Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería en Electrónica y Automática Industrial

Trabajo Fin de Grado

Development of a Visual Odometry System as a Location Aid for Self-Driving Cars

Author: Carmen Pérez Carmona

Advisor: Luis Miguel Bergasa Pascual

2021

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería en Electrónica y Automática Industrial

Trabajo Fin de Grado

Development of a Visual Odometry System as a Location Aid for Self-Driving Cars

Author: Carmen Pérez Carmona

Advisor: Luