailed and consistent. This suggests that, although not perfect, the estimated calibration parameters were sufficiently accurate. Indeed, the rectification does not heavily distort the images, which aligns with the relatively low reprojection error obtained during calibration (0.251 px). However, this highlights that reprojection error alone may not capture all sources of calibration inaccuracies.

Potential causes for these imperfections include the use of a printed checkerboard mounted on cardboard, which may introduce warping or reflectivity artifacts. Moreover, the mechanical setup, which constrained the stereo box to slide on a fixed plane, might have limited the diversity of acquisition angles, reducing the algorithm's ability to estimate the full camera model (generalize).

Chapter 4

VO pipeline

This chapter presents the operational pipeline of the visual odometry system, that is the sequence of processing steps which, starting from distortion-free, rectified stereo image pairs acquired from calibrated cameras, allows estimating the trajectory traveled by the system.
The spatial reference frame adopted for motion estimation is the left camera, which is the standard reference in most stereo vision systems.

4.1 The disparity map

Given a pair of stereo images, the disparity is defined as the displacement of the same 3D point projected onto the two image planes. After rectification, the disparity becomes a horizontal shift along the epipolar lines (ideally), which simplifies the correspondence search. The basic formula is:

$$d = x_{\text{left}} - x_{\text{right}}$$

A disparity map is a matrix in which each element represents the horizontal displacement of a pixel in the left image with respect to its corresponding pixel in the right image. This disparity is essential for estimating the depth of the scene.
Since the left camera is taken as the reference frame (a standard convention in stereo setups), the disparity is used to compute the depth map relative to

that viewpoint.
In this implementation, disparity is computed using OpenCV's stereo matching algorithms. Two options were considered and integrated into the system:

- **StereoBM** (*Block Matching*): a simpler and computationally efficient algorithm based on sliding window matching. It is fast and suitable for real-time applications. However, it may fail in low-texture regions or produce noisy disparity maps.

- **StereoSGBM** (*Semi-Global Block Matching*): a more advanced algorithm that considers multiple path directions for matching cost aggregation. It significantly improves disparity accuracy, especially in challenging areas, but is more computationally expensive.

In various scenes, `StereoBM` provided insufficient matching quality, which motivated the use of `StereoSGBM` as an alternative for better performance. The system was explicitly designed to support both, enabling testing and trade-offs between speed and accuracy. Deep learning-based methods were excluded due to their high computational demands and complexity, which are not ideal for future real-time deployment.
A comparative performance evaluation between the two methods will be presented in the next sections.

4.1.1 Technical Overview of StereoBM

To better understand how the `StereoBM` algorithm works, we can break it down into three main stages:

Preprocessing

First, the stereo images are converted to grayscale if necessary, since block matching operates on intensity rather than color channels. A horizontal gradient filter is typically applied to emphasize vertical edges, which are crucial for matching along epipolar lines. A common choice is the Sobel filter, which

approximates the horizontal derivative:

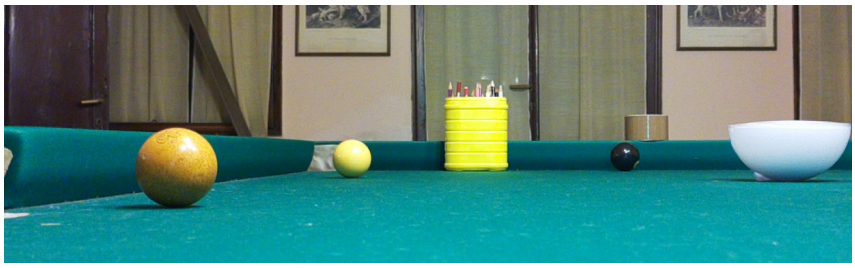
$$G_x = \frac{\partial I}{\partial x}$$

This can be implemented via convolution with the Sobel kernel:

$$K_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Note the central column of zeros: this ensures that only horizontal intensity changes are captured. For vertical gradients, one would simply use the transposed kernel.

This filtering yields a gradient image where each pixel encodes the local horizontal intensity change. It is possible to directly use this gradient image as input to the block matching stage.



(a) Original image

(b) Sobel filter applied image

Figure 4.1: Application of Sobel filter for stereo preprocessing

Correspondence Search (SAD Block Matching)

The core idea of block matching is to compare small windows (or blocks) of pixels between the left and right images, and find the best match based on

intensity similarity.

For each pixel (x_0, y_0) in the left image, a square window centered around it is extracted. This reference block is then compared with candidate blocks on the same row (due to rectification) in theright image. The comparison is performed using the Sum of Absolute Differences (SAD), computed as:

$$\text{SAD}(d) = \sum_{(i,j)\in W} |I_L(x_0 + i, y_0 + j) - I_R(x_0 - d + i, y_0 + j)|$$

where W is the window size (e.g., 5×5), I_L and I_R are the left and right images, and d is the disparity shift being tested.

The block with the lowest SAD value is considered the best match. The disparity assigned to pixel (x_0, y_0) is thus:

$$d = x_0 - x_{\text{best\_match}}$$

This means that the disparity is simply the horizontal distance between the center of the reference block in the left image and the center of the best-matching block in the right image.

The search is performed over a fixed disparity range defined by:

- `minDisparity`: the starting point for the search (default: 0)
- `numDisparities`: the number of disparity levels to evaluate (must be divisible by 16 for performance reasons)

For instance, with `minDisparity = 0` and `numDisparities = 64`, the algorithm evaluates all blocks from x_0 to $x_0 - 64$ in the right image. Disparity increases as matches move further left. Disparity can also be refined at subpixel resolution (e.g., 1/16th pixel steps), controlled by the `subpixelDisparities` parameter.

Post-Filtering

Once candidate matches are found, a post-filtering stage is applied to reject unreliable disparities. This helps to reduce false matches and noise in the disparity map:

- **uniquenessRatio** (default: 12): ensures that the best match is significantly better than the second-best. A match is rejected if:

$$\frac{\text{second\_best} - \text{best}}{\text{best}} < \text{uniquenessRatio}$$

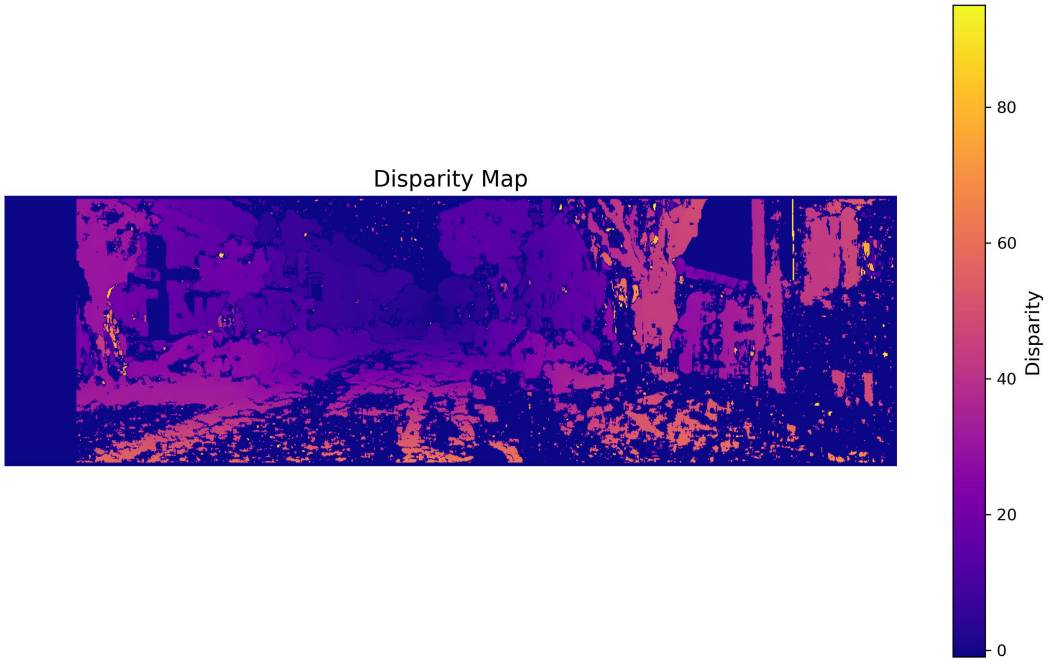
- **textureThreshold** (default: 10–15): discards matches in low-texture areas, where the SAD window response is below a threshold.

- **speckle filtering**: small regions of inconsistent disparity (called "speckles") are removed using the parameters `speckleWindowSize` (default: 9) and `speckleRange` (default: 4). This helps eliminate artifacts at object boundaries.

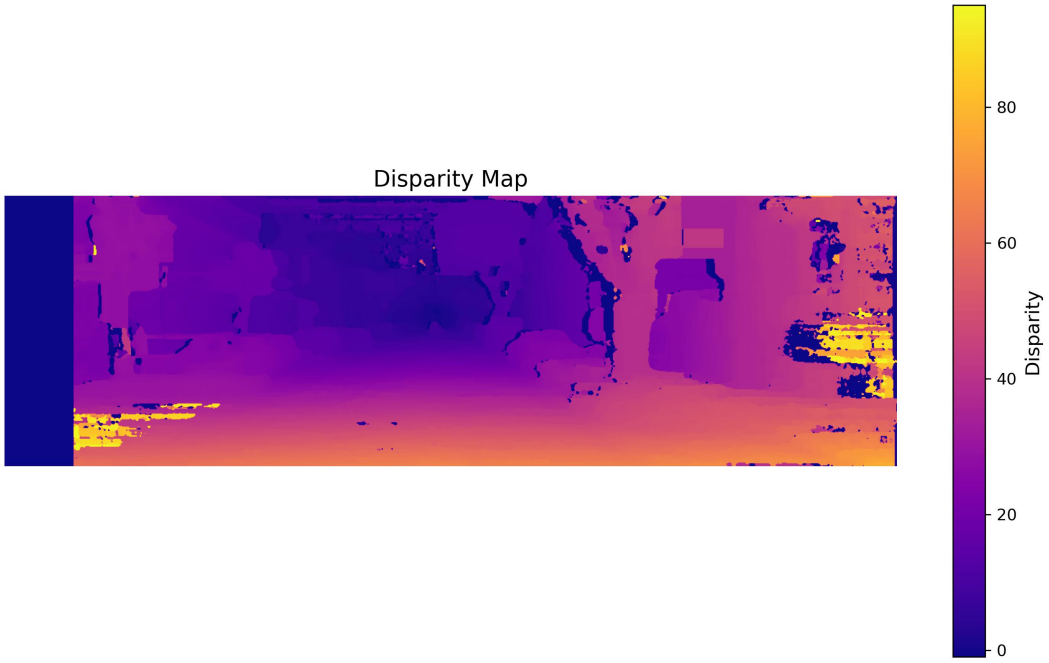
These filters improve the robustness of the final disparity map, especially in low-texture or occluded regions.

4.1.2 Results

A more detailed explanation has been provided for the StereoBM algorithm, as it represents a foundational method for understanding block-matching-based stereo correspondence. SGBM (Semi-Global Block Matching), while being a more advanced and accurate evolution of the same concept, is not analyzed in depth since its internal workings are not central to the goals of this thesis. However, for completeness, the reader is referred to the original paper describing the SGBM algorithm [6]. Below are the disparity maps obtained using both BM and SGBM:



(a) BM

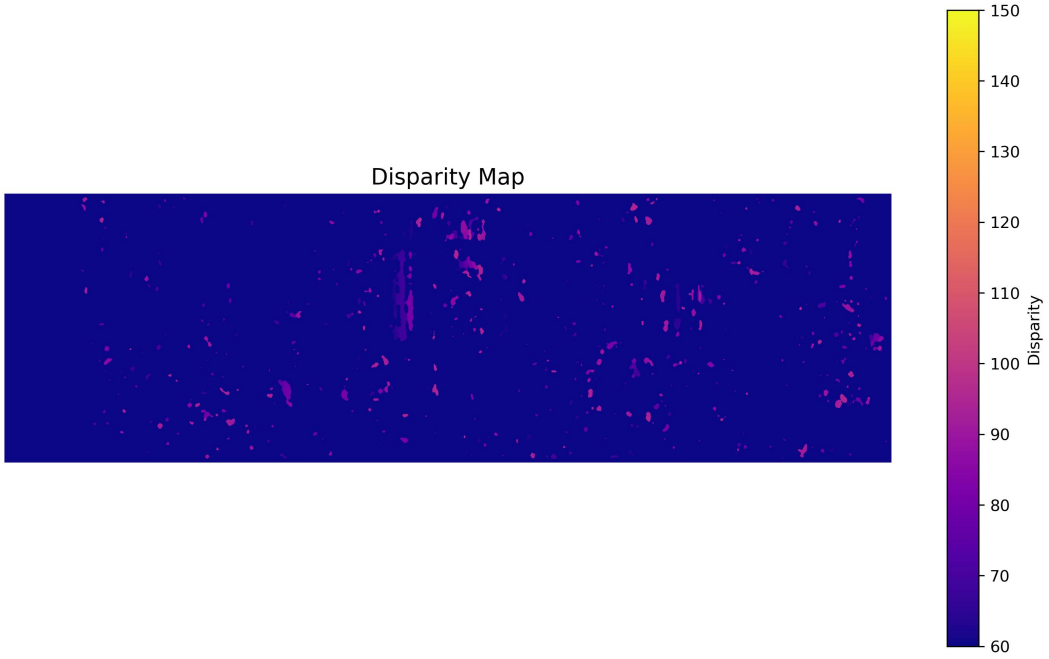


(b) SGBM

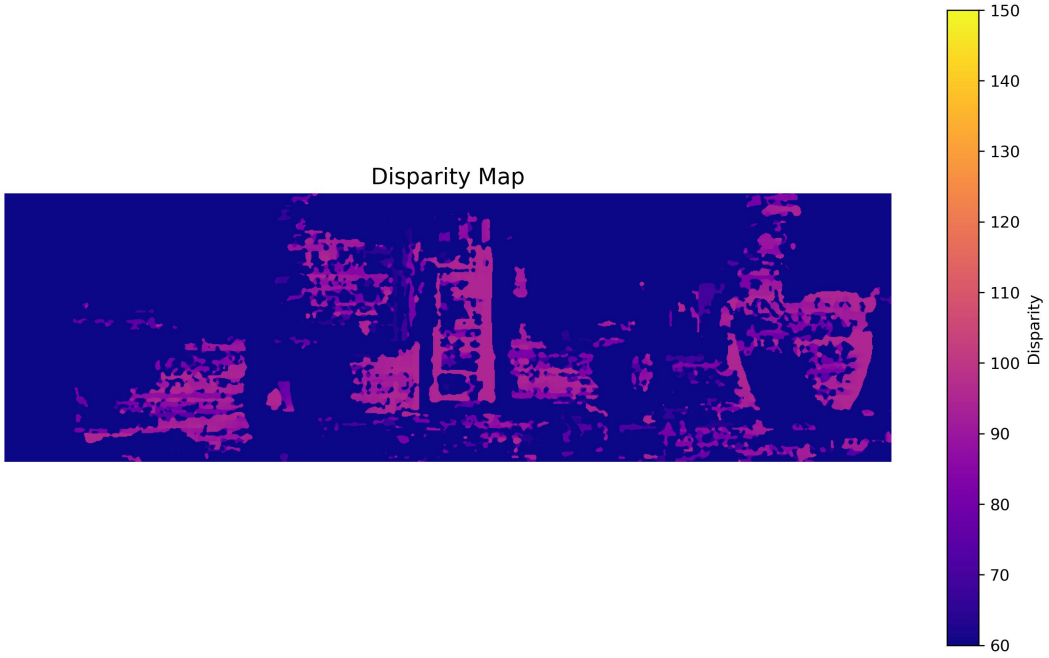
Figure 4.2: Comparison between disparity maps obtained using StereoBM and StereoSGBM on KITTI dataset.

Execution times:

- **StereoBM**: `0.009721 s`
- **StereoSGBM**: `0.475045 s`



(a) BM



(b) SGBM

Figure 4.3: Comparison between disparity maps obtained using StereoBM and StereoSGBM on Lab dataset.

Execution times:

- **StereoBM**: `0.15250 s`
- **StereoSGBM**: `0.41906 s`

It is worth noting the presence of a dark rectangle on the left side of the disparity maps. This region contains features visible in the left image but not in the right one. As a result, no valid correspondences could be found for those pixels, and they are marked as undefined or invalid in the disparity map.
However, the results confirm the observations discussed earlier. Although it requires more processing time, SGBM provides more accurate and detailed disparity maps. This is especially evident with the Lab dataset, where BM performs poorly and is insufficient for robust and reliable depth estimation. BM, therefore, requires highly textured scenes and very precise calibration and rectification to function properly. Even with the current dataset, it proves unusable in practice.

4.2 The depth map

Once the disparity map is obtained, it can be used to compute the depth map, which is a matrix where each pixel stores its estimated distance from the camera.
The calculation is based on a straightforward geometrical model derived from the similarity of triangles, as introduced in the pinhole camera model (see Chapter 2). The geometry of stereo vision is illustrated in Figure 4.4.

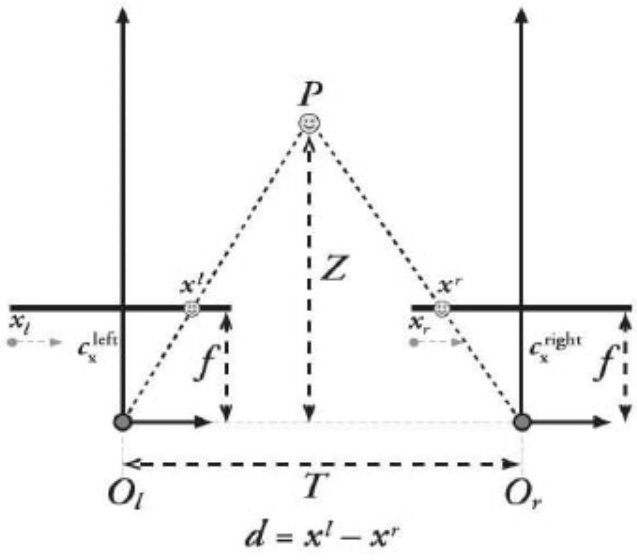


Figure 4.4: Stereo vision geometry and disparity formation. Source [7].

Given a stereo pair of rectified images, and assuming that each point is projected in the reference frame of its respective camera, the depth Z of a point can be derived from the similar triangles:

$$\frac{Z}{f} = \frac{X}{x_L}, \quad \frac{Z}{f} = \frac{T - X}{x_R}$$

where f is the focal length, T is the baseline (i.e., the distance between the two camera centers), x_L and x_R are the horizontal coordinates of the projection of the 3D point on the left and right images.
By defining the disparity as $d = x_L - x_R$, and solving the system, we obtain the fundamental equation:

$$Z = \frac{f \cdot T}{d}$$

This tells us that depth is inversely proportional to disparity: small disparities correspond to distant objects, while large disparities represent closer objects. Note that if:

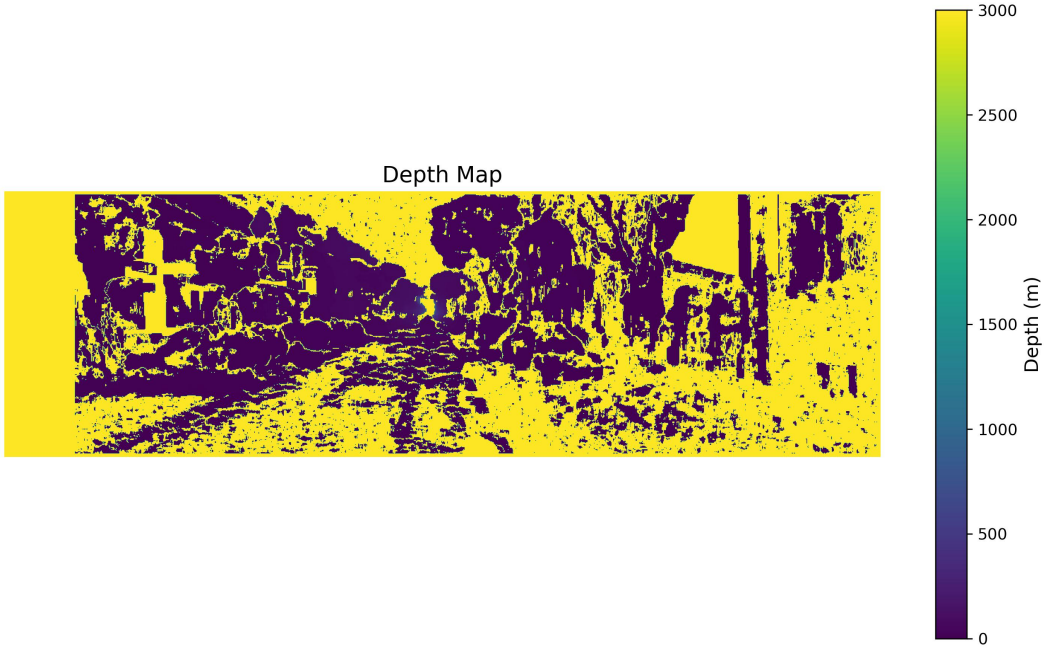
- The focal length f and the disparity d are measured in pixels;

- The baseline T is in meters (as is standard),

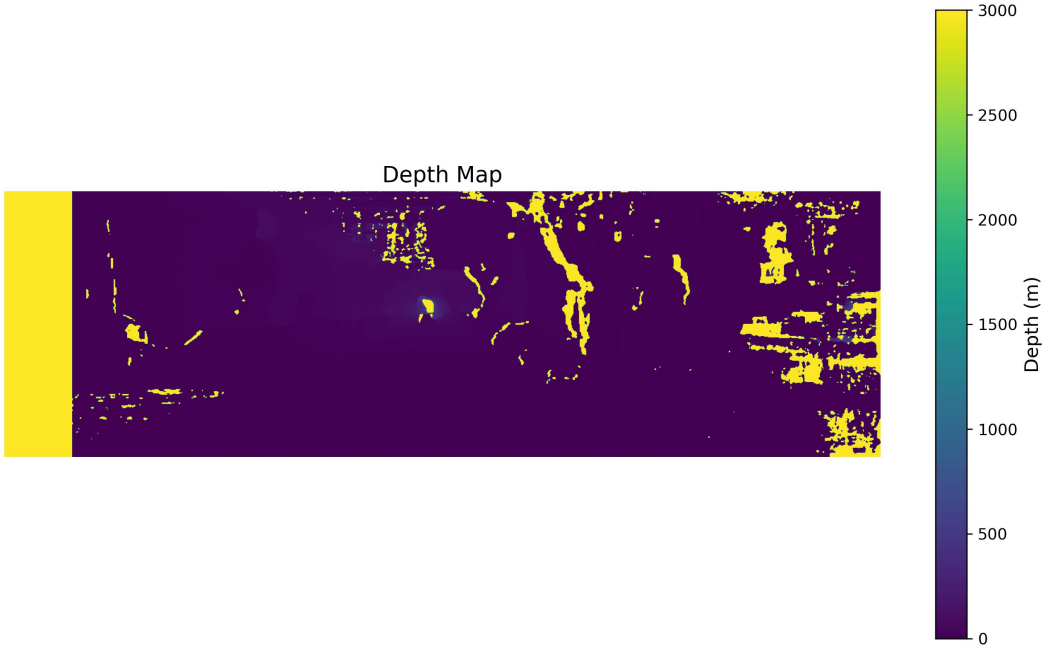
then the resulting depth Z is expressed in meters as well, which is ideal for a visual odometry pipeline.

Results

The following figures show the depth maps obtained using disparity maps computed respectively with StereoBM and StereoSGBM:

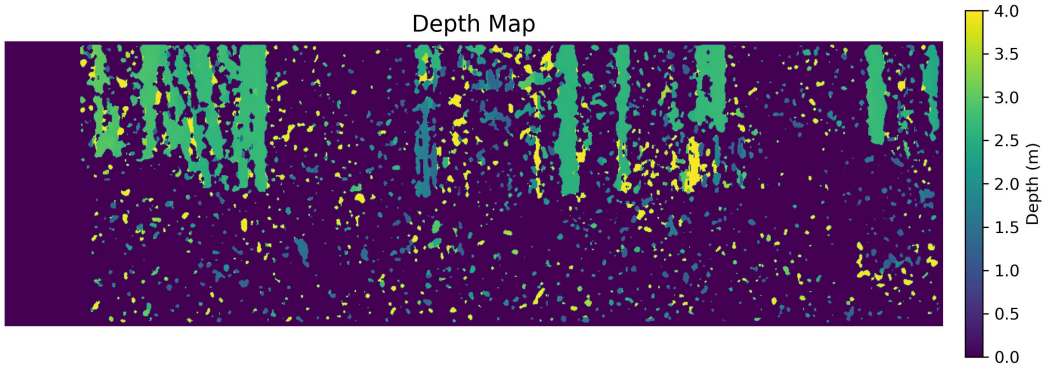


(a) BM

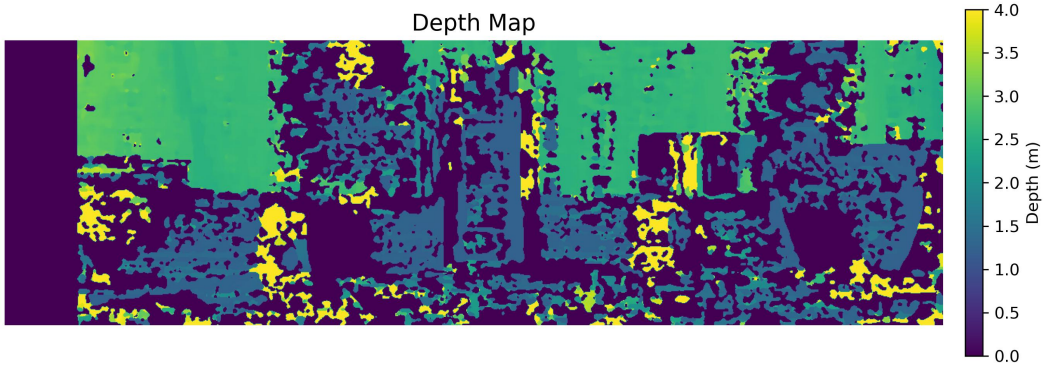


(b) SGBM

Figure 4.5: Comparison between depth maps obtained using StereoBM and StereoSGBM on KITTI dataset.



(a) BM



(b) SGBM

Figure 4.6: Comparison between depth maps obtained using StereoBM and StereoSGBM on Lab dataset.

The same considerations made for the disparity maps also apply here.

4.3 Feature tracking

4.3.1 Problem Introduction

Up to this point in the pipeline, we have considered only stereo image pairs, namely, left and right images captured at the same time instant. The main goal was to compute the disparity map for the left image, thus enriching it with depth information.

However, to estimate the trajectory of the moving camera, we must analyze how image features change over time. In particular, we now focus on tracking the features in the left image across different time instants, such as between I_t^L and I_{t+1}^L.

This task, commonly known as optical flow, consists in estimating the apparent motion of features between two consecutive frames.
Optical flow estimation is one of the most challenging problems in computer vision. Several factors complicate the tracking process, including:

- Changes in illumination,
- Local occlusions,
- Rotation,
- Changes in scale or viewpoint,
- Noise and motion blur.

For this reason, robust tracking algorithms are needed to ensure accurate feature matching over time. In the following sections, we will briefly describe the chosen tracking method.

4.3.2 Optical Flow Estimation Approaches

Several approaches exist to tackle the feature tracking problem. As in previous stages of the pipeline, deep learning-based methods were not considered due to their computational demands, which are incompatible with the available hardware and real-time processing requirements.
Classical approaches to optical flow estimation can be grouped into two main categories:

- **Direct optical flow estimation:** These methods attempt to compute the 2D motion vector directly for each point. They rely on strong assumptions, such as the brightness constancy constraint, which assumes that the intensity of a moving pixel remains constant across frames. Notable examples include:
  - **Lucas-Kanade** (1981): a sparse method that tracks only selected points across frames. It is efficient and widely used in real-time systems.

  - **Horn-Schunck** (1981): a dense method that estimates motion for all pixels in the image using a global energy minimization framework.

- **Feature descriptor-based tracking:** These methods address the optical flow problem indirectly. First, keypoints are extracted from the image, typically corners or other highly distinctive features. Each keypoint is then represented as a (potentially high-dimensional) vector descriptor. Matching is performed by searching for similar descriptors in the subsequent image. If a similar descriptor is found, a correspondence is established. Well-known algorithms in this category include:

  - **SIFT** (1999)
  - **ORB** (2011)

  These approaches are typically more computationally demanding, but tend to yield more accurate results.

Although classical methods like Lucas-Kanade are faster, in practice, they exhibited significant limitations when applied to the available dataset and hardware. As a result, the primary algorithm adopted in this pipeline is ORB.
ORB was designed as a faster alternative to SIFT, making it particularly suitable for real-time applications. Nevertheless, in challenging conditions where robustness is more critical than speed, SIFT can be used as an alternative due to its generally superior accuracy. The next section provides a brief technical overview of the ORB algorithm.

4.3.3 Technical Overview of ORB

ORB (Oriented FAST and Rotated BRIEF) was introduced in 2011 [8] as a fast and efficient alternative to more computationally intensive methods such as SIFT. It combines two key ideas: the FAST corner detector and the BRIEF descriptor, both modified to improve robustness and accuracy.

FAST

FAST (Features from Accelerated Segment Test) is a keypoint detector designed to efficiently find corners in an image. Its main advantage, as the name suggests, is its computational speed. It outperforms older detectors such as Harris or Difference of Gaussians (used in SIFT) in terms of efficiency.
The FAST algorithm considers a circle of 16 pixels (discretized using Bresenham's circle algorithm) surrounding a candidate point p. If a set of N contiguous pixels in the circle are all brighter than the intensity of p, denoted I_p, plus a threshold t, or all darker than $I_p - t$, the point p is classified as a corner.

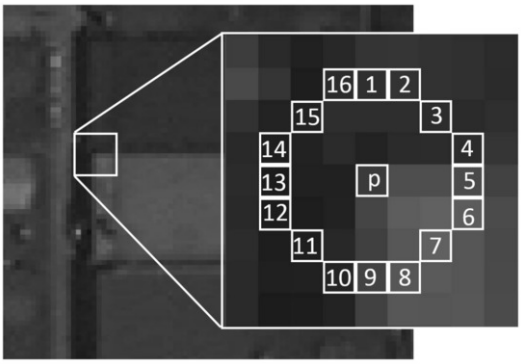


Figure 4.7: Bresenham circle used by FAST detector. Source: Wikipedia.

This check can be expressed through two conditions:

- **Condition 1:** $\forall x \in S, I_x > I_p + t$ (dark corner on bright background)
- **Condition 2:** $\forall x \in S, I_x < I_p - t$ (bright corner on dark background)

To reduce computation, FAST first applies a high-speed test that examines only four pixels (positions 1, 5, 9, and 13) around the candidate point. If at least three of these are not significantly brighter or darker than the center pixel, the candidate is immediately discarded, since it's unlikely to be a corner. Thanks to this early rejection strategy, most non-corner points are eliminated very quickly, and on average, only 3.8 pixels are checked per candidate instead of all 16.
Although FAST efficiently detects keypoints, it lacks orientation estimation.

Without it, descriptors are not rotation-invariant, which makes matching unreliable if the scene is rotated. ORB resolves this by computing an orientation angle for each keypoint, which is then subtracted before the descriptor is computed, normalizing keypoints to a common frame.
Additionally, ORB ranks and selects the best keypoints using the Harris corner response measure to retain only the most stable ones.

BRIEF

Once keypoints are detected, they are encoded into vectors using BRIEF (Binary Robust Independent Elementary Features). BRIEF creates a binary string, a compact descriptor, that serves as a unique "fingerprint" for each keypoint.
For each keypoint, a patch (typically 31×31) is extracted around it. Although patches may overlap, each is centered on a different keypoint and thus contains different content.
A set of fixed pixel pairs is selected within the patch (randomly or through learned patterns). For each pair (p_a, p_b), a binary test is performed:

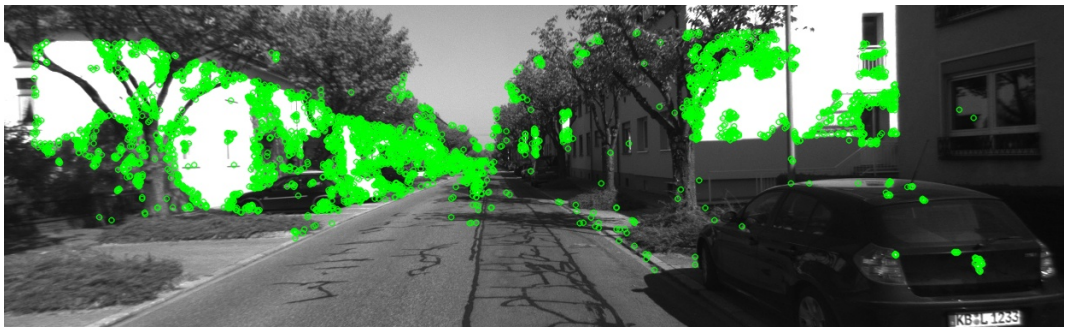
$$\text{bit} = \begin{cases} 1 & \text{if } I(p_a) < I(p_b) \\ 0 & \text{otherwise} \end{cases}$$

This results in a binary descriptor vector, usually 256 bits long. The selection of pixel pairs remains fixed to ensure consistency across frames.
ORB enhances BRIEF by using a rotated version, called **rBRIEF**, where the patch is rotated by the orientation angle computed from FAST before descriptor extraction. This step ensures robustness to rotation.

Summary of ORB Robustness

ORB is robust to image rotation and moderate changes in illumination, making it ideal for real-time applications. This robustness to illumination changes largely comes from the BRIEF descriptor, which compares pixel intensities locally; since illumination changes tend to affect neighboring pixels similarly, the binary tests remain stable. However, ORB is not scale-invariant by default,

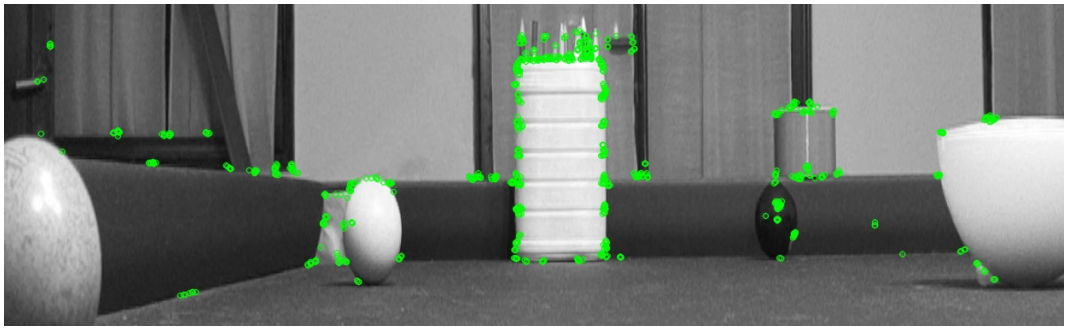
which can reduce performance in scenes with significant scale variation. Despite this, additional techniques (not discussed here) can be used to improve scale robustness when needed.



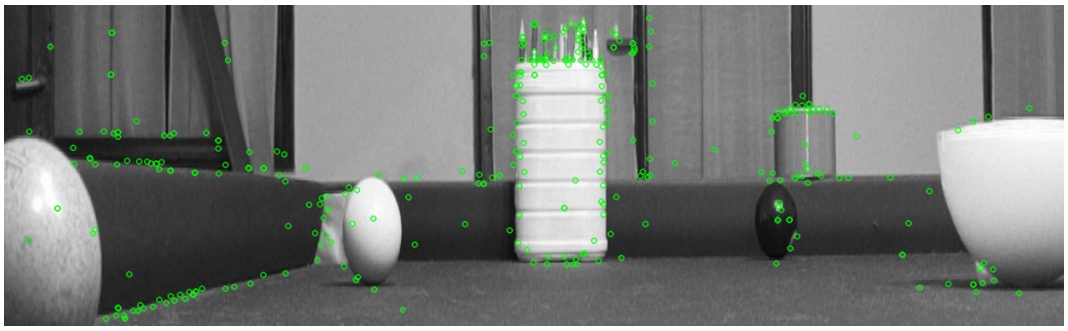
(a) ORB

(b) SIFT

Figure 4.8: Comparing features extraction using ORB and SIFT on KITTI dataset.



(a) ORB



(b) SIFT

Figure 4.9: Comparing features extraction using ORB and SIFT on Lab dataset.

Table 4.1: Comparison betwwn ORB and SIFT

Algoritmo	**Dataset**	**Tempo (s)**	**# Keypoints**
ORB	Lab	0.002744	873
SIFT	Lab	0.065704	352
ORB	KITTI	0.024575	4972
SIFT	KITTI	0.097083	3420

As expected, the algorithms detected significantly more features in the KITTI dataset. ORB proved to be much faster and consistently detected a higher number of features in both cases. This is generally true, as ORB applies less strict criteria to classify a point as a feature. However, relying solely on this aspect to evaluate a feature detection algorithm would be a serious mistake. For instance, visual inspection of the results reveals that SIFT keypoints in the KITTI dataset are more widely distributed compared to those of ORB, which may suggest better generalization capabilities. Therefore, to carry out a more meaningful comparison, it is necessary to assess the robustness of the detected features, specifically, how many of them can be matched in the following frame.

