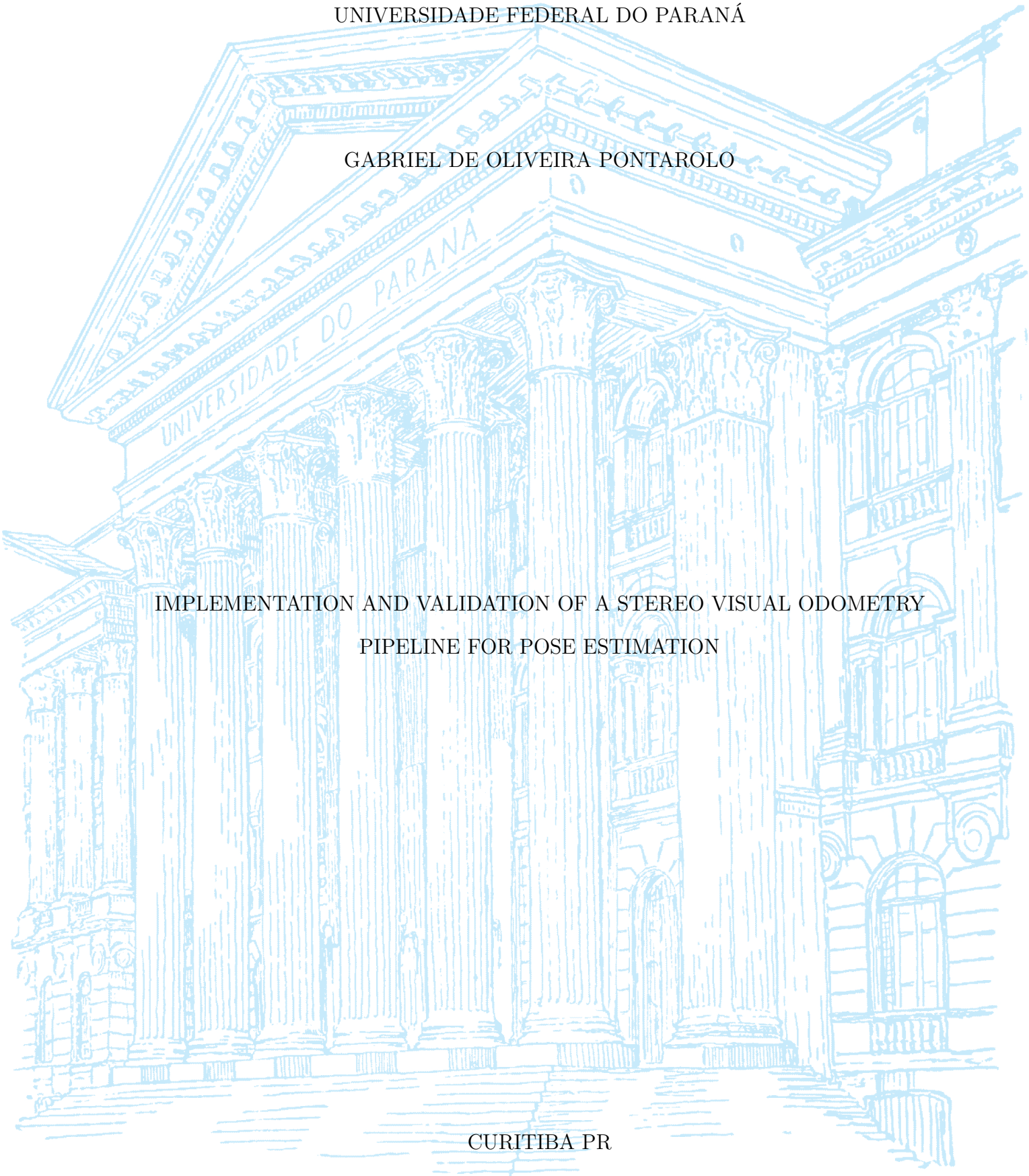
UNIVERSIDADE FEDERAL DO PARANÁ

GABRIEL DE OLIVEIRA PONTAROLO




# IMPLEMENTATION AND VALIDATION OF A STEREO VISUAL ODOMETRY PIPELINE FOR POSE ESTIMATION

CURITIBA PR

2024

GABRIEL DE OLIVEIRA PONTAROLO

# IMPLEMENTATION AND VALIDATION OF A STEREO VISUAL ODOMETRY PIPELINE FOR POSE ESTIMATION

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação.*

Orientador: Eduardo Todt.

CURITIBA PR

2024

*To Deisy and Gelvane Pontarolo.*

## ACKNOWLEDGEMENTS

I want to express my thanks to Professor Eduardo Todt for guiding me as my advisor. To Professor Andrey Pimentel and Master Felipe Bombardelli for composing the examination board. I also express gratitude to my friends and colleagues from the Yapira robotics team, with special mention to Gabriel Hishida for first introducing me to the area of robotics and computer vision. Finally, to all those who contributed, in some way, to the completion of this work.

## RESUMO

Estimação de pose é uma tarefa essencial para sistemas robóticos. Diversos algoritmos estão disponíveis para solucionar este problema, utilizando sensores como GPS, IMUs ou odômetros. A odometria visual (OV) é um desses algoritmos, o qual vem demonstrando ser particularmente vantajoso pois requer apenas um sensor de câmera para efetuar a estimação de pose. Embora seja semelhante à odometria de rodas, uma vez que ambos estimam a pose iterando sobre as alterações na leitura de um sensor geradas pelo movimento de um agente, a odometria visual é resistente a terrenos instáveis e irregulares. Este trabalho começa analisando as técnicas e os algoritmos necessários para realizar esta tarefa e, em seguida, propõe uma *pipeline* de odometria visual *stereo* para a estimação da pose. O objetivo é avaliar o quão bem um método utilizando apenas OV consegue realizar a tarefa de estimar o caminho de um agente em ambientes do mundo real. A implementação foi estruturada no formato de um projeto de ROS2 para facilitar a integração com outros sistemas robóticos. Para fazer a avaliação, foi utilizado o conjunto de *datasets* do KITTI para odometria visual. Os resultados mostram que o método para OV é promissor, alcançando um erro médio de aproximadamente 10% do comprimento total da trajetória na maioria das sequências, embora seja altamente suscetível à acumulação de erro. Como será discutido, ainda há muito espaço para melhorias e utilização de técnicas que podem ser combinadas com a OV para reduzir o erro, como o fechamento de *loop* e outros algoritmos utilizados no SLAM. O código do projeto é aberto e está disponível em `https://github.com/VRI-UFPR/VRI-stereo-vo`.



Palavras-chave: Odometria visual. Estimação de pose. Visão Computacional.

## ABSTRACT

Pose estimation is an essential task for robotic systems. Multiple algorithms are available to tackle this problem, using sensors such as GPS, IMUs, or wheel odometers. Visual Odometry (VO) is one such algorithm that has already been shown to be particularly advantageous because it requires only a camera sensor to perform pose estimation. While being similar to wheel odometry, as both methods estimate pose by iterating over the changes on a sensor readings caused by the agent's movement, visual odometry is robust to slippery and uneven terrains. This work first reviews the techniques and algorithms required to perform this task and then proposes a stereo visual odometry pipeline for pose estimation. The goal is to evaluate how well a VO only method can perform in the task of estimating an agent's path in real world environments. The implementation was structured in a ROS2 project format to facilitate integration with other robotic systems. For evaluation, the KITTI odometry benchmark suite was used. The results show the VO method is promising, achieving an average error of approximately 10% of the length of the trajectory in most sequences, although high susceptible to drift accumulation. As discussed, there is still much room for improvement and techniques that can be combined with VO to reduce drift, such as loop closure and other SLAM-related algorithms. The project is open-source and available on `https://github.com/VRI-UFPR/VRI-stereo-vo`.



Keywords: Visual odometry. Pose estimation. Computer vision.

## LIST OF FIGURES

## LIST OF TABLES

## LIST OF ACRONYMS

| | |
|---|---|
| UFPR | Universidade Federal do Paraná |
| VO | Visual Odometry |
| SLAM | Simultaneous Location and Mapping |
| SIFT | Scale-invariant Feature Transform |
| ORB | Oriented FAST and Rotated BRIEF |
| P3P | Perspective-3-Point |
| SAD | Sum of Absolute Differences |
| BM | Block Matching |
| SGM | Semi-Global Block Matching |
| RANSAC | Random Sample Consensus |
| CUDA | Compute Unified Device Architecture |
| ROS | Robot Operating System |
| FAST | Features from Accelerated Segment Test |
| BRIEF | Binary Robust Independant Features |
| RMSE | Root-Mean-Square Deviation |
| ATE | Absolute Trajectory Error |
| RPE | Relative Pose Error |

## CONTENTS

# 1 INTRODUCTION

In recent years, due to significant investments in applications such as autonomous vehicles, domestic assistant robots, industrial robots, and more, the interest in mobile robotics has grown. On that note, the ability to accurately determine the position and orientation of these autonomous systems - known as pose estimation — has become crucial, as it is a foundational task for navigation, obstacle avoidance, and interaction with the environment, directly impacting their safety and efficiency.

Among the techniques used for pose estimation, visual odometry has emerged as a particularly advantageous method. It refers to estimating the motion of a camera (or a camera-equipped agent) by analyzing a sequence of images captured over time. (Nister et al., 2004) was the first to use this term due to its similarity to wheel odometry, since both estimate the pose by incrementing the changes on a sensor induced by the motion of an agent. However, as noted in (Scaramuzza and Fraundorfer, 2011), wheel odometry is highly affected by wheel slip on uneven terrain. Although visual odometry (VO) is resistant to these scenarios, it has also been able to provide more accurate trajectory estimates compared to wheel odometry, with relative position errors ranging from 0.1 to 2%.

## 1.1 OBJECTIVE

This work first reviews the techniques and algorithms required to perform this task and then proposes a pure stereo visual odometry pipeline for pose estimation. The goal is to evaluate how well a VO only method can perform in the task of estimating an agent's path in real world environments. For integration with other robotic systems, the implementation was structured as a ROS2 project. The widely recognized KITTI odometry benchmark suite, from (Geiger et al., 2012), was used for tests and evaluation. The code is open source and available on `https://github.com/VRI-UFPR/VRI-stereo-vo` under GPL-3 license.

## 1.2 PAPER STRUCTURE

First, chapter 2 will define the problem and elaborate on the fundamental techniques used to accomplish this task. The structure and tools of the Robot Operating System (ROS) meta-operating system will be briefly introduced as well. Chapter 3 explores some early work on visual odometry and the state-of-the-art for both VO and visual Simultaneous Localization and Mapping (SLAM). Chapter 4 presents the proposed visual odometry pipeline implementation in detail, while 5 will discuss the experiments performed on the KITTI dataset. Finally, 6 summarizes and provides a few potential improvements to the pipeline, which can be achieved in future work.

# 2 BACKGROUND

The next sections will present a few essential concepts for visual odometry, such as the pinhole camera model and the basis of epipolar geometry. A formalization of the problem will be presented as well, and subdivided into four core algorithms: feature extraction, feature matching, depth estimation and motion estimation. Lastly, a brief introduction to the ROS meta-operating system and the context of usage in the project was also included.

## 2.1 PINHOLE CAMERA MODEL

Before approaching the topic of visual odometry, it is necessary to lay down a few concepts regarding the camera model, as they provide most of the foundation for the algorithms described in the following sections. The pinhole was one of the earliest mathematical representations of a camera, and it is illustrated by Figure 2.1.

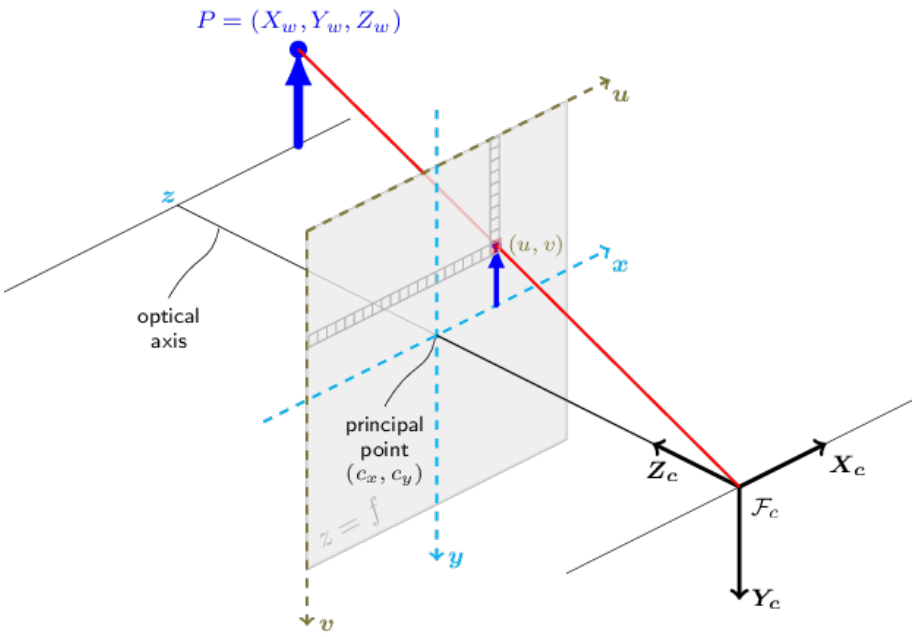



Figure 2.1: A visualization of the pinhole camera model and their respective parameters. A transformation from a 3D object point $P$ to it pixel coordinates $(u, v)$ is represented as well. Source: OpenCV documentation (Bradski, 2000)

Most of the definitions and equations related to this model were compiled by (Bradski, 2000), in the OpenCV library documentation. First of all, the projection equation, which maps a point from the real world to pixel coordinates, is defined as:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \tag{2.1}$$

Where $u, v$ are the pixel coordinates of a point in the image, $s$ is the depth scale, which is lost during the 3D to 2D transformation, and $X_c, Y_c, Z_c$ are the 3D coordinates of this point relative to the camera frame. From the camera matrix $K$: $c_x, c_y$ are the

coordinates of the optical center and $f_x$, $f_y$ is the focal length, both in pixel units. If the pixel aspect ratio is 1:1, then $f = f_x = f_y$.

Since real world lenses are not ideal, the projection to pixel coordinates will always include a certain distortion. A model to describe this distortion has also been proposed, dividing it into a radial and tangential component. Equation 2.1 is then rewritten as:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f_x x'' + c_x \\ f_y y'' + c_y \end{bmatrix} \tag{2.2}$$

Where $x'', y''$ terms represent the distortion, and are modeled as:

$$\begin{bmatrix} x'' \\ y'' \end{bmatrix} = \begin{bmatrix} x' \frac{1+k_1 r^2+k_2 r^4+k_3 r^6}{1+k_4 r^2+k_5 r^4+k_6 r^6} + 2p_1 x' y' + p_2(r^2+2x'^2) \\ y' \frac{1+k_1 r^2+k_2 r^4+k_3 r^6}{1+k_4 r^2+k_5 r^4+k_6 r^6} + p_1(r^2+2y'^2) + 2p_2 x' y' \end{bmatrix} \tag{2.3}$$

with:

$$\begin{aligned} r^2 &= x'^2 + y'^2 \\ \begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} X_c/Z_c \\ Y_c/Z_c \end{bmatrix}, \end{aligned} \tag{2.4}$$

The radial distortion of the lenses is represented by the coefficients $k_1, k_2, k_3, k_4, k_5$ and the tangential distortion by $k_6, p_1, p_2$. Some higher order coefficients, such as thin prim distortion, were omitted from 2.3, as they have minimal effect on the projection. Furthermore, the most common types of radial and tangential distortions can be modeled with high accuracy by only five coefficients: $k_1, k_2, p_1, p_2, k_3$.

The Zhang's camera calibration algorithm (Zhang, 2000) is a popular method to find an estimation for those parameters by observing a planar pattern with at least two orientations, typically a checkerboard. The coefficients can be used to perform a process of image rectification, where the original pixels coordinates are re-mapped to produce an image captured with nearly ideal lens, allowing the use of the original projection equation 2.1.

This method also estimates the camera matrix $K$ and, combining with the distortion coefficients - also known as the distortion vector $D$ - provides the so-called camera intrinsics parameters.

Referring back to 2.1 and considering now an ideal - or calibrated - camera, the formula to retrieve the $X_c$ and $Y_c$ object coordinates can be derived. Supposing $Z_c$ is known:

$$X_c = \left( \frac{u - c_x}{f} \right) \times Z_c \tag{2.5}$$

and

$$Y_c = \left( \frac{v - c_y}{f} \right) \times Z_c \tag{2.6}$$

These concepts will be useful later on when combined to a stereo depth estimator to find $Z_c$.

Finally, consider the matrix $T_{w,c} \in \mathbb{R}^{4\times4}$, which describes the transformation from the camera to the world coordinate frame. In other words, the camera extrinsics:

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = T_{w,c} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{2.7}$$

combining with 2.1:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K\Pi T_{k,c} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{2.8}$$

where $\Pi$ is the projection model to convert to homogenous coordinates. Expanding each term:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{2.9}$$

which is the full projection matrix to map an object point in real-world coordinates - such as meters - to a camera pixel coordinate.

## 2.2 VISUAL ODOMETRY

(Nister et al., 2004) first described it as a motion estimation system that uses a video feed from a moving camera alone as an input, operating in real-time with low latency and no prior knowledge of the scene.

(Scaramuzza and Fraundorfer, 2011) later formulates the problem as, given a set of images $I_{0:n} = \{I_0, \ldots, I_n\}$ taken by a moving agent with a camera system every $k$ time instants, the rigid body transformation $T_{k,k-1} \in \mathbb{R}^{4\times4}$ relates two camera poses at instants $k$ and $k-1$ on the form:

$$T_{k,k-1} = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix} \tag{2.10}$$

where $T_{k,k-1}$ is composed by the rotation matrix $R_{k,k-1} \in SO(3)$ and the translation vector $t_{k,k-1} \in \mathbb{R}^{3\times1}$.

A pose is a combination of an agent's position and orientation in space with respect to a coordinate frame. In other words, how far away and in what direction is the agent pointing relative to an initial location. The agents path can be represented by a set of consecutive poses.

There are a few different ways to represent a pose depending on the application, but the transformation matrix will be used in this case, due to simplicity. Considering $C_0$ the camera pose - in the format of a homogeneous transformation matrix - at instant $k = 0$, then $C_n = C_{n-1}T_{n,n-1}$ is the camera pose at instant $k = n$ relative to $C_0$. The VO problem is to estimate the path $C_{0:n} = \{C_0, \ldots, C_n\}$ of an agent with a camera system.

The core components of the algorithm consist of a feature detector that finds notable points in an image (for instance, corners), a feature matcher that can match these

features between consecutive pairs of images, and a motion estimator that computes the transformation $T_{k,k-1}$ between each pair.

### 2.2.1 Feature detection

(Scaramuzza and Fraundorfer, 2011) divides VO-related works into two categories: appearance-based, which uses the information from all pixels on the image, and feature-based, which uses only notable keypoints. The original work from (Nister et al., 2004) utilizes the latter. Specifically, the Harris corner detector described on (Harris and Stephens, 1988) due to being resistant to small image distortions. The current work will also focus solely on feature-based methods.

(Fraundorfer and Scaramuzza, 2012) a describes a feature as a pattern on an image that differentiates from its neighbors in intensity, color, and texture. Most VO algorithms focus on point detectors since their position can be measured with a higher accuracy. Points are features that can be described as either a corner - an intersection between two edges, where an edge is a change in image brightness - or a blob - a pattern with a distinct texture that is neither an edge nor a corner.

Most feature detection algorithms consist of two steps, applying a feature response function on the entire image (corner response on the Harris detector, for instance) followed by a non-maximum suppression to filter the outliers from the first step. To ensure scale invariance, the first step is applied to both down and up scaled versions of the image, and features that consistently appear across different scales are selected.

After detection, it is necessary to represent the feature with a descriptor that is compact and able to be effectively matched with others. (Fraundorfer and Scaramuzza, 2012) states that the appearance - the intensity of the nearby pixels - is often not a good descriptor, as it is not invariant to changes in orientation, scale, and viewpoint. The scale-invariant feature transform, SIFT, represents the features by dividing the area around it into patches and calculating the local histogram of oriented gradients, which is then concatenated and represented by a vector of 128 elements.

Each feature detection algorithm has advantages under different environments and computational constraints. Two algorithms in particular were evaluated on this work, SIFT and ORB, which will be described in details on chapter 4.

### 2.2.2 Feature matching

The next step consists of matching the features found between two consecutive frames. Typically, this matching follows a distance metric determined by the feature descriptor. (Bradski, 2000) recommends using L2 norm for descriptors similar to SIFT and the Hamming distance for binary string-based descriptors such as ORB.

A straightforward approach to this problem is to compare each feature in the image $I_{k-1}$ with all features in the image $I_k$ and select the closest match. To reduce the search space, constraints can be applied, such as a disparity value, which restricts potential matches to those within a certain pixel distance from the feature in the first image. (Nister et al., 2004) applies this method with a disparity set at 10% of the original image size, although this value may vary depending on the agent's speed and the smoothness of the input stream. To improve the quality of the matches, the author also applies the mutual consistency check by matching features from $I_k$ to $I_{k-1}$ and only accepting matches that mutually have each other as their closest.

### 2.2.3 Motion estimation

Considering two images of a same feature point $X$ taken by cameras at different poses, $\tilde{p} = [\tilde{u}, \tilde{v}, 1]$ are the pixel coordinates of the feature in image $I_k$, and $p$ is the corresponding point on image $I_{k-1}$. The epipolar constraint defines the line where $\tilde{p}$ lies in image $I_k$. For a calibrated camera, this relationship can be expressed as:

$$\tilde{p}^T E p = 0 \tag{2.11}$$

where $E$ is known as the essential matrix. This matrix is obtained by multiplying the 3x3 rotation matrix $R$ with the skew-symetric representation $[t]_\times$ of the 3x1 translation vector from the camera center $C_k$ to $C_{k-1}$:

$$E = R[t]_\times \tag{2.12}$$

Figure 2.2 illustrates this concept. An epipolar plane is defined by at least the camera centers $C_k$, $C_{k-1}$ along with one of the feature points. The epipolar lines are drawn where this plane intersects the image frames.

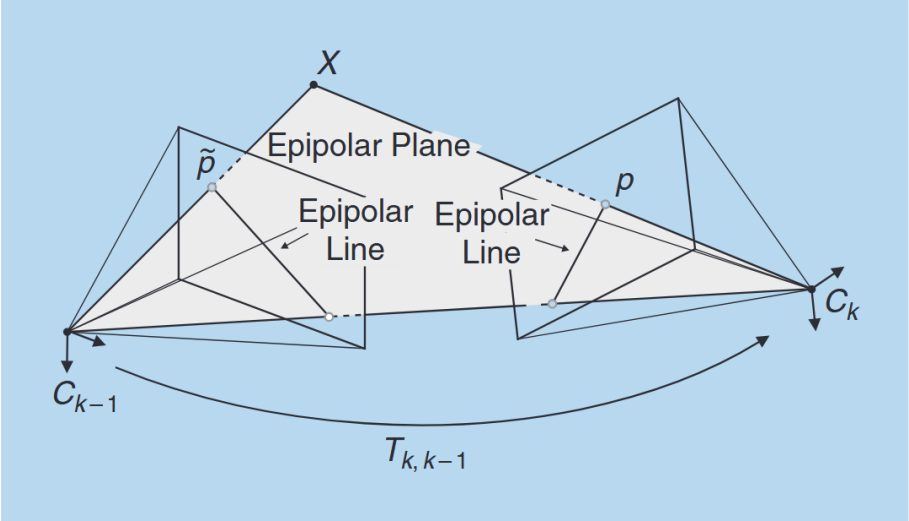



Figure 2.2: Illustration of the epipolar constraint. The feature point $X$, its pixels coordinates on each image $\tilde{p}$, $p$ and camera centers $C_k$, $C_{k-1}$ are all inside the epipolar plane. Source: (Scaramuzza and Fraundorfer, 2011)

In other words, estimating the motion between the two frames requires finding the essential matrix. (Nister, 2004) suggests an implementation of a 5-point algorithm, as a more efficient alternative to the classical Longuet-Higgins' 8-point algorithm. In the same work, Nister also introduces a decomposition of $E$ into the $R$ and a displacement vector $b$ using the triangulation of a single point. It's important to note that the vector $b$ does not directly represent the camera's translation motion, as the scale is not a constraint of the essential matrix. Thus, the movement cannot be expressed with real world units - usually in meters. The next sections will elaborate on how it can be recovered, typically through triangulation of a few points, either within a single video stream a stereo setup.

(Scaramuzza and Fraundorfer, 2011) summarizes this process, known as 2D-to-2D motion from image, in the following algorithm:

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

1: Set initial $C_0$
2: **for all** new frame $I_k$ **do**
3: Extract matches between $I_k$ and $I_{k-1}$
4: Compute essential matrix $E_k$ for image pair $I_k$, $I_{k-1}$
5: Decompose $E_k$ into $R_k$ and $b_k$
6: Compute relative scale $s_k$
7: $t_k := s_k b_k$
8: $C_k := C_{k-1} T_k$
9: **end for**

### 2.2.4 Recovering the scale

A stereo setup consists of a pair of cameras with similar intrinsics, mounted parallel to each other, and within a known baseline distance. The scale can be recovered by triangulating pairs of features that are visible on both images of the stereo pair. Although this process can also be performed using consecutive frames, requiring only a single camera, it's not as efficient and will not be the focus of this work.

Referring back to the projection equation 2.1, the scale this process aims to recovery is the term $s$, which is usually implied by the coordinate $Z_c$. To estimate $Z_c$, consider a pair of images $I_l$ on the left and $I_r$ on the right, taken by a stereo setup of two similar cameras - with the same focal length. The disparity $D$ of a point is defined by:

$$D = x_{I_l} - x_{I_l} = \frac{B f_x}{Z_c} \tag{2.13}$$

Where $x_{I_l}$ and $x_{I_l}$ are the distances between the point and their respective optical centers on each image, $B$ is the baseline of the stereo setup and $f_x$ the focal length. This relation is illustrated in Figure 2.3.

In this scenario, the disparity can be estimated using a stereo correspondence algorithm. A classical example is the block matching algorithm, which works by dividing the image into patches and, along an epipolar line, searching for the corresponding area in the stereo pair using a similarity score such as the sum of absolute differences (SAD). This concept is visually represented in Figure 2.4, where the patches are searched within $D$ pixels of their corresponding coordinates on the right image.

Semi-global block matching (SGM) was later introduced by (Hirschmuller, 2008) and became one of the widely used stereo correspondence algorithms for disparity estimation. It performs the search by optimizing a cost function pathwise in all directions on the image. According to the author, although it is more costly computationally, the disparity map estimated is more accurate when compared to the BM.

Finally, with a known disparity $D$, the $Z_c$ in world coordinates is calculated with:

$$Z_c = \frac{B f_x}{D} \tag{2.14}$$

The remaining two coordinates $X_c, Y_c$ are calculated respectively from 2.5 and 2.6. With the 3D object points, the motion can be estimated with the depth included as a constraint.

Although 3D-3D estimation is possible, and (Scaramuzza and Fraundorfer, 2011) presents an algorithm for this problem, the camera projection properties can be used to perform a more efficient 3D-2D estimation with the Perspective-n-points (PnP) algorithm.

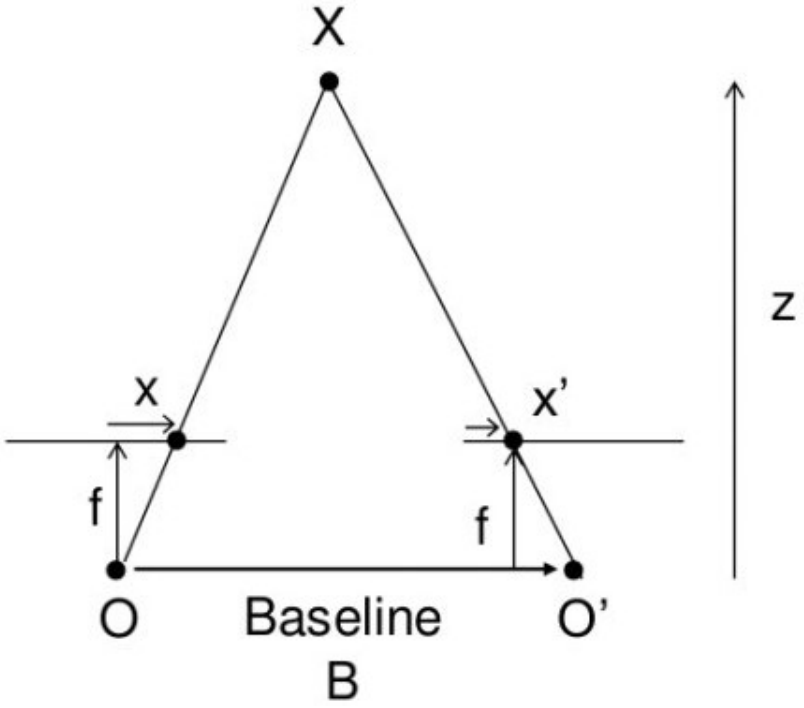



Figure 2.3: Illustration of the correlation between depth and disparity on a stereo setup. In world coordinats, $O$, $O'$ are the camera centers and $X$ is the projected point. For pixel coordinates, $f$ is the focal length and $x$, $x'$ are the distance between the points projection and their camera centers. Source: (Bradski, 2000)

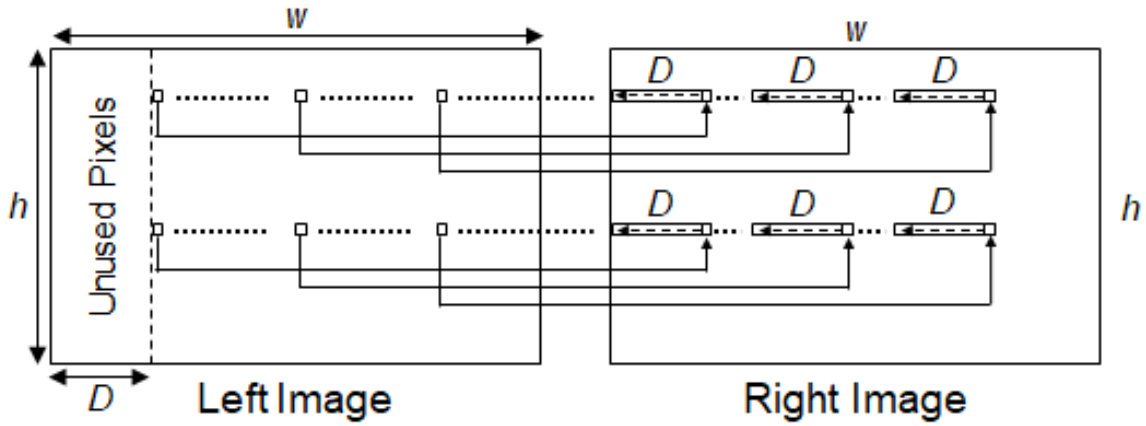



Figure 2.4: Representation of the pixel search in the stereo BM algorithm. The offset $D$ has the lenght of the baseline of the stereo setup in pixel coordinates. Source: MathWorks documentation

### 2.2.5 3D-2D motion estimation

This approach has been proved by (Scaramuzza and Fraundorfer, 2011) to be more accurate and less computationally expensive than a pure 3D-3D method. The former only requires the reprojection error between the object points and image points to be minimized, while the latter uses the 3D Euclidian distance between the points, often requiring a least squares algorithm, such as the Levenberg-Marquard, as a backbone.

Considering the full projection matrix 2.9 - with known camera intrinsics - as well as a set of 3D object points in world coordinates and their corresponding 2D pixel

projections, the remaining extrinsics transformation can be estimated by applying the PnP method. In this case however, the goal is to determine a transformation between two camera poses. Therefore, instead of using world object points and pixel coordinates from the same camera, the set of 3D points in the camera frame of $I_{k-1}$ and the 2D points from $I_k$ will be used, resulting in the transformation $T_{k,k-1}$.

The general intuition is to find a set of $n$ pairs of 3D-2D points where the transformation between them minimizes the reprojection error across all pairs. Several efficient methods exist to determine the transformation matrix from these points, and this work employs the P3P (Perspective-3-Point) algorithm as described by (Ke and Roumeliotis, 2017), which requires four pairs of points to recover the transformation. Usually, the P3P solution is combined with a search algorithm such as the Random Sample Consensus (RANSAC), which aims to find a set of points close enough to the optimal. If the camera matrix $K$ is unknown, at least eight pair of points are required. However, in this case, only four are required, as the camera has already been calibrated.

(Scaramuzza and Fraundorfer, 2011) also presents an algorithm for general 3D-2D visual odometry, although it only considers the monocular case. Algorithm 2 is adapted from his to use stereo depth estimation for scale recovery:

**Algorithm 2** VO from 3D-to-2D correspondences

1: Set initial $C_0$
2: **for all** new stereo pair $I_{l,k}, I_{r,k}$ **do**
3: $D_k = DepthMap(I_{l,k}, I_{r,k})$
4: Extract matches between $I_{l,k}$ and $I_{l,k-1}$
5: Convert the keypoints from $I_{l,k}$ to 3D using $D_k$
6: Compute camera pose $T_{k,k-1}$ with PnP from 3D-to-2D matches
7: $C_k := T_{k,k-1}C_{k-1}$
8: **end for**

## 2.3 ROBOT OPERATING SYSTEM

According to (Macenski et al., 2022), ROS functions as a meta-operating system for robots, offering an environment and a comprehensive set of tools that facilitate the development and integration of robotic applications, such as message exchange between processes, package management, client-server interfaces, and various other useful abstractions.

A basic ROS system consists of two or more processes, also called nodes, running independently, which can send the data to a topic via a publisher structure or receive data from a topic via a subscriber. There's no limit to the number of publishers, subscribers, or topics a node can be connected to or the number of nodes in a ROS network. The data exchanged is structured in ROS messages, similar to C structures or dictionaries. The ROS library includes message types for simple variables - integer, float, string, booleans, and arrays - as well as a few composed messages commonly used in robotics, such as *Point*, *Pose*, *Twist*, *Odometry*, *Image*, etc. Figure 2.5 illustrates this abstraction. Each node within a ROS system can be implemented in a different programming language provided it supports the ROS library to access the message system abstractions, for example, *rclpy* for Python and *rclpy* for C++.

The primary reason ROS was chosen as a wrapper for this work, was because it provides an abstraction layer to work with hardware - or sensors and actuators in general - specifically, the video stream from the cameras. For instance, a camera driver can run

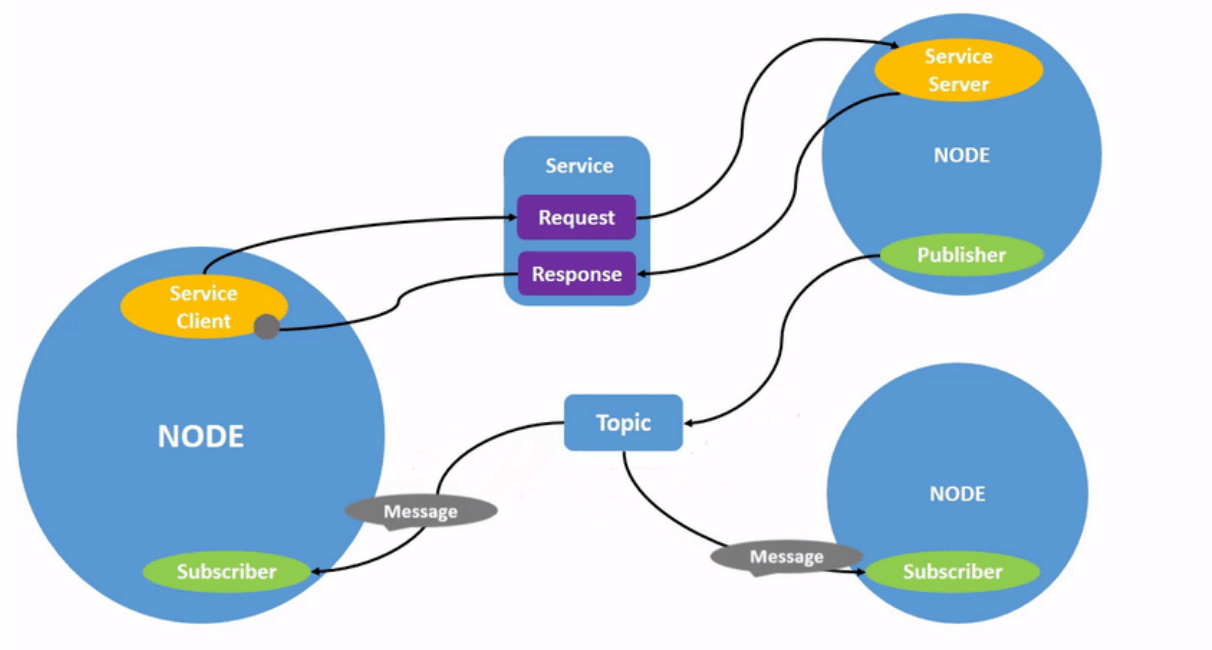



Figure 2.5: Representation of the ROS message exchanging between nodes. The message is transmitted from a publisher to its subscribers via a topic. A client-server model node which will answer requests can be implemented as well. Source: ROS2 documentation (Macenski et al., 2022).

independently from the core VO system as a separate process, or ROS node, while ensuring compatibility through ROS's message interface. Additionally, the framework includes the *rosbag* tool, which enables recording and playback of messages at the original frequencies, facilitating the creation and evaluation of datasets and to simulate real time behavior.

In the other way, ROS provides a well-established interface within the robotics community, allowing seamless communication with other ROS-based applications that may want to utilize the odometry data from this node as a black-box module.

The inter-process communication interface uses TCP/IP, allowing nodes to be visible and send messages across different devices in the same local network. This provides a significant advantage for autonomous systems distributed across various embedded computers.

Two main iterations of the meta-operating system are currently being maintained: ROS and ROS2. Each of them are subdivided into versions, known as distros. This work was implemented using the latter, ROS2, and the *Humble Hawksbill* distro, one of the current long-term support (LTS) releases.

## 2.4 CONCLUSION

This chapter has presented a base algorithm for monocular and stereo visual odometry, as well as the sub-routines required by them: feature extraction and matching, depth estimation with block matching, and motion estimation using PnP. An introduction to the structure and operation of a ROS system was also presented. As indicated previously, the focus of this work will be stereo visual odometry and 3D-2D pose estimation.

Before entering in details about the implementation, a few visual odometry related works will be explored in the following chapter.

# 3 RELATED WORKS

This chapter first explores one of the classical works which focused on a pure visual odometry method similar to Nister's approach. Then, some notable recent works are described, each of them exploring how VO can be combined with other computer vision and navigation techniques to further improve accuracy and performance.

## 3.1 VISUAL ODOMETRY

(Howard, 2008) was one of the first classical works to follow Nister's approach by presenting a real-time stereo-visual odometry algorithm for ground robots. The authors tested using an autonomous four-legged, as well as a steered ground robot, and they were able to keep the average error around 1 meter throughout a 400-meter trajectory with a nearly pure VO approach. Figure 3.1 shows the terrain and trajectories obtained. However, it is worth mentioning that their module required a pre-processing of the rectified frames, depth map, and correlation instead of estimating these in real time, as accomplished in this work. They also differ by using a 3D-to-3D motion estimation based on the Levenberg-Marquard least squares algorithm, which allows running an inlier filtering step using a graph clique but, as mentioned previously, is more costly to compute and less accurate than 3D-2D estimation.

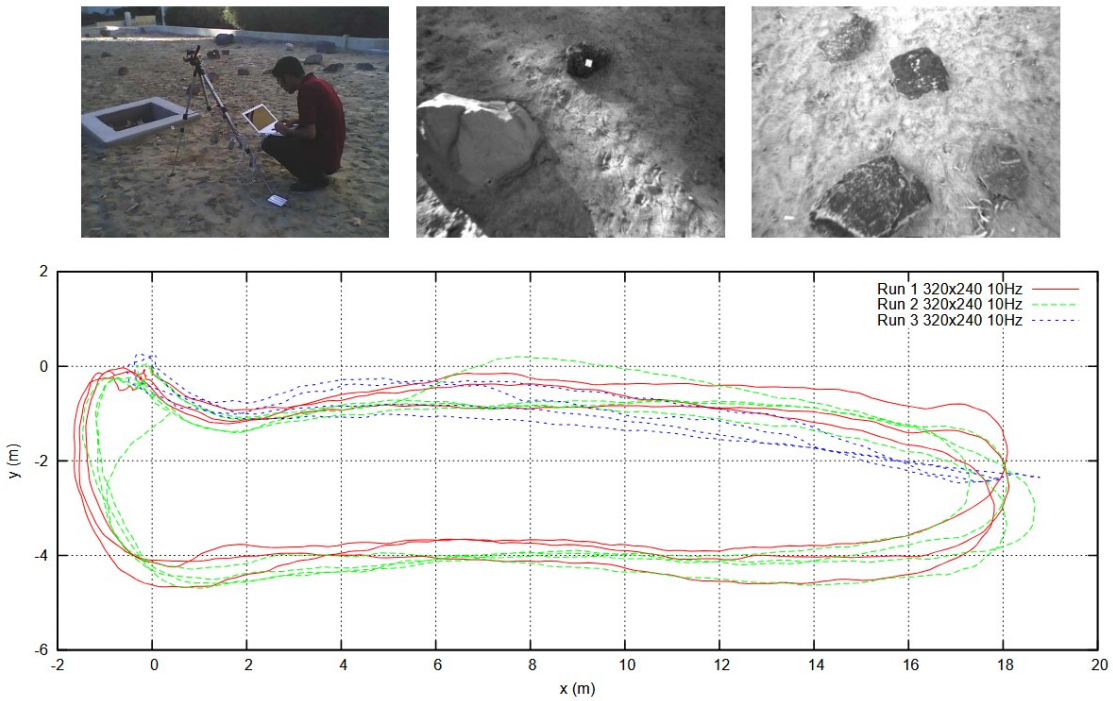



Figure 3.1: Tests made with the four-legged robot on (Howard, 2008). The three images at the top show the scenario mounted for evaluation and the plot contains the paths estimated by the visual odometry.

### 3.1.1 State-of-the-art

As a method that only increments estimations over time, a pure visual odometry solution would be highly susceptible to drift accumulation. Most practical modern solutions employ techniques such as bundle adjustment, loop closures and also combine different sensors with the VO estimation - using a Kalman Filter for instance - to reduce the drift. These methods start to fall under the category of Visual SLAM. Particularly, algorithms that incorporate the fusion of VO and Inertial Measurement Units (IMUs) can be classified as Visual-Inertial Odometry (VIO) approaches. The widely used (Qin et al., 2018) VINS-Mono, and the later extension VINS-Fusion, utilized both techniques mentioned - monocular or stereo camera with IMU - achieving top-ranked open source algorithm on the (Geiger et al., 2012) KITTI benchmark suite, with 1.03% of translational error on 2019 (30th lowest, currently is placed 58th). Figure 3.2 illustrates their pipeline.

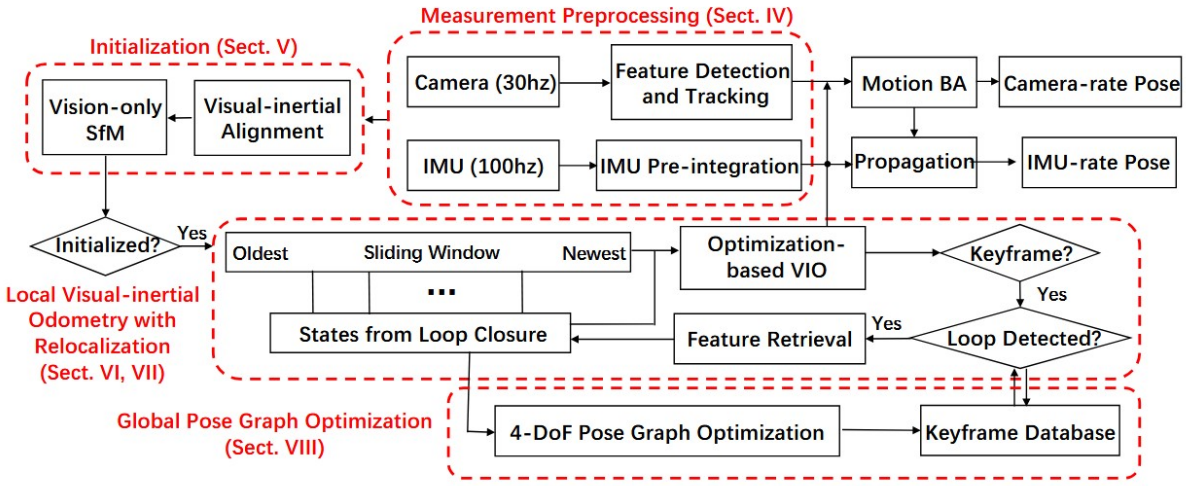



Figure 3.2: Block diagram of the full pose estimation pipeline on VINS-MONO (Qin et al., 2018). It's possible to see the camera and IMU fusion, as well as the use of visual SLAM techniques.

Other notable open source mentions include the (Campos et al., 2021) ORB3-SLAM, which introduces VIO with a multi-map system and is the first to perform loop closure with short, mid, and long-term data association. Aditionaly, the Kimera library (Rosinol et al., 2020), combines VIO with pose graph estimation and other computer vision tasks, such as mesh reconstruction and semantic labeling.

A large portion of the current state-of-the-art research focuses on integrating machine learning methods into the VO pipeline for tasks such as monocular depth estimation or feature matching between frames. D3VO, from (Yang et al., 2020), trains three deep neural networks to estimate depth, pose directly from the consecutive frames - without extracting and matching features - and an uncertainty map, using only a monocular camera. A diagram of the pipeline and their neural networks is shown in Figure 3.3. The authors were able to achieve 0.88% translational error on the KITTI benchmark, currently ranking 43th.

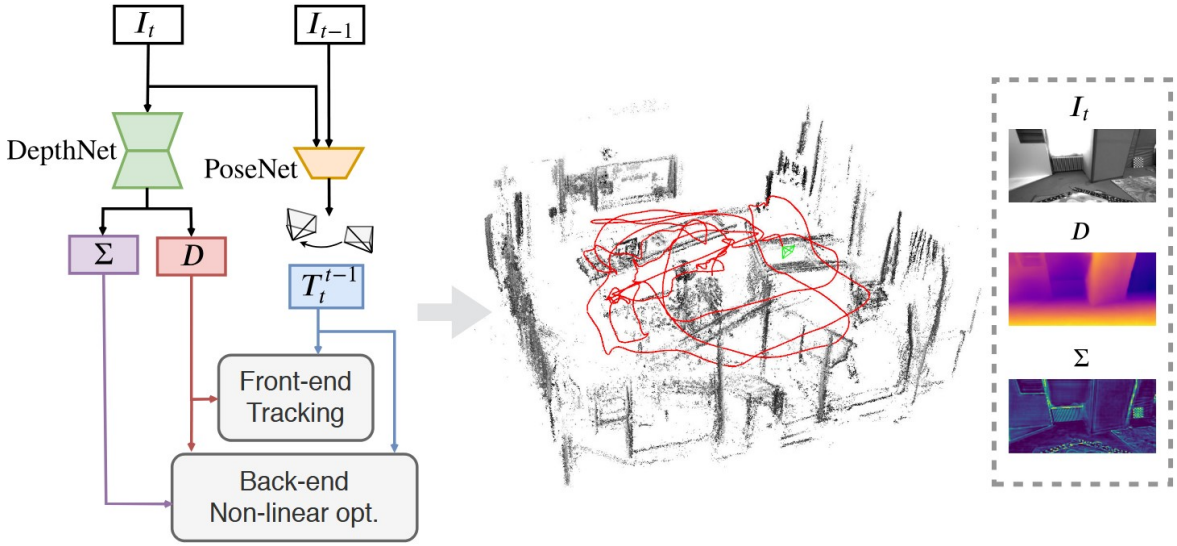



Figure 3.3: On the right, diagram of the pipeline and neural networks used on D3VO (Yang et al., 2020). The image on the center is the trajectory estimated on the EuRoC MAV dataset and, on the left, the depth as well as the uncertainty maps.

One of the main advantages of these methods is the flexibility in adverse scenarios. (Zhan et al., 2020) states that most geometry-based algorithms, although able to reach a high accuracy, are designed for specific conditions known beforehand, such as distance, lighting, and number of features in the environment. They also explore how to properly integrate deep learning with the classical epipolar geometry methods and propose two convolutional networks to estimate depth and optical flow - in other words, the motion - between images.

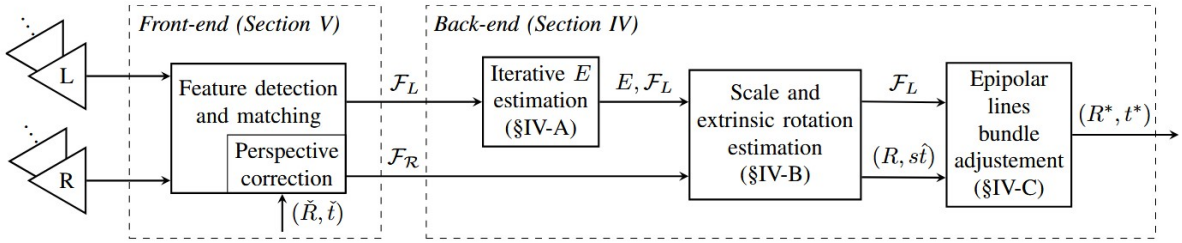



Figure 3.4: Stereo visual odometry pipeline proposed on SOFT2 (Cvišić et al., 2023). The main novelty presented by the authors is the bundle adjusted based on epipolar lines.

Finally, the current top algorithm for the KITTI dataset is still a pure geometry-based, stereo VO method. SOFT2 (Cvišić et al., 2023) reaches 0.53% translational and 0.0009 deg/m rotational errors by exploring epipolar as well as kinematic constraints to improve stereo depth estimation and bundle adjustment, showing that geometry-based methods are still worth researching. Their pipeline is illustrated in Figure 3.4.

## 3.2 CONCLUSION

The first module presented, from (Howard, 2008), focuses primarily on the foundational visual odometry approach, aligning more closely the implementation proposed in this work.

The following authors start to tackle into the area known as visual SLAM. Some examples include VINS-MONO (Qin et al., 2018) which utilizes sensor fusion and various SLAM techniques, as well as ORB3-SLAM which tries to further explore loop closures. Deep learning is also becoming prominent for this area. D3VO (Yang et al., 2020) proposes a pipeline of neural networks to perform pose estimation from a monocular camera image stream.

The next chapter will present in detail implementation proposed for a real-time pure stereo visual odometry pipeline.

# 4 PROPOSED IMPLEMENTATION

This chapter presents the implementation of a real-time stereo-visual odometry pipeline for pose estimation using the aforementioned techniques. As shown in chapter 2, the pipeline can be separated into three fundamental steps: feature extraction and matching, depth estimation, and motion estimation. An adjacent ROS node, or *rosbag*, as a data source, and a metrics node to compare the estimation with the ground truth are present as well.

The largest portion of the project was written in C++, so OpenCV (Bradski, 2000) was utilized for most image-related operations, and the Eigen library for matrix arithmetic. The following sections will describe each module in detail. Figure 4.1 presents a visual representation of the general pipeline architecture as well. Each blue square is a standalone process, or ROS node, and their ROS topics are represented by the purple squares. The internal modules and state variables of each node are illustrated, respectively, in green and orange.

## 4.1 DATA INPUT

The image source can be either another ROS node that will act as a driver for a camera device and publish the frames, or a recorded dataset in the format of a *rosbag*. Each frame from the stereo pair is received at the main ROS node from a separated *Image* message topic.

### 4.1.1 Stereo pair synchronization

Since the frames are received from separated topics, to ensure that both frames correspond originally to the same stereo pair, an *Approximate Time Synchronizer* was used. This ROS data structure subscribes to two or more topics simultaneously, only accepting a set of messages if it receives data from all topics and the difference between their timestamps are within a specified threshold. This value must be smaller than the image period to avoid mismatching. In this implementation, the synchronizer subscribes to both image topics and accepts a threshold of 0.02 seconds.

It's also necessary to provide the calibration data for both cameras in the format of the camera matrix $K$, distortion coefficients array $D$, and the baseline in meters $b$ of the stereo mounting. After receiving the stereo pair, the pipeline converts the image to grayscale, rectify them if required, and simultaneously makes an asynchronous call to the depth and feature modules. Since they do not have any dependencies between each other, they can run in parallel.

## 4.2 DEPTH ESTIMATOR

Mostly, this module consists of a wrapper for the stereo-matching functions provided by OpenCV. It receives a stereo pair as input, processes it using one of the available algorithms, and returns a disparity map for each pixel in the overlapping area of the images. The motion estimator will later calculate the actual depth in meters from the disparity and camera parameters.

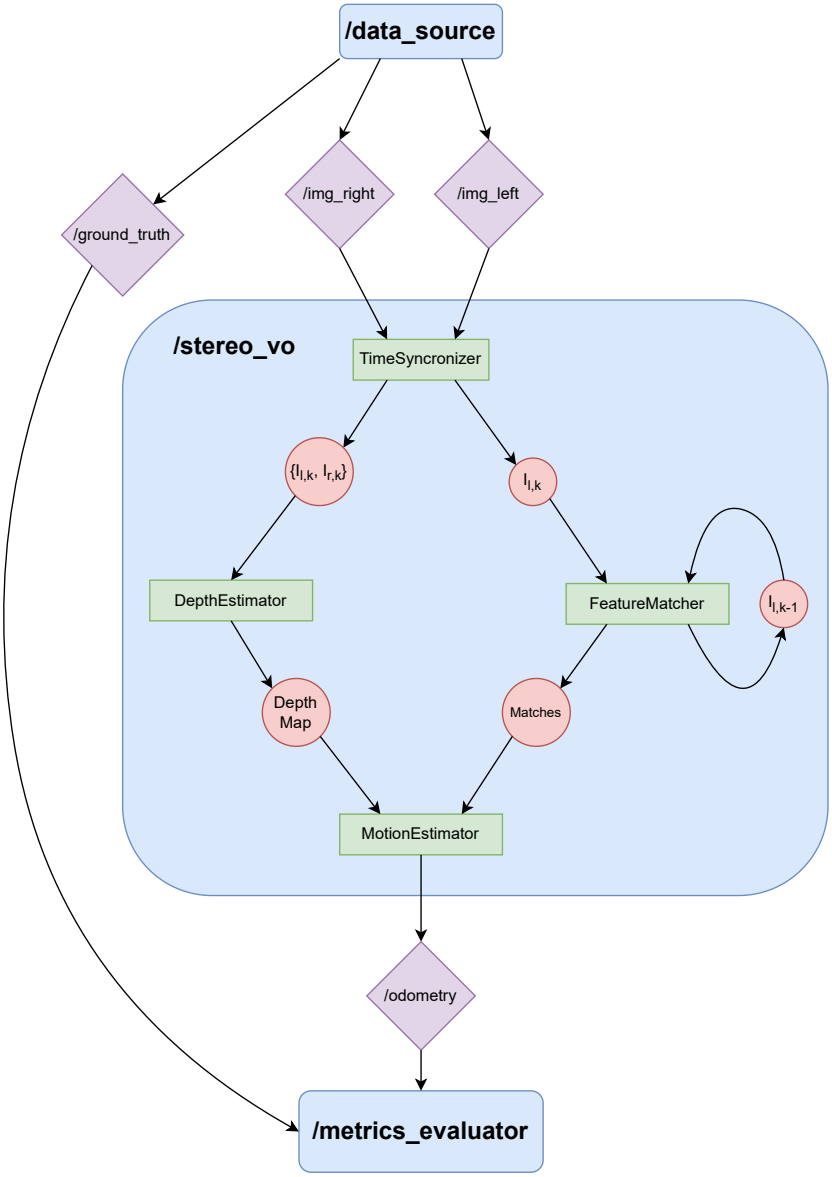



Figure 4.1: General architecture of the proposed visual odometry pipeline. The ROS nodes are represented in blue, the ROS topics in purple, the internal modules in green and state variables in orange.

As discussed in the chapter 2, Semi-Global Block Matching (SGM) still remains one of the most accurate algorithms for this task and had the best results in our experiments with the KITTI dataset (Geiger et al., 2012). The two key parameters are the level of disparity, which controls within how many pixels the algorithm will search a point from the left image in the right image, and the pixel block size, which sets the size of the sliding window used for matching. Considering the image size of 1382x512 from the dataset, the disparity is set to 96 and the block size to 11. These values were found empirically and they also must meet certain constraints: the disparity should be divisible by 16, and the block size must be an odd number. Figure 4.2 displays a disparity map estimated using SGBM with the parameters described earlier. The purple rectangle on the left has exactly

the width of the baseline of the camera, representing the area where the algorithm has no matching information, as it's outside of the common region of the stereo pair.

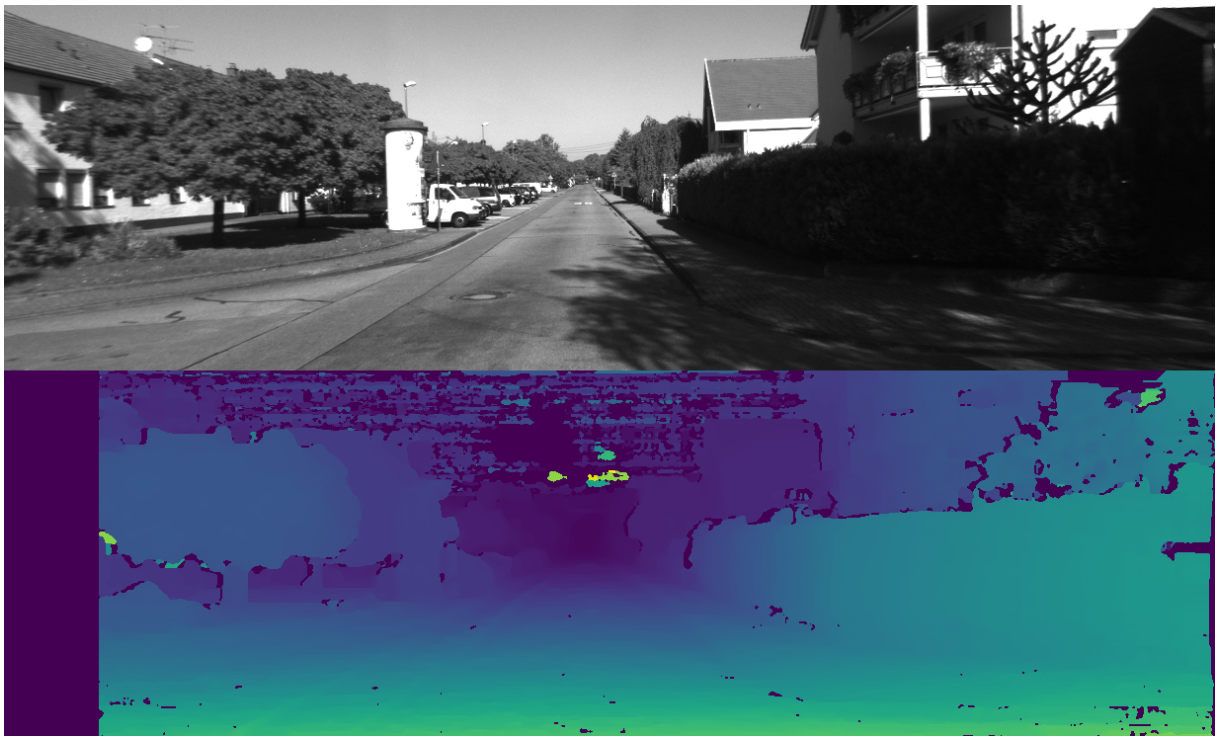


Figure 4.2: Right image of a stereo pair extracted from the KITTI dataset (Geiger et al., 2012), and their respective disparity map estimated using SGBM.

A few other algorithms are also available to test. The aforementioned Block Matching (BM), as well as its CUDA-based implementation, which presents a significant speed-up at the cost of accuracy. The module also includes implementations of the Stereo Belief Propagation (Felzenszwalb and Huttenlocher, 2006) and the Constant Space Belief Propagation. However, both algorithms are implemented exclusively in CUDA, requiring a GPU, and, on the hardware used for evaluation, their computation time was too large (close to 1 second) to be used in real time.

## 4.3 FEATURE EXTRACTOR AND MATCHER

Both processes of extracting and matching features between consecutive frames are implemented on this module. The input contains the current frame from the left camera, along with the previous frame's descriptors - or none in the case of the first iteration. The return includes the descriptors of the current frame and an array containing the best matches found. Both ORB and SIFT algorithms are available in the implementation, offering a trade-off between speed and accuracy.

### 4.3.1 Scale-Invariant Feature Transform

The Scale-Invariant Feature Transform (SIFT), described by (Lowe, 2004), identifies keypoints using a Difference of Gaussians (DoG) approach. First, different magnitudes of Gaussian blurs are applied to the image and the adjacent ones are subtracted from each other. The results of these subtractions are stacked on top of each other and features that

stand out across the three axis - x,y and blur level - are selected. As the name suggests, to grant scale-invariance this process is repeated across different scales in an image pyramid, and points that are inconsistent across them are filtered.

Once the keypoints are identified, the area around each one is divided into a 4x4 grid, and further into another 4x4 subdivision. The gradient for each sub-region is computed and an histogram of these gradients is calculated. Each of the 16 histograms are then discretized into eight bins of 45° intervals, resulting in a 128-bit descriptor per keypoint. Gradients rather than, for example, the pixel intensity were chosen because they offer robustness against changes in illumination, scale and viewpoint.

For matching, a brute-force search can be used. For large enough data, a KD-tree can offer speed improvements, but, during experiments on KITTI (Geiger et al., 2012), the performance was very similar or even worse than the brute-force matcher.

### 4.3.2 Oriented FAST and Rotated BRIEF

This algorithm was first described on (Rublee et al., 2011) as an open-source alternative to other popular alternatives such as the aforementioned SIFT and SURF. Initially, it detects key points using the FAST extractor in an image pyramid, to produce features on multiple scales, and filters the best corners using the Harris measure (Harris and Stephens, 1988). To grant rotation invariance, it also computes the orientation from the vector between the centroid of the feature and the patch at the center of the image.

BRIEF is used as an image descriptor, but the matrix containing the binary values is rotated according to the orientation vector calculated in the previous step. The angles are discretized, and a lookup table with the corresponding rotation matrices is implemented to speed up this process.

Since BRIEF is a binary descriptor, the matching between the frame's feature points is made using a Locally Sensitive Hashing (LSH) described on (Gionis et al., 2000). This technique separates the points into buckets and uses a hash function to search the buckets. Given a point to query, its descriptor is compared to the common signature bits of each bucket using the Hamming distance as a metric.

To improve accuracy in both cases, the mutual consistency check described on (Nister et al., 2004) is applied. Additionally, as recommended by (Bradski, 2000), Lowe's ratio test is used to reject matches where the distance between the best and second-best pairs is not sufficiently distinct. For most cases, the recommended value is 0.7, meaning that the closest match's distance must be at least 70% of the next closest match. Since SIFT usually identifies a much larger number of keypoints, this value was lowered to 0.2 when running with this algorithm.

A CUDA implementation of ORB is also available, which can offer an speed-up of nearly 10x when compared to SIFT. The trade-off is in the number of features, which still yield low reprojection error, but are much fewer, being occasionally not enough to run the P3P algorithm for motion estimation. Although the computation time of SIFT slightly higher than limit of the dataset to keep up with the real time - around 7Hz - it displayed a better result in general. This might not hold true depending on the camera scene or with even higher camera frequencies.

Figure 4.3 compares ORB and SIFT feature extractors at different Lowe's ratio thresholds. In all three cases, the mutual consistency check was also applied. For matching, ORB used LSH and SIFT a brute-force matcher. Table 4.1 presents the computation speed, in milliseconds, of each step for the same three experiments. The Post-Filtering

step includes both the Lowe's ratio test and mutual consistence check. The C++, OpenCV (Bradski, 2000) implementation was used for these tests.

| | Extraction | Matching | Post-filtering | Total |
|---|---|---|---|---|
| `ORB, 0.7` | 21 ms | 2 ms | 1 ms | 24 ms |
| `SIFT, 0.7` | 139 ms | 32 ms | 8 ms | 179 ms |
| `SIFT, 0.2` | 141 ms | 29 ms | 1 ms | 170 ms |

Table 4.1: Comparison between ORB and SIFT computational speeds for feature extraction, matching and filtering with mutual consistency and Lowe's test. The first column indicates the algorithm and the Lowe's ratio used.

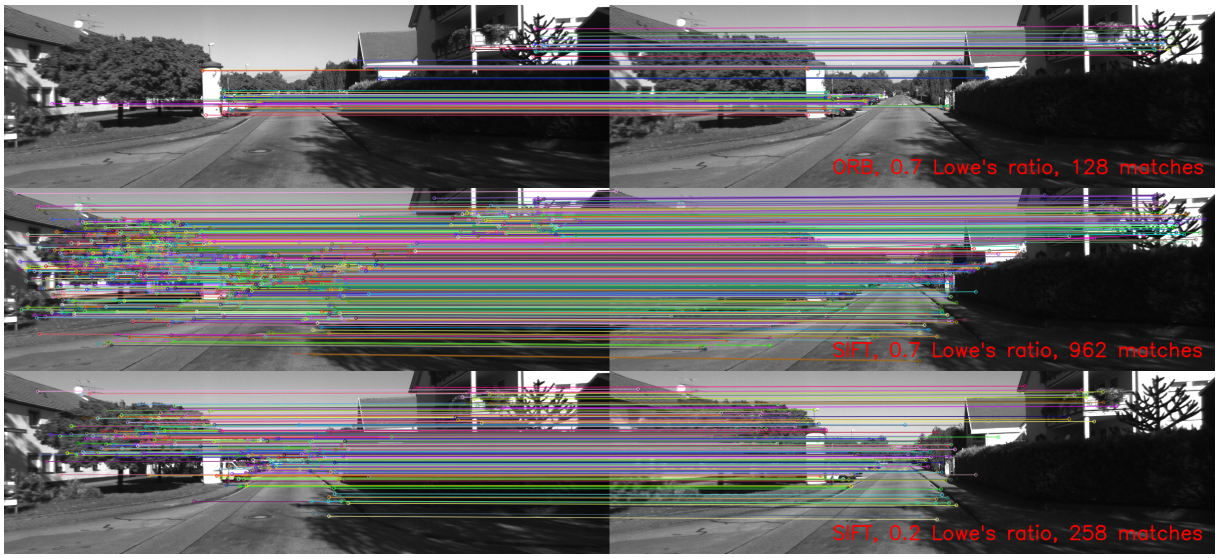



Figure 4.3: Comparison between ORB and SIFT matches number and quality after filtering with mutual consistency and Lowe's test. The red text on each image indicates the algorithm, the Lowe's ratio used and the number of matches.