4.3.4 Feature matching

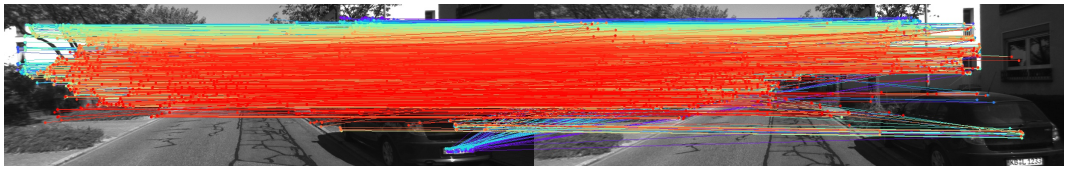
Now that the features have been extracted, the next step is to identify matches, which are the correspondences between salient points in the two images. A match is considered when a feature detected in the left image has a similar descriptor to that of a feature in the right image.

Matching algorithms

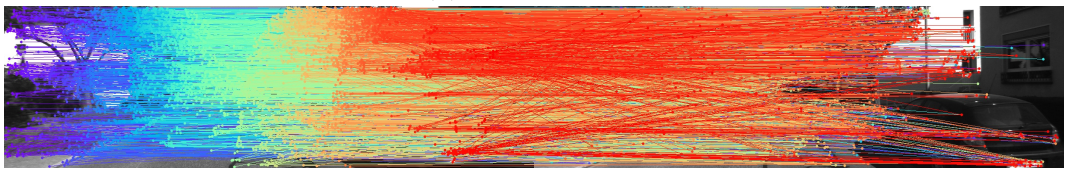
One of the matching strategies employed in this thesis is the BFMatcher (Brute Force Matcher). This approach simply compares the descriptor of a feature in the first image with all descriptors in the second image and returns the best match based on a chosen distance metric.
BFMatcher supports both binary and floating-point descriptors. For binary descriptors, such as those produced by ORB, the Hamming distance is used. For floating-point descriptors, like those from SIFT, the Euclidean distance is employed.
While BFMatcher combined with binary descriptors and Hamming distance is a common, simple, and fast solution, its performance with floating-point descriptors tends to be slower. In such cases, the FLANN matcher (Fast Library for Approximate Nearest Neighbors) is a better alternative.
FLANN is designed for efficient nearest neighbor searches in high-dimensional spaces. It leverages optimized data structures like k-d trees and algorithms such as k-means clustering to significantly reduce the search time. This makes it particularly well-suited for matching SIFT descriptors.
Therefore, the matching algorithm and descriptor pairs used in this work are:

- **ORB** + `BFMatcher` (Hamming distance)

- **SIFT** + `FLANN` (Euclidean distance with approximate search)

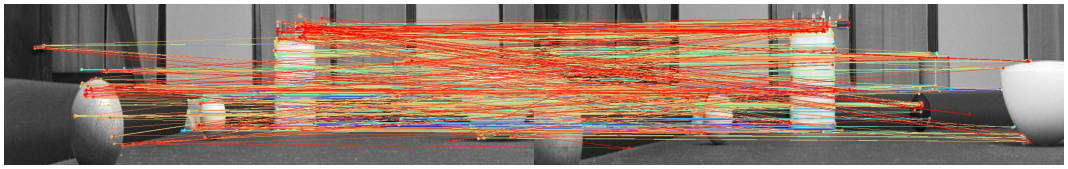


(a) ORB + BFM

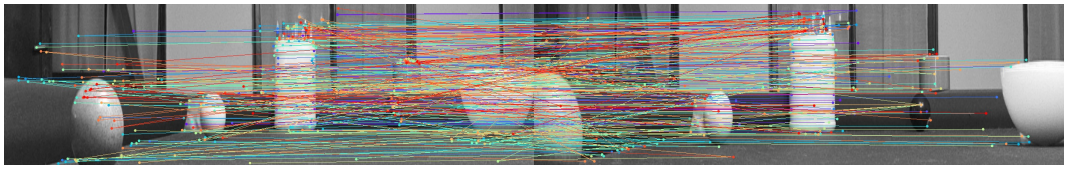


(b) SIFT + FLANN

Figure 4.10: Comparing features matching using ORB+BFM and SIFT+FLANN on KITTI dataset.



(a) ORB+BFM



(b) SIFT+FLANN

Figure 4.11: Comparing features matching using ORB+BFM and SIFT+FLANN on Lab dataset.

Table 4.2: Comparison between ORB+BFMatcher and SIFT+FLANN

Algoritmo	**Dataset**	**Tempo (s)**	**# Match**
ORB + BFMatcher	Lab	0.01	955
SIFT + FLANN	Lab	0.02	345
ORB + BFMatcher	KITTI	0.21	4920
SIFT + FLANN	KITTI	0.33	3361

Filtering matches

In feature tracking systems, once matches between descriptors are obtained, it is common practice to apply a filtering step to increase robustness and reduce the percentage of outliers, i.e., incorrect correspondences that may degrade subsequent estimations (e.g., within a RANSAC scheme).

One of the most widely used filtering methods is the ratio test, introduced in Lowe's seminal 2004 paper [9]. The general principle of the ratio test is simple: for each descriptor in the first image, the two best matches in the second image are identified, along with their respective distances, denoted as d_1 and d_2, with $d_1 \leq d_2$. The match is accepted only if the ratio

$$\frac{d_1}{d_2} < \tau$$

where τ is a threshold value, indicating that the best match is significantly better than the second-best one. This helps discard ambiguous matches where the first and second best matches are too similar.

To apply the ratio test, matching algorithms, both brute-force and approximate methods like FLANN, return the two best candidate matches for each descriptor, sorted by distance. This allows an easy comparison between the two closest matches for filtering.

In this thesis, the threshold value τ used is 0.5, chosen to balance between keeping correct matches and discarding uncertain ones. A lower threshold results in stricter filtering, potentially removing more matches but increasing reliability.

Table 4.3: Effect of the ratio test ($\tau = 0.5$)

Method	**Dataset**	**Raw Matches**	**After Ratio Test**	**Ratio**
ORB + BF	Lab	955	47	0.049
SIFT + FLANN	Lab	345	93	0.269
ORB + BF	KITTI	4920	1059	0.215
SIFT + FLANN	KITTI	3361	1052	0.313

As expected, SIFT has a higher ratio than ORB. In particular, we see that in the Lab dataset, ORB provides many less reliable matches, and the robust ones are actually fewer than those found with SIFT. The quality of matches filtered by the ratio test directly impacts the robustness of the geometric estimation phase, which will be analyzed later using RANSAC-based inlier ratios

4.4 Pose estimation

4.4.1 Definition of the problem

To summarize the pipeline so far, we have reconstructed the 3D position of a set of points at time t, thanks to the depth map computed from stereo disparity. Then, we have identified the corresponding 2D image locations of these points in the next left frame at time $t + 1$, using feature tracking.
The next objective is to estimate the camera pose, that is, the rigid-body transformation between the two consecutive time steps. Specifically, we aim to recover the rotation matrix $R \in \mathbb{R}^{3\times3}$ and the translation vector $\mathbf{t} \in \mathbb{R}^3$ that describe the motion of the camera from time t to $t + 1$.
This task is formally known as the Perspective-n-Point (PnP) problem, and it assumes a pinhole camera projection model, already discussed in Chapter 2.
Solving this problem allows us to compute the trajectory of the camera by concatenating the relative poses estimated at each step.

4.4.2 Solving the PnP problem with solvePnPRansac

Introduction

To solve the pose estimation problem, the algorithm `cv2.solvePnPRansac` provided by OpenCV was adopted. This approach combines an efficient algorithm for solving the PnP problem, such as EPnP, with a robust RANSAC scheme.
The integration of RANSAC allows for the identification and rejection of outliers, significantly improving the reliability of the estimated pose. Furthermore, it provides a useful metric for evaluating the quality of the feature matches obtained in the previous tracking phase.
To better understand the workings of `solvePnPRansac`, we will now examine the EPnP algorithm and how it is embedded within the RANSAC framework.

The EPnP algorithm

EPnP (Efficient Perspective-n-Point) [10] aims to improve efficiency when many correspondences are available. The core idea is to express the 3D points as a linear combination of four virtual control points with normalized weights. In fact, to represent any point in an n-dimensional space, $n+1$ points are needed. A simple way to choose these control points is to select the centroid of the point cloud as the first point, and then define the other three along the coordinate axes starting from the centroid. For example, if the centroid is $\mathbf{c}$, then the control points can be:

$$\mathbf{C}_1^w = \mathbf{c}, \quad \mathbf{C}_2^w = \mathbf{c} + \Delta x\hat{\mathbf{i}}, \quad \mathbf{C}_3^w = \mathbf{c} + \Delta y\hat{\mathbf{j}}, \quad \mathbf{C}_4^w = \mathbf{c} + \Delta z\hat{\mathbf{k}},$$

where $\hat{\mathbf{i}}, \hat{\mathbf{j}}, \hat{\mathbf{k}}$ are the unit vectors along the x, y, z axes, and $\Delta x, \Delta y, \Delta z$ are offsets based on the spread of the point cloud.

Formally, each 3D point $\mathbf{X}_i^w$ in the world coordinate system can be rewritten as:

$$\mathbf{X}_i^w = \sum_{j=1}^{4} \alpha_{ij} \mathbf{C}_j^w,$$

where $\mathbf{C}_j^w$ are the coordinates of the virtual control points in the world reference frame and α_{ij} are the normalized weights satisfying $\sum_j \alpha_{ij} = 1$.

Switching to the camera coordinate system, each 3D point can be expressed as:

$$\mathbf{X}_i^c = \sum_{j=1}^{4} \alpha_{ij} \mathbf{C}_j^c,$$

where $\mathbf{C}_j^c$ are the coordinates of the control points in the camera frame. These are related to the control points in the world frame $\mathbf{C}_j^w$ via a rigid-body transformation:

$$\mathbf{C}_j^c = R\mathbf{C}_j^w + \mathbf{t},$$

where $R \in \mathbb{R}^{3\times3}$ is the rotation matrix and $\mathbf{t} \in \mathbb{R}^3$ is the translation vector. Substituting into the previous equation, we get:

$$\mathbf{X}_i^c = \sum_{j=1}^{4} \alpha_{ij}(R\mathbf{C}_j^w + \mathbf{t}) = R\left(\sum_{j=1}^{4} \alpha_{ij}\mathbf{C}_j^w\right) + \mathbf{t}.$$

This shows that once the coordinates of the control points in the camera frame are known, the camera pose (i.e., R and $\mathbf{t}$) can be estimated. The problem thus reduces to finding the $4 \times 3 = 12$ unknowns representing the camera-frame positions of the control points.

Using the pinhole camera model (with intrinsic parameters f_x, f_y, u_0, v_0), the projection equations for point i are:

$$\sum_{j=1}^{4} \alpha_{ij} \left(f_x x_j^c + (u_0 - u_i) z_j^c\right) = 0,$$

$$\sum_{j=1}^{4} \alpha_{ij} \left(f_y y_j^c + (v_0 - v_i) z_j^c\right) = 0,$$

where (u_i, v_i) are the observed pixel coordinates of point i in the image, and (x_j^c, y_j^c, z_j^c) are the coordinates of the control points in the camera frame.

Collecting these equations for all points i gives a linear system of equations that can be written in matrix form as:

$$\mathbf{M}\mathbf{x} = \mathbf{0},$$

where $\mathbf{x}$ is the vector of unknown control points coordinates in the camera frame. This system is solved via Singular Value Decomposition (SVD), yielding the coordinates of the control points. From these, the rotation R and translation $\mathbf{t}$ can be recovered.

RANSAC scheme

RANSAC (RANdom SAmple Consensus) is an iterative algorithm used to estimate the parameters of a mathematical model from a dataset that contains outliers. It is a widely adopted strategy not only in computer vision, but also in fields such as robotics, and data fitting.

In the context of camera pose estimation, RANSAC helps to robustly estimate the rotation R and translation $\mathbf{t}$ by rejecting incorrect 2D–3D correspondences

(outliers). The general structure of the RANSAC algorithm is as follows:

1. Randomly select a minimal subset of 2D–3D correspondences (at least four are required for EPnP to produce a valid estimate).

2. Use EPnP to compute a candidate camera pose $(R, \mathbf{t})$ from this subset.

3. Project all 3D points $\mathbf{X}_i$ into the image using the candidate pose:

$$\mathbf{x}_i^{\text{proj}} = K[R \mid \mathbf{t}]\mathbf{X}_i,$$

   where K is the intrinsic camera matrix and $\mathbf{x}_i^{\text{proj}}$ is the projected 2D point.

4. Compute the reprojection error for each correspondence:

$$e_i = \|\mathbf{x}_i^{\text{proj}} - \mathbf{x}_i^{\text{obs}}\|,$$

   where $\mathbf{x}_i^{\text{obs}}$ is the observed 2D point in the image.

5. Count how many correspondences have an error e_i below a given threshold, these are considered inliers.

6. Repeat the above steps for a fixed number of iterations or until a sufficiently high number of inliers is found.

At the end of the iterations, the pose with the highest number of inliers (or the lowest total reprojection error among the best models) is selected as the final solution. This method improves robustness, especially when some of the input correspondences are incorrect due to noise or tracking failures.

4.4.3 Results

The trajectory will be shown in the next section. Here we simply show the % of inliers in the matches.

Table 4.4: Percentage of Inliers after pose estimation on Lab and KITTI datasets

Algorithm + Matcher	**Dataset**	**% Inliers (valid depth points)**
ORB + BFMatcher	Lab	75.00% (12/16)
SIFT + FLANN	Lab	76.72% (20/26)
ORB + BFMatcher + SIFT	KITTI	99.21% (874/881)
SIFT + FLANN	KITTI	99.02% (909/918)

Note that, in the Lab dataset, many associated points obtained an invalid depth and were therefore discarded by the algorithm (e.g., from 93 points in the SIFT+FLANN case, we dropped to only 26). As expected, however, in KITTI, the percentage of inliers is significantly higher. This is, in fact, a varied, outdoor dataset with many strong features available. The more "difficult" conditions of the Lab dataset created problems for the algorithm, which, however, achieved decent results.

4.5 Computing the trajectory

4.5.1 Pose concatenation

Once the relative camera pose between two consecutive frames has been estimated, it becomes possible to compute the full trajectory of the camera by iterating this process across the entire image sequence.

The goal is to estimate the position of the camera with respect to a fixed world coordinate system, which, without loss of generality, is chosen to coincide with the camera pose at the initial frame $t = 0$. In other words, we want to answer the question: *"Where is the camera located with respect to its original position?"*

The function `cv2::solvePnPRansac` provides as output a rotation vector $\mathbf{r} \in \mathbb{R}^3$ and a translation vector $\mathbf{t} \in \mathbb{R}^3$. The rotation vector is then converted to a 3×3 rotation matrix $R \in \mathbb{R}^{3\times3}$ via the Rodrigues transformation, as previously discussed:

$$R = \texttt{Rodrigues}(\mathbf{r}).$$

These parameters define the rigid-body transformation $[R|\mathbf{t}]$ that maps 3D points from the coordinate system at time t to the coordinate system at time $t+1$. However, for trajectory reconstruction, we are interested in expressing each pose with respect to the initial frame $t = 0$. This means that at each step, we need to invert the current relative transformation before composing it with the accumulated global pose.

To facilitate this, the relative pose is embedded into a homogeneous transformation matrix:

$$T_{t\to t+1} = \begin{bmatrix} R & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4}.$$

This representation allows us to easily invert the transformation:

$$T_{t+1\to t} = T_{t\to t+1}^{-1},$$

and compose it with the global transformation T_{global} accumulated up to time t:

$$T_{\text{global}}^{(t+1)} = T_{\text{global}}^{(t)} \cdot T_{t+1\to t}.$$

By storing the global transformation matrix at each frame, we build up the full camera trajectory relative to the initial frame. At each time step, the 3×1 block of the matrix (i.e., the translation vector) contains the 3D position of the camera in world coordinates:

$$\mathbf{p}^{(t)} = T_{\text{global}}^{(t)}[0:3,\ 3].$$

In this way, the trajectory is constructed as a sequence of transformations, each of which relates the camera pose at a given time step to the origin frame.

4.5.2 Results

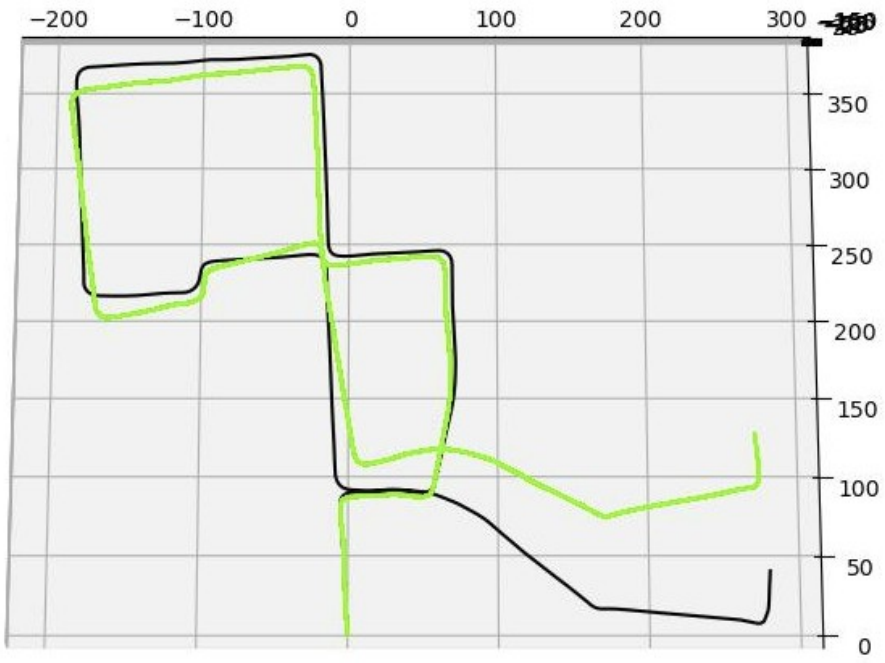


Figure 4.12: KITTI dataset trajectory, ground truth (black) vs estimated trajectory (green).