## 4.4 MOTION ESTIMATOR

Using the depth map and equations 2.14, 2.5 and 2.6, the 2D points from $I_{k-1}$ can be projected to their 3D object points at the camera frame coordinates. With this set of 3D points and their respective 2D matches on $I_k$, the PnP is used to calculate the transformation that minimizes the reprojection error.

OpenCV has an implementation of the P3P solver from (Ke and Roumeliotis, 2017) mentioned previously, which already includes a RANSAC filter. The output from the function is actually the translation, in the same units as the object points, and rotation vector, composed by the $x, y, z$ Euler angles. This vector can be converted into a rotation matrix by using the Rodrigues formula:

Considering $r = [r_x, r_y, r_z]$ the rotation vector, we first normalize $r$ with:

$$\begin{aligned} \theta &= norm(r) \\ r &= r/\theta \end{aligned} \tag{4.1}$$

Then, the rotation matrix $R$ is obtained with:

$$R = \cos(\theta)I + (1 - \cos(\theta))rr^T + \sin(\theta)\begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix} \tag{4.2}$$

Referring back to the original description of the visual odometry problem, the camera pose $C_k = [X_{C_k}, Y_{C_k}, Z_{C_k}, 1]^T$ is given by concatenating the previous pose $C_{k-1}$ with the estimated homogeneous transformation matrix $T_{k,k-1}$. Starting from a known initial position $C_0$, repeating this process iteratively across all the frames from the camera feed allows to reconstruct the agent's path.

After each iteration, the current transformation matrix is converted into a ROS *Odometry* message, containing the 3D orientation, position and covariance, and published, together with the feature and depth maps for debuging. The descriptors and keypoints of the current frame $I_k$ are stored, and the node waits to receive the next stereo pair to repeat this process.

## 4.5 CONCLUSION

The pipeline presented on this chapter can be summarized as: the main ROS node receives the image stream from the data source and calls the depth and feature modules. The first estimates the disparity map - with BM or SGBM - and the second performs feature extraction and matching - with SIFT or ORB. Then, the 2D feature points of the previous image are transformed to 3D object points using the depth map, and PnP is applied to calculate the transformation between the cameras. This transformation is then concatenated with the previous one and the final pose is derived and published as a ROS *Odometry* message.

Using this implementation, the experiments and results obtained on the KITTI dataset (Geiger et al., 2012), will be presented on the following chapter.

# 5 EXPERIMENTS

The primary datasets used for the experiments were the sequences 00 to 10 of the KITTI odometry benchmark suite from (Geiger et al., 2012). The authors initial goal was to create a challenging benchmark that could simulate real world environments for computer vision tasks such as stereo, optical flow, visual odometry, visual SLAM and 3D object detection. Figure 5.1 shows a top view of their recording platform, equipped with high resolution stereo cameras, a laser scanner and a GPS/IMU positioning system. The data were captured on the city of Karlsruhe, Germany, and includes mid-size residential areas, rural areas and highway scenes.

For visual odometry evaluation, 21 sequences containing RGB and grayscale stereo pairs are provided. Ground truth transformation matrices are available for the first eleven, which are meant to be used for training. The remaining ten are intended for evaluation and are the benchmark the authors use on their official ranking, available on `https://www.cvlibs.net/datasets/kitti/eval_odometry.php`.

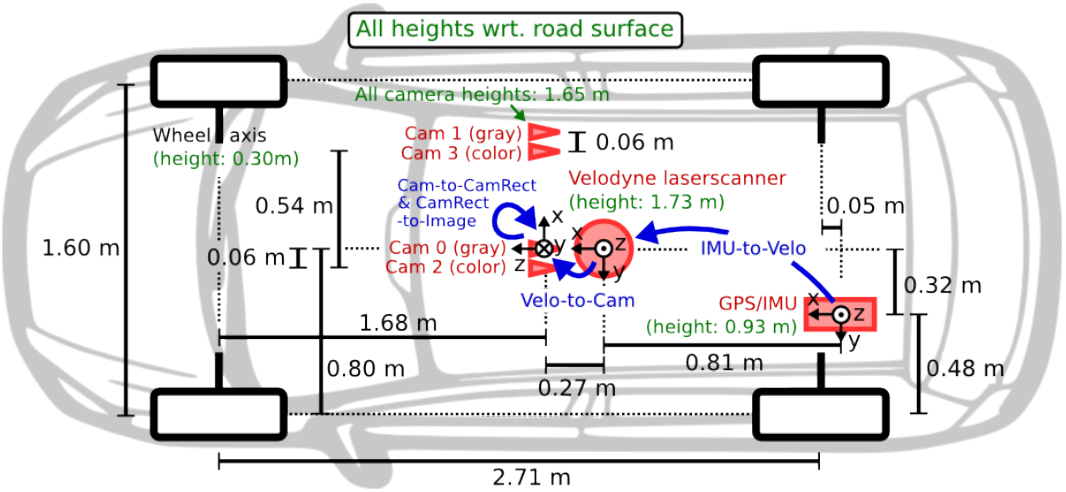



Figure 5.1: Top view of the dimensions and positioning of the sensor setup used on a car to capture the KITTI dataset. Source: (Geiger et al., 2012).

The authors also provide the camera parameters and extrinsics of the mounting, where it is possible to extract the exact baseline of the stereo cameras, as shown in Equation 5.1. The distortion coefficients are not necessary, since the dataset images are rectified.

$$\textbf{baseline} = 0.537$$
$$K = \begin{bmatrix} 718.856 & 0.0 & 607.1928 \\ 0.0 & 718.856 & 185.2157 \\ 0.0 & 0.0 & 1 \end{bmatrix} \tag{5.1}$$

(Zhan et al., 2020) developed and made available a toolbox to evaluate a VO pipeline in the KITTI sequences using two key metrics:

- **Absolute Trajectory Error (ATE)** evaluates the global consistency of the trajectory by comparing each estimated trajectory point with its ground truth.

Considering $p_k = [x_k, y_k, z_k]$ is the ground truth position and $\hat{p}_k$ is the estimated position, ATE can be calculated using the root-mean-squared error (RMSE):

$$ATE_{RMSE} = \sqrt{\frac{1}{N}\sum_{k=0}^{N} \|p_k - \hat{p}_k\|^2} \tag{5.2}$$

- **Relative Position Error (RPE)** evaluates the local consistency of the estimation by comparing the difference between consecutive estimated poses with the ground truth. Considering $t_{k,k+1}$ the translation from position $k$ to $k+1$, RPE can be calculated by:

$$RPE_{trans,RMSE} = \sqrt{\frac{1}{N}\sum_{k=0}^{N} \|t_{k,k+1} - \hat{t}_{k,k+1}\|^2} \tag{5.3}$$

And, considering $R_{k,k+1}$ the rotation matrix from $k$ to $k+1$, the RPE for the orientation can be calculated by:

$$RPE_{rot,RMSE} = \sqrt{\frac{1}{N}\sum_{k=0}^{N} \arccos^2\left(\frac{\text{trace}(R_{k,k+1}\hat{R}_{k,k+1}^{\top}) - 1}{2}\right)} \tag{5.4}$$

To simulate a real-time environment, a *rosbag* was used to replay the dataset at its original frequency - 10hz. If a new frame arrived while the pipeline was still processing the previous pose, the new frame was dropped. Besides, only the video streams from *Cam 0 (gray)* and *Cam 1 (gray)* were used as input to the pipeline. The parameters are summarized in Table 5.1.

| | |
|---|---|
| Depth algorithm | Stereo SGM |
| Number of disparities | 96 |
| Block size | 11 |
| Feature extractor | SIFT |
| Feature matcher | Brute force matcher |
| Lowe's distance ratio | 0.3 |
| PnP method | P3P |
| PnP iterations | 200 |
| Reprojection threshold | 8 |

Table 5.1: Parameters for the visual odometry pipeline used on the KITTI dataset tests.

## 5.1 RESULTS

A CPU with 6 cores and 2.9 GHz was used to run the experiments. The results, evaluated using the metrics described above, are shown in Table 5.2. The path length and the error percentage - ATE over the path length - were also calculated.

| | Path Lenght (m) | ATE (m) | RPE (m) | RPE (deg) | ATE (%) |
|---|---|---|---|---|---|
| 00 | 4000 | 156.89 | 9.79 | 1.53 | 3.92 |
| 01 | 2500 | 1181.69 | 364.45 | 17.74 | 47.26 |
| 02 | 5500 | 363.90 | 21.75 | 1.06 | 6.61 |
| 03 | 635 | 101.60 | 1.37 | 0.48 | 16.0 |
| 04 | 395 | 96.59 | 2.01 | 0.29 | 24.45 |
| 05 | 2340 | 127.58 | 1.47 | 0.55 | 5.45 |
| 06 | 1260 | 119.68 | 3.95 | 0.42 | 9.49 |
| 07 | 755 | 92.97 | 8.86 | 4.19 | 12.31 |
| 08 | 3435 | 320.14 | 1.05 | 0.47 | 9.31 |
| 09 | 1845 | 184.41 | 2.20 | 0.65 | 9.99 |
| 10 | 1040 | 131.69 | 5.69 | 8.96 | 12.66 |

Table 5.2: Average errors in the first eleven sequences of the KITTI dataset. The path length and ATE percentage were also calculated and included for comparison.

The average percentage error across all sequences was 14.31%. Excluding sequence 01, which represents a more challenging dataset, as will be demonstrated in the subsequent sections, the average error decreases to 10.91%. The execution time of each module was also measured and is present in Table 5.3. Since depth and feature modules run in parallel and the motion estimation takes less than 2 milliseconds on average, the total pose estimation time is primarily bounded by the most time-consuming operation, in this case, the feature matching process.

| | Depth Estm. | Feature Match. | Total Pose Estm. |
|---|---|---|---|
| 00 | 41 | 126 | 127 |
| 01 | 40 | 92 | 94 |
| 02 | 40 | 160 | 161 |
| 03 | 40 | 161 | 163 |
| 04 | 38 | 134 | 136 |
| 05 | 39 | 130 | 132 |
| 06 | 39 | 124 | 127 |
| 07 | 38 | 129 | 130 |
| 08 | 40 | 146 | 148 |
| 09 | 39 | 135 | 137 |
| 10 | 39 | 138 | 139 |

Table 5.3: Average computation time for the feature matching, deep estimation and the total pose estimation. Both depth and feature modules run in parallel.

Considering the average estimation time is 140 milliseconds, the pipeline can run at around 7hz. In other words, 3 out of 10 frames per second were dropped on average.

A common behavior observed across all datasets, and in pure odometry systems in general, is the drift over time of the estimated position, relative to the ground truth. This is caused accumulating smaller errors from each step, such as mismatches of a few pixels on the reprojection matrix. Using the result from sequence 00 as an example, Figure 5.2 illustrates the estimated path compared to the ground truth.

Initially, the general shape of the paths matches, with significant overlap near the start of the track. By the end, the estimation has accumulated nearly 100 meters of

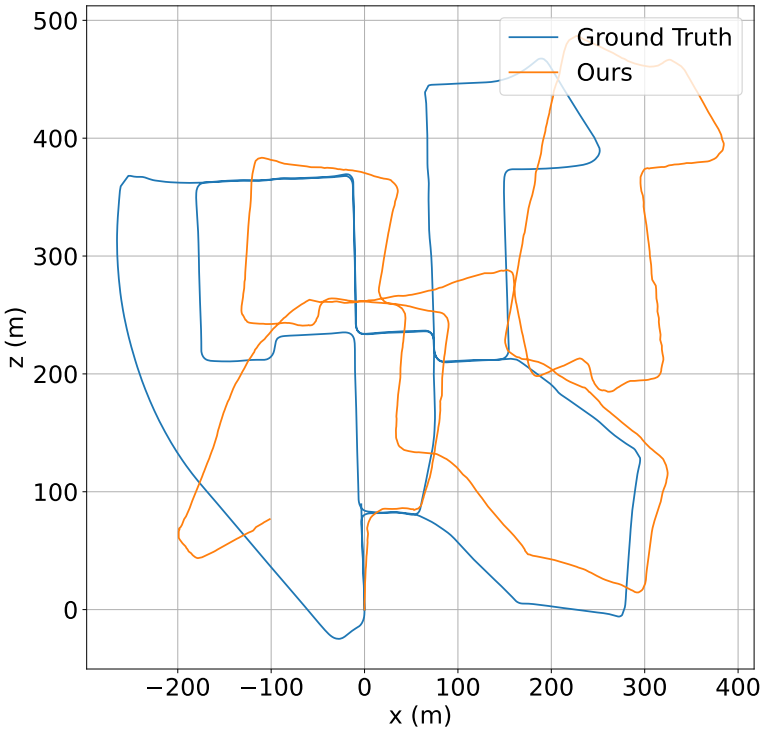



Figure 5.2: Comparison between the estimated path and the ground truth on sequence 00 of KITTI odometry dataset. Close to the third turn, the drift accumulation becomes visible and the error starts to increase until around 100 meters.

drift. Despite this, the RPE remains low, indicating good local consistency of the poses compared to the ground truth. Sequence 08, shown in Figure 5.3, presents a more extreme example of this behavior, where the final pose diverged around 300 meters from the ground truth.