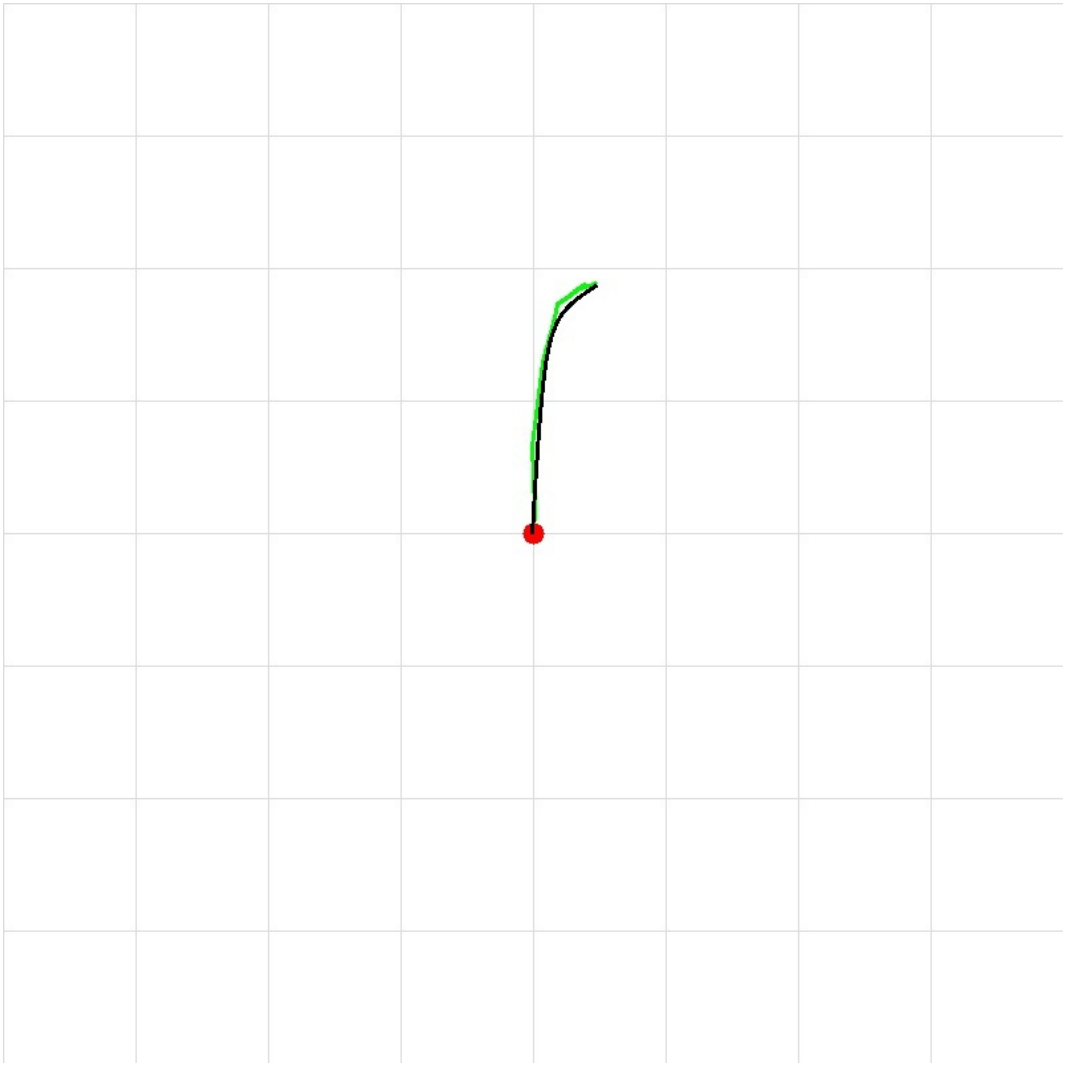


Figure 4.13: Lab dataset trajectory, ground truth (black) vs estimated trajectory (green). Square size: 1.5m

Chapter 5

Field testing

The system is now being tested on a dataset acquired in an uncontrolled environment. During a mission to Chioggia with the Nautilus team, a dataset was collected along one of the city's canals.

5.1 Setup

The stereo vision modules described in chapter 3 were used to acquire data. In particular, two of them were mounted on wooden arms and rigidly attached to the front and back ends of a stand-up paddleboard (SUP), as shown in Figure 5.1. The wooden mechanical arms allowed for flexible adjustment of both height and pitch angle of each module relative to the water surface. This adaptability made it possible to change the acquisition geometry depending on the environment and scene, improving the effectiveness of the visual odometry pipeline under various conditions.

To monitor internal thermal conditions during outdoor experiments, often conducted under high ambient temperatures, temperature sensors were installed within each module. In many cases, active cooling was not sufficient to prevent thermal throttling. Monitoring allowed for early detection of overheating on the Raspberry Pi 5.

Each stereo module was further protected from water splashes by a transparent plexiglass plate covering the cameras.

The use of two stereo vision modules, one facing forward and the other back-

ward, increased the robustness of the system in constrained environments like narrow canals. In these scenarios, parts of the scene might be temporarily occluded or lack sufficient texture. A back-facing module enhances the chance of capturing reliable features for tracking and allows for backward motion analysis when landmarks are no longer visible in the forward direction.

(a) Front-mounted stereo module (b) Side view of the complete setup

Figure 5.1: Stereo vision system mounted on the stand-up paddleboard (SUP)

5.2 Observed results

We can now observe the results of the main steps of the system.

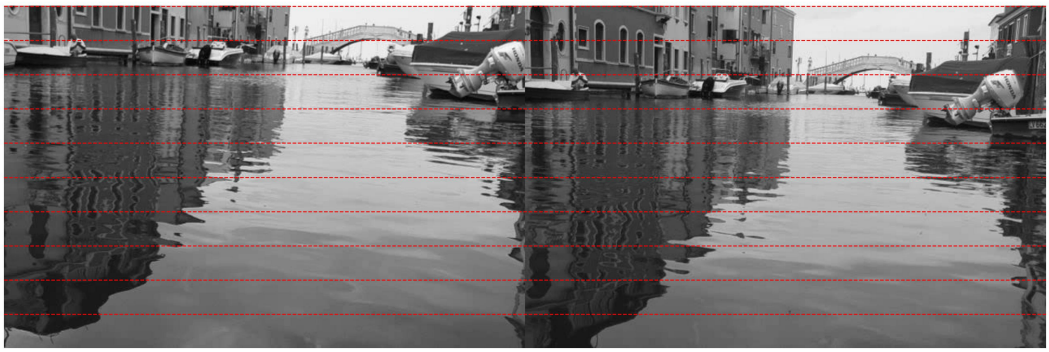


Figure 5.2: Epipolar lines on a Chioggia dataset frame.

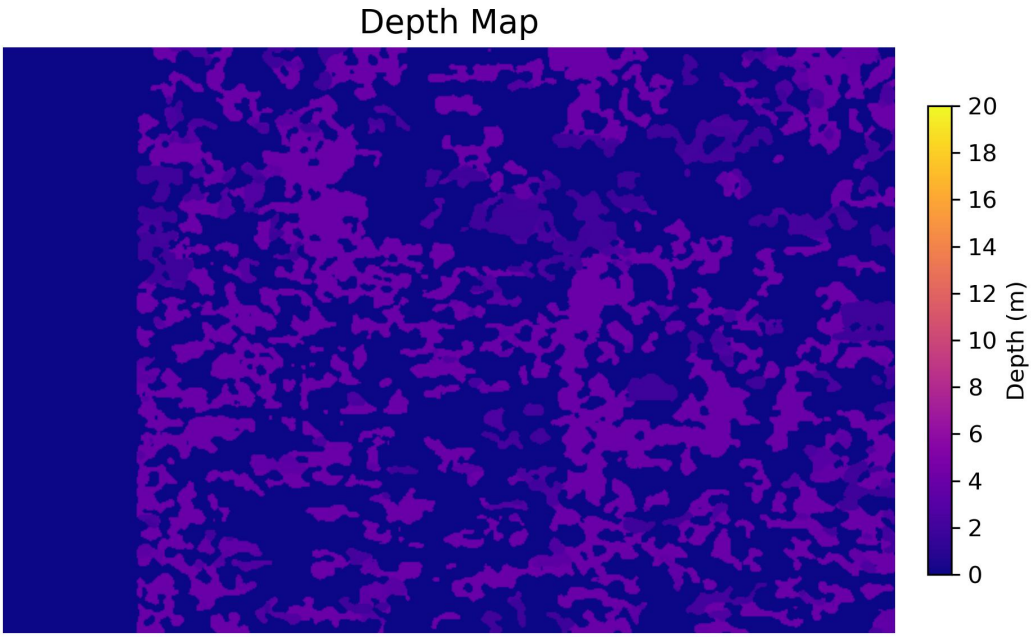


Figure 5.3: Depth map (SGBM) of a Chioggia dataset frame.

From the very first steps, it's clear that the system encountered significant difficulties. The left and right images have a significant vertical offset, and as a result, the depth map, even though it's produced by SGBM disparities, is very noisy and unable to distinguish any details of the real scene. Furthermore, all the points are between 0 and 8 m deep, which isn't realistic for the dataset.

Figure 5.4: Matching features (ORB+BFM) on Chioggia dataset.

For feature tracking, ORB with Brute Force Matching was used, as it yielded a higher number of robust matches. Specifically, 46 matches were found, with an inlier ratio of 93.75% (15 out of 16). Although this number is relatively low, it was still sufficient for the EPnP algorithm to proceed.

However, clearly incorrect depth values led to a final trajectory that did not reflect the real motion of the left camera.