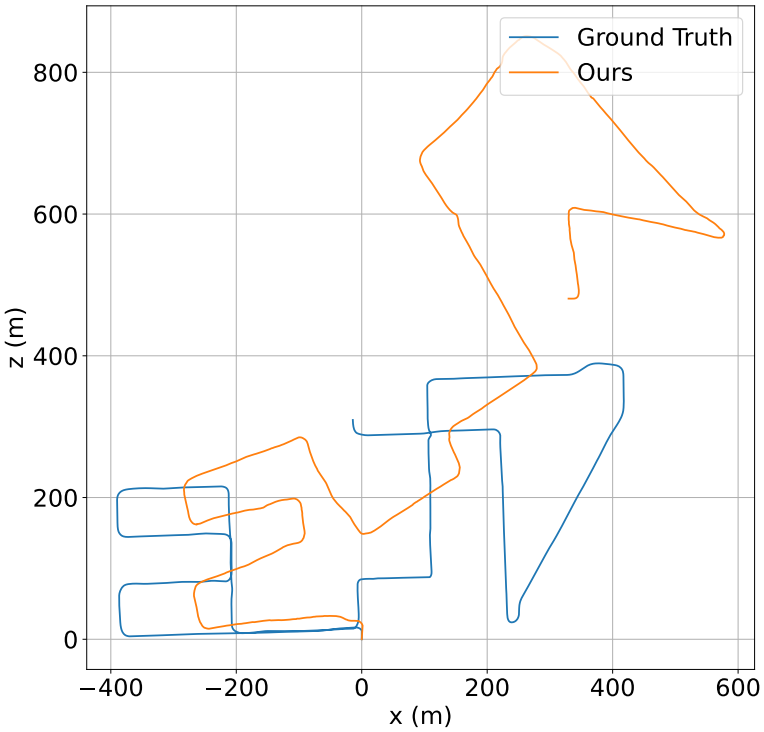



Figure 5.3: Comparison between the estimated path and the ground truth on sequence 08 of KITTI odometry dataset. The drift accumulation is more severe on this dataset, and reaches about 350 meters at the end.

One potential solution to mitigate drift accumulation is loop closure, a common technique in SLAM algorithms. The loop closure process identifies previously visited areas and realigns past poses to ensure consistency when revisiting those positions. Most of the sequences from KITTI include loop closures by overlapping the start and endpoints. Particularly, sequences 00, 02, 05, and 08 have multiple spots revisited along their tracks, making it possible to improve significantly the results through this technique.

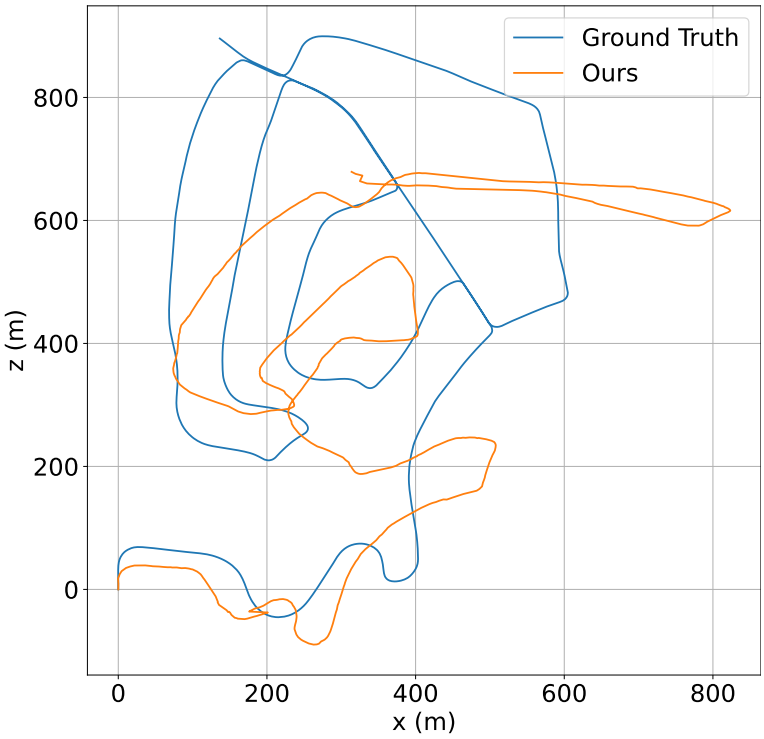



Figure 5.4: Comparison between the estimated path and the ground truth on sequence 02 of KITTI odometry dataset. The first half of the track is composed by a forest area with poor features, which degrades the pose estimation.

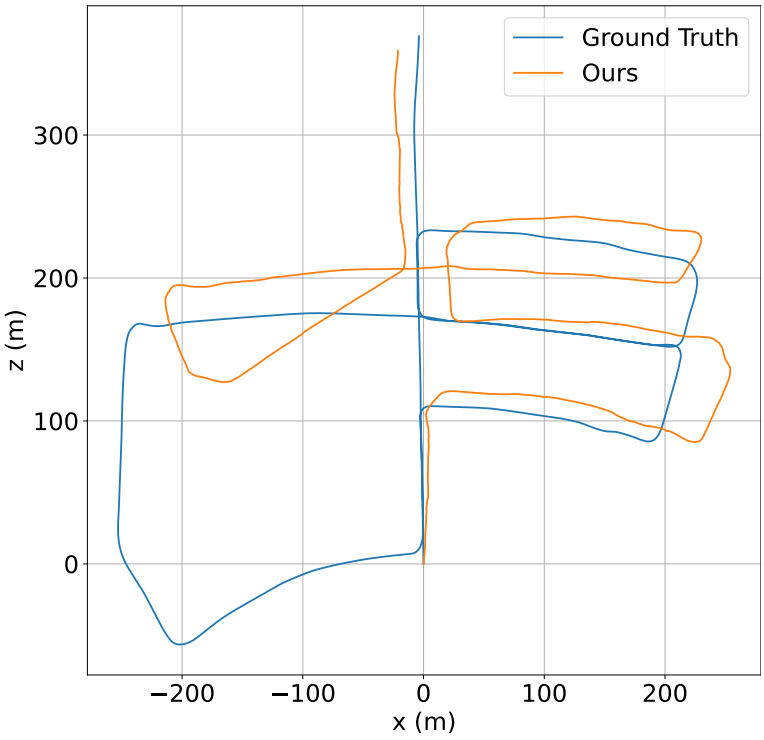



Figure 5.5: Comparison between the estimated path and the ground truth on sequence 05 of KITTI odometry dataset. Until the last large curve, the estimated path is able to stay fairly close to the ground truth.

Another scenario where the algorithm does not present a good result is in environments with few distinguishable features. Sequence 01 - Figure 5.6 - primarily consists of a

highway with feature-poor scenes, despite having a relatively simple trajectory. Figure 5.7 shows that the number of matches is small, and they are also concentrated around the line between the road and the horizon. The features must be well-distributed across the image to recover an accurate pose.

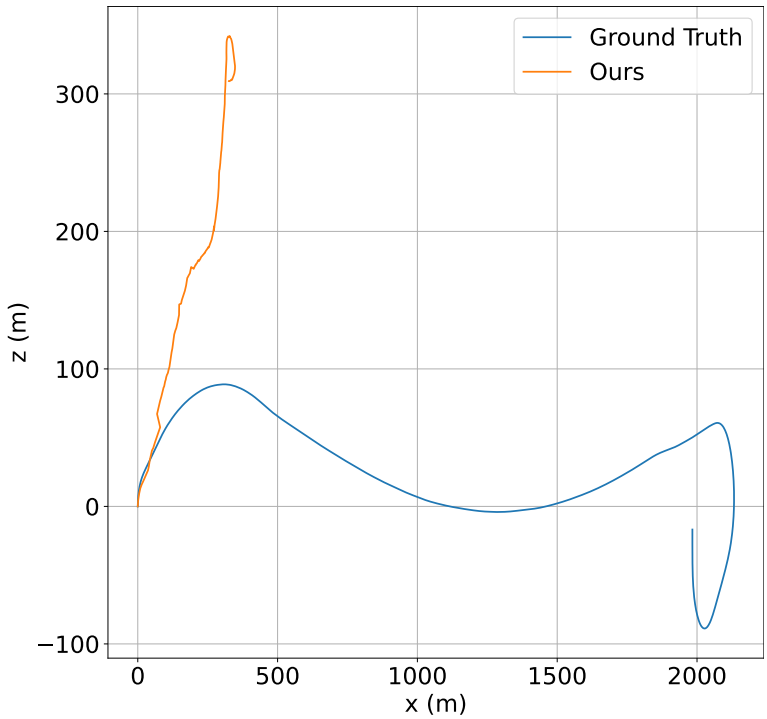



Figure 5.6: Comparison between the estimated path and the ground truth on sequence 01 of KITTI odometry dataset. The open highway area affects both the feature matching and depth estimation.

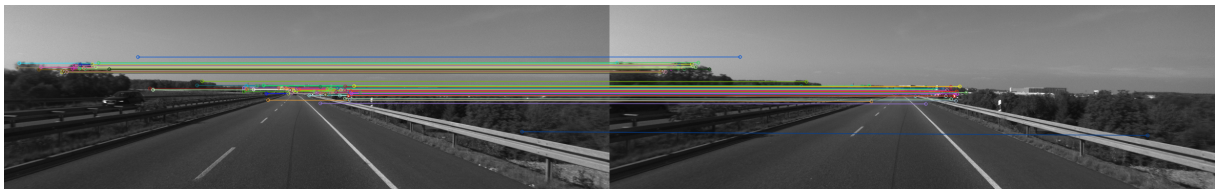


Figure 5.7: Feature matching between two consecutive frames extracted from sequence 01 of KITTI odometry dataset. The low number of feature and bad distribution across the image degrades the transformation recovery.

Furthermore, the open space also negatively affects performance. As noted in (Scaramuzza and Fraundorfer, 2011), stereo visual odometry methods degrade when the scene depth significantly exceeds the camera baseline. This issue can be seen in Figure 5.8, where the upper half of the depth estimation - the sky - contains a significant amount of noise.

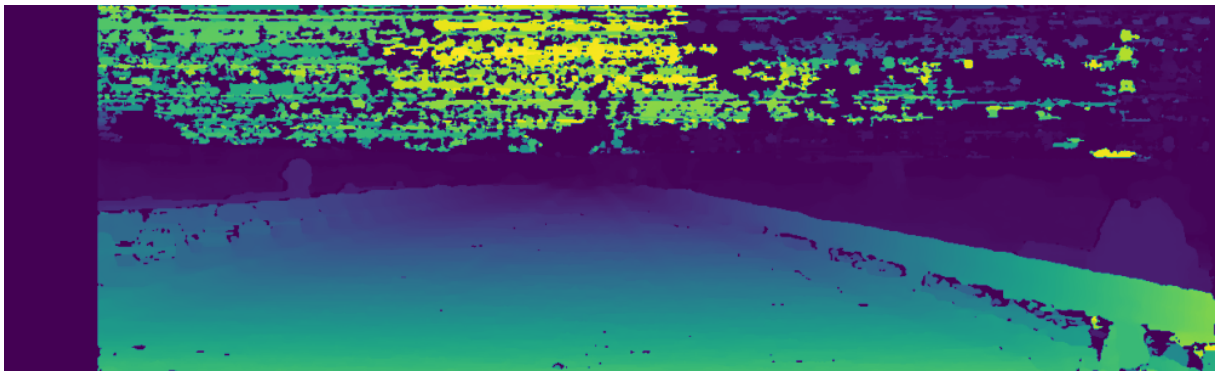


Figure 5.8: Depth estimation on a stereo pair extracted from sequence 01 of KITTI odometry dataset. The upper half of the image is mostly composed by the sky, where the lack of textures and distance decrease the accuracy of the depth estimation.

The sequences with results that match more closely the ground truth are 10 and 09, shown in Figures 5.9 and 5.10, respectively. These datasets feature relatively simple trajectories in urban areas.