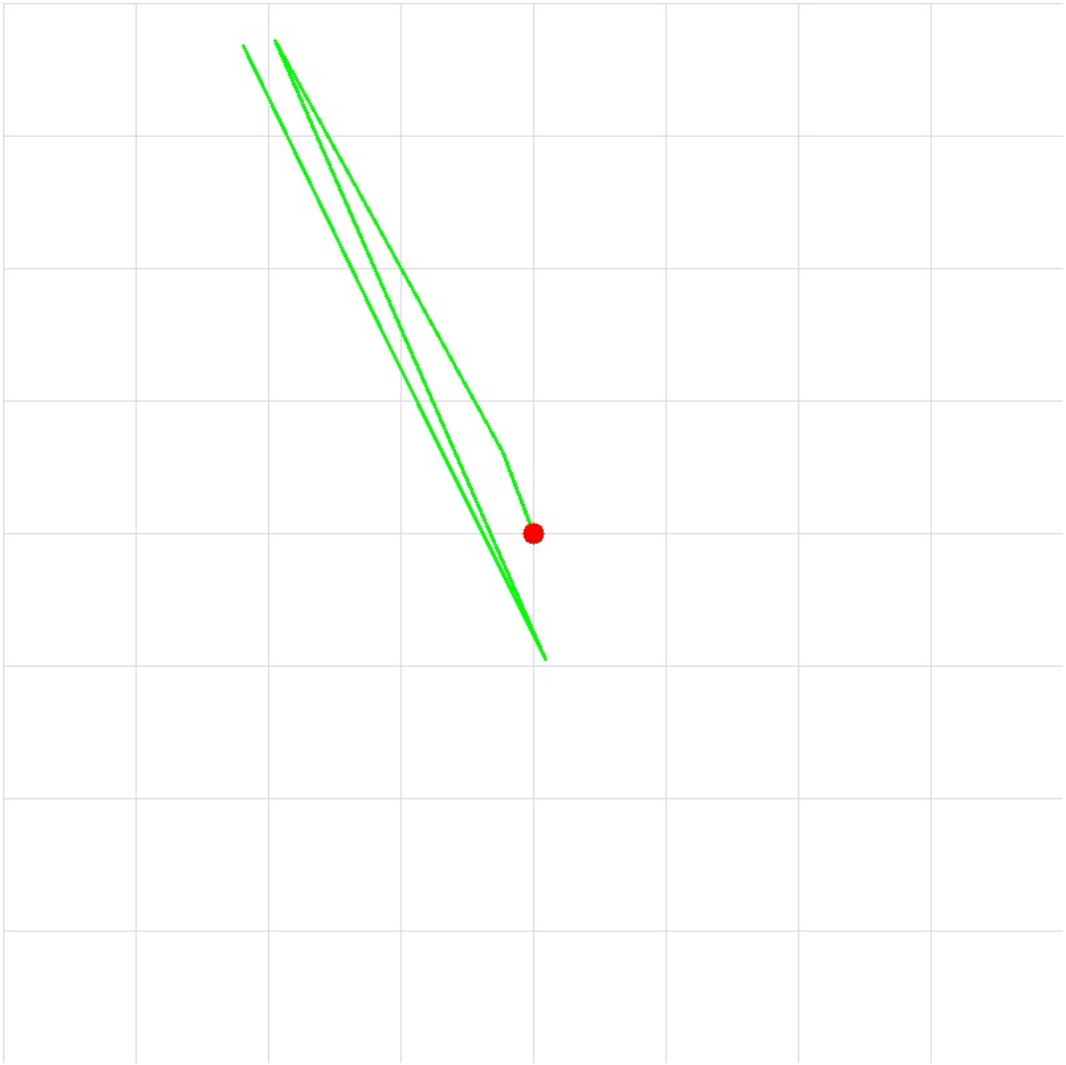


Figure 5.5: Chioggia dataset trajectory. Square size: 10 m

5.3 Considerations

Although the feature tracking did not produce a highly robust set of matches, the most significant limitations emerged during the image rectification step. This section outlines the main problems observed during the field test and presents possible improvements for a future acquisition in Chioggia.

Problems

- **Potentially dynamic environment:** The developed system lacks specific mechanisms to remain robust in scenes containing moving elements (e.g., water reflections, people), which may have introduced inconsistencies in feature tracking and depth estimation.

- **Inaccurate camera mounting:** The stereo cameras were fixed inside the box using non-rigid or imprecise methods. Even minor shifts during data acquisition may have invalidated the calibration parameters, leading to poor rectification and consequently unreliable trajectory estimation.

- **High percentage of low-texture surfaces:** The scene contained a large portion of uniform surfaces (e.g., the water in the canal), which are particularly challenging for stereo disparity computation and feature extraction. This led to invalid or erroneous depth measurements and a low number of robust matches.

- **Low-quality calibration setup:** Calibration was performed using a printed checkerboard on cardboard rather than a professional calibration target. Potential bending, reflections, or misalignment during the process may have degraded the intrinsic and extrinsic parameter estimation.

- **Camera motion due to unstable support:** The camera box exhibited visible oscillations due to water motion, introducing additional noise into the image stream and compromising both feature tracking and rectification stability.

Possible improvements

For a second mission in Chioggia, several adjustments can be made to mitigate the issues observed during the first acquisition. The following measures are proposed:

- **Rigid stereo mount:** The most critical improvement is ensuring that

the two cameras remain perfectly fixed relative to each other. Any mechanical shift during recording inevitably invalidates the trajectory

- **Level camera orientation:** The stereo arm should be kept parallel to the ground, rather than tilted downward. This configuration minimizes the proportion of low-texture surfaces, improving disparity estimation and increasing the number of trackable features.

- **Improved calibration setup:** Maybe using a professional, flat, and non-reflective calibration target (e.g., rigid checkerboard panel) under controlled lighting conditions could be useful. This will certainly improves the calibration process

- **Motion planning:** If possible, adopt a smoother motion path to simplify pose estimation and allow for more consistent tracking. This may require coordinating the paddling method so that it generates less lateral motion.

Chapter 6

Conclusions and future developments

This thesis presented the development of a complete stereo visual odometry pipeline, designed to operate with calibrated images and provide fairly accurate motion estimation in static environments. The focus was not on optimizing every component for real-time execution, but rather on building a functional and modular system that allows for performance evaluation and future expansion. Although no hard constraints were set for computational time, algorithmic choices prioritized lightweight alternatives, alongside a few more complex ones to benchmark accuracy and runtime.

The system was evaluated on both the KITTI dataset and a custom Lab dataset. On KITTI, results were qualitatively good, although a noticeable drift accumulated over long trajectories, as is typical in VO systems without global optimization. In the Lab dataset, which consisted of short sequences (approximately 20 frames), the estimated trajectory appeared globally more accurate than in the KITTI dataset. However, this is primarily due to the limited sequence length, which reduces the chance of drift accumulation. By analyzing intermediate results, such as feature tracking stability and disparity consistency, it becomes evident that the KITTI dataset provides more accurate outputs. Indeed, if only the initial frames of KITTI were considered, results would be comparable or even superior to those from the Lab dataset. A valu-

able future test would involve capturing a longer Lab sequence to assess the long-term robustness of the system under custom calibration and rectification conditions.

Testing on an uncontrolled environment (the Chioggia dataset) revealed a strong sensitivity to extrinsic camera perturbations during acquisition, which significantly degraded the estimated trajectory. This highlights a known limitation of VO systems: the need for stable calibration throughout operation. To address this, future developments could include self-calibration mechanisms, especially relevant for real-time use.

Self-calibration is also particularly useful in complex environments such as underwater, where standard calibration is often impractical. Although this thesis did not include testing on an aquatic dataset, it is likely that calibration would represent a major challenge. For this reason, the introduction of concepts like the Fundamental and Essential matrices and the Image of the Absolute Conic (IAC) aims to support future self-calibrating systems capable of operating in such conditions. Future work may therefore explore:

- Collection and testing of underwater sequences, ideally with artificial lighting.
- Robustification of the system in dynamic scenes through segmentation or obstacle-aware tracking;
- Fusion with additional sensors (e.g., IMU), enabling probabilistic filtering (e.g., Kalman filter) to refine motion estimation;
- Real-time deployment optimization;
- Improving system robustness by adding an auto-calibration mechanism;
- Bundle adjustment and loop closure integration for long-term consistency;

Although ground truth was not available for all test sequences, each step of the pipeline was evaluated to ensure internal consistency and correct behavior. In this sense, the results and visual outputs (disparity maps, matching quality,

trajectories) provide qualitative confidence in the system's performance.

In conclusion, this work establishes a robust foundation for stereo VO, offering a starting point for the development of more advanced localization systems suited to complex, unstructured, or underwater environments.

Bibliography

[1] Anand George et al. “Visual–Inertial Odometry Using High Flying Altitude Drone Datasets”. In: *Drones* 7.1 (2023), p. 36. DOI: 10.3390/drones7010036.

[2] Ryan M Eustice, Hanumant Singh, and John J Leonard. “Visually augmented navigation for autonomous underwater vehicles”. In: *IEEE Journal of Oceanic Engineering* 33.2 (2006), pp. 103–122.

[3] Andreas Geiger, Philip Lenz, and Raquel Urtasun. “Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[4] Mark Maimone, Yang Cheng, and Larry Matthies. “Two years of visual odometry on the Mars Exploration Rovers”. In: *Journal of Field Robotics* 24.3 (2007), pp. 169–186.

[5] Zhengyou Zhang. “A Flexible New Technique for Camera Calibration”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11 (2000), pp. 1330–1334. DOI: 10.1109/34.888718.

[6] Heiko Hirschmuller. “Accurate and efficient stereo processing by semi-global matching and mutual information”. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*. Vol. 2. IEEE. 2005, pp. 807–814.

[7] Anonymous. *Formation of disparity in a stereo vision system*. https://www.researchgate.net/figure/Formation-of-disparity-in-a-stereo-vision-system_fig1_269877216. Accessed: 2025-06-28.

[8] Ethan Rublee et al. “ORB: An efficient alternative to SIFT or SURF”. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2011, pp. 2564–2571.

[9] David G Lowe. “Distinctive image features from scale-invariant keypoints”. In: *International journal of computer vision* 60.2 (2004), pp. 91–110.

[10] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. “EPnP: An accurate O(n) solution to the PnP problem”. In: *International Journal of Computer Vision*. Vol. 81. 2. Springer, 2009, pp. 155–166.

Acknowledgements

I would like to thank my advisor, Professor Damiano Varagnolo, for his availability and the trust he placed in me and in all the members of the Nautilus project. I feel I have grown through this experience and will always remember the educational and fun moments of this adventure. A special thanks to my family, who have always supported me, and to all the friends I've met along the way, who have brightened my study days with laughter and happy moments.