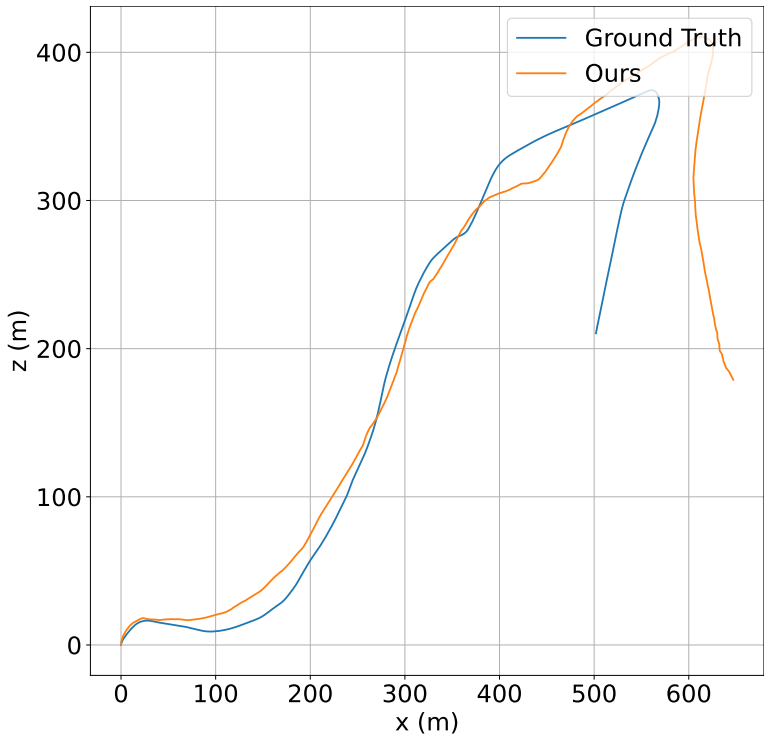



Figure 5.9: Comparison between the estimated path and the ground truth on sequence 10 of KITTI odometry dataset. The error remains fairly low during most of the trajectory.

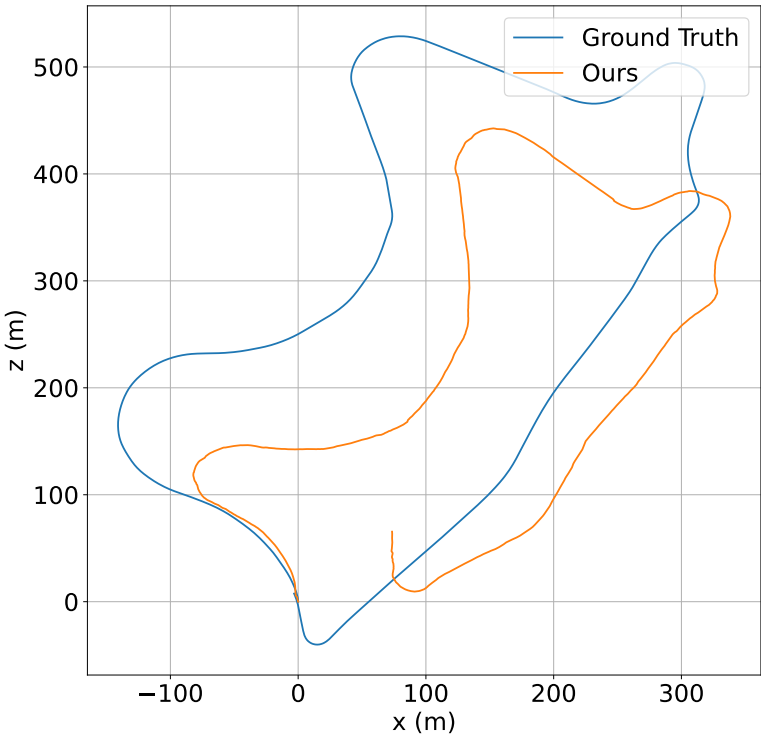



Figure 5.10: Comparison between the estimated path and the ground truth on sequence 09 of KITTI odometry dataset. Although the drift at the start propagates for most of the trajectory, the general shape still matches the ground truth.

In sequence 07 (Figure 5.11), the trajectory's general shape is preserved, but a drift of about 50 meters near the start propagates throughout the path estimation.

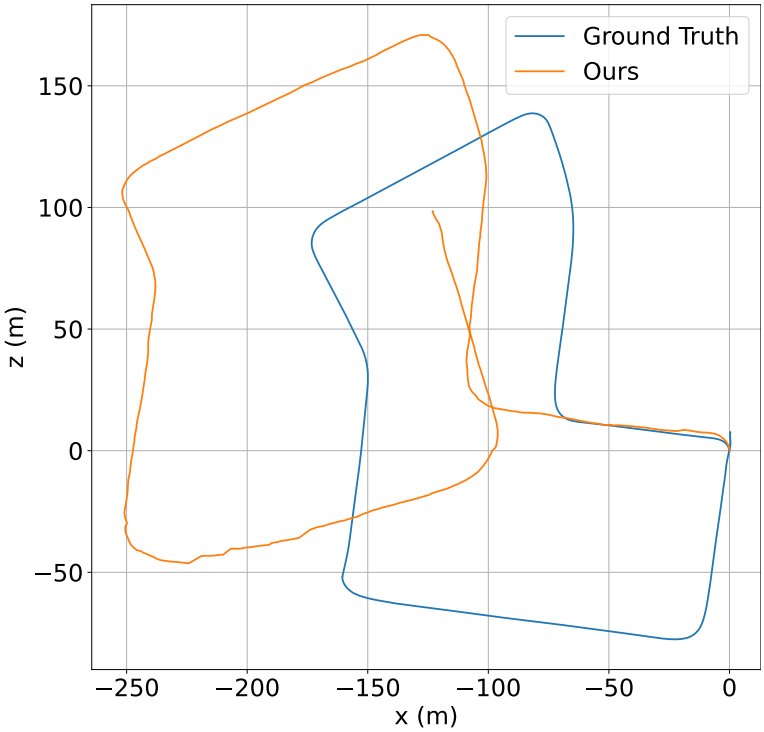



Figure 5.11: Comparison between the estimated path and the ground truth on sequence 07 of KITTI odometry dataset. Similar to sequence 09, the general shape still matches the ground truth.

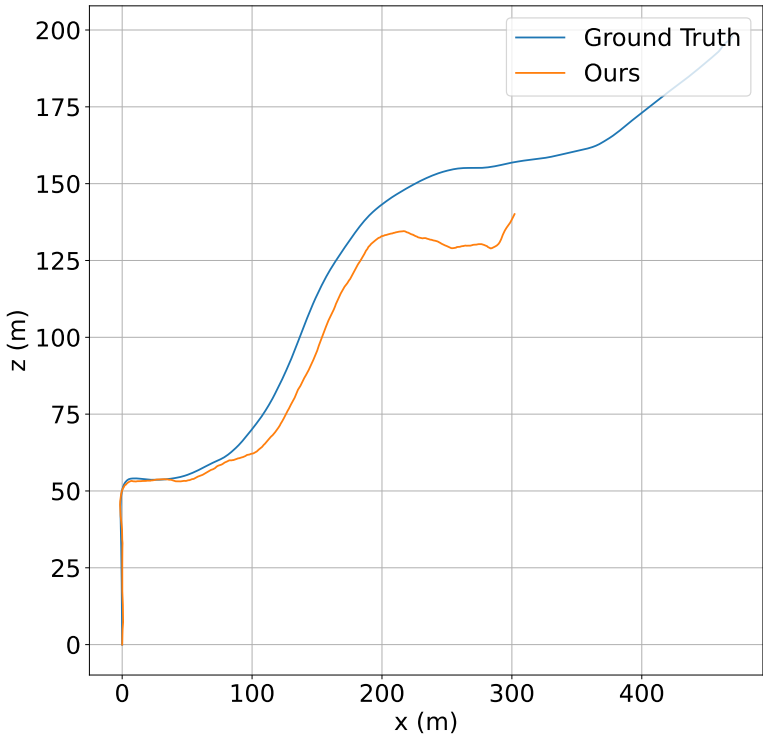



Figure 5.12: Comparison between the estimated path and the ground truth on sequence 03 of KITTI odometry dataset. The drift at the end is mostly due to entering a forest area with poor feature matching.

Conversely, sequence 03 (Figure 5.12) starts with low error, but the error increases toward the end, as the trajectory enters a forested area. Trees often do not produce good matches due to the similar patterns on the leaves, as shown in Figure 5.13. Most of the poor matches are filtered by Lowe's ratio test, leaving only a few valid matches. A similar issue is observed in sequence 02 (Figure 5.4), where, near 200 meters, the estimation drifts significantly.

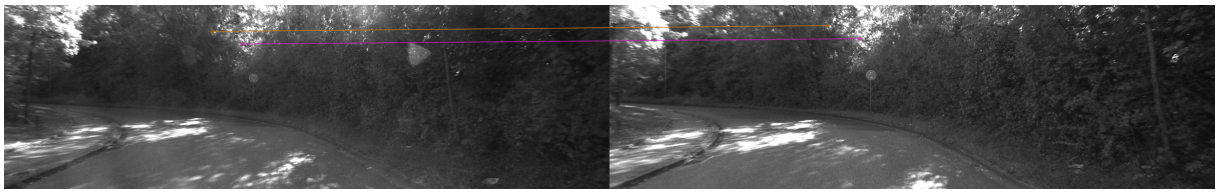


Figure 5.13: Lack of good features on a forest area caused by the similar textures of the trees. The pair of consecutive images was extracted from the start of KITTI dataset's sequence 02.

## 5.2 CONCLUSION

The experiments with the KITTI dataset (Geiger et al., 2012) have shown that the pipeline can achieve promising results, but there still a large room for improvement. Although the effect of the drift accumulation is visible, the RPE was low and the ATE was still within 10% of the trajectory length in most sequences. The general shape of the estimated path was similar to the ground truth as well. As discussed in each sequence, accuracy can be increased by further optimizing the pipeline - to match the real-time frequency - as well as integrating SLAM algorithms, such as loop closure, and researching ways to refine the feature matching in adverse scenarios.

# 6 CONCLUSION

Most modern autonomous robots are already equipped with a navigation system and SLAM-related algorithms. Visual odometry is one such algorithm, enabling pose estimations using only a camera feed. First, this work has presented a literature review of the foundational concepts necessary for VO, including the mathematical model of a pinhole camera, core principles of epipolar geometry, feature extraction, feature matching techniques, depth estimation, and a PnP-based method for pose estimation. Based on these techniques, a ROS-based C++ implementation of a real time stereo VO pipeline was developed and evaluated using the widely recognized KITTI odometry benchmark suite (Geiger et al., 2012), achieving an average error of approximately 10% of the trajectory length across most sequences.

As shown by the results, there is still significant room for improvement. It is visible that one of the main concerns is the drift accumulation over time. Future work could explore integrating visual SLAM techniques, such as loop closure and pose optimization through bundle adjustment, as well as other algorithms mentioned on (Scaramuzza and Fraundorfer, 2011), to improve performance.

Besides, experimenting with different feature extraction algorithms could help achieve a better balance between accuracy and computational speed, directly reducing the total estimation time. Deep learning research is also another promising area for visual odometry as a way to address one of its fundamental weaknesses, which is featureless scenarios.

Finally, visual-inertial odometry systems have shown great accuracy by fusing IMU data with the odometry estimations and are another area that can be explored in the future as well.

# REFERENCES


Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Campos, C., Elvira, R., Rodriguez, J. J. G., M. Montiel, J. M., and D. Tardos, J. (2021). Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE Transactions on Robotics*, 37(6):1874–1890.

Cvišić, I., Marković, I., and Petrović, I. (2023). Soft2: Stereo visual odometry for road vehicles based on a point-to-epipolar-line metric. *IEEE Transactions on Robotics*, 39(1):273–288.

Felzenszwalb, P. and Huttenlocher, D. (2006). Efficient belief propagation for early vision. *Int J Comput Vision*, 70:41–56.

Fraundorfer, F. and Scaramuzza, D. (2012). Visual odometry : Part ii: Matching, robustness, optimization, and applications. *IEEE Robotics Automation Magazine*, 19(2):78–90.

Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361.

Gionis, A., Indyk, P., and Motwani, R. (2000). Similarity search in high dimensions via hashing. *Proceeding VLDB '99 Proceedings of the 25th International Conference on Very Large Data Bases*, 99.

Harris, C. and Stephens, M. (1988). A combined corner and edge detector. In *Proceedings of the Alvey Vision Conference*, pages 23.1–23.6. Alvety Vision Club. doi:10.5244/C.2.23.

Hirschmuller, H. (2008). Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):328–341.

Howard, A. (2008). Real-time stereo visual odometry for autonomous ground vehicles. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3946–3952.

Ke, T. and Roumeliotis, S. (2017). An efficient algebraic solution to the perspective-three-point problem.

Lowe, D. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–.

Macenski, S., Foote, T., Gerkey, B., Lalancette, C., and Woodall, W. (2022). Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074.

Nister, D. (2004). An efficient solution to the five-point relative pose problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):756–770.

Nister, D., Naroditsky, O., and Bergen, J. (2004). Visual odometry. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 1, pages I–I.

Qin, T., Li, P., and Shen, S. (2018). Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020.

Rosinol, A., Abate, M., Chang, Y., and Carlone, L. (2020). Kimera: an open-source library for real-time metric-semantic localization and mapping.

Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571.

Scaramuzza, D. and Fraundorfer, F. (2011). Visual odometry [tutorial]. *IEEE Robotics Automation Magazine*, 18(4):80–92.

Yang, N., von Stumberg, L., Wang, R., and Cremers, D. (2020). D3vo: Deep depth, deep pose and deep uncertainty for monocular visual odometry. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1278–1289.

Zhan, H., Weerasekera, C. S., Bian, J.-W., and Reid, I. (2020). Visual odometry revisited: What should be learnt? In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4203–4210.

Zhang, Z. (2000). A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334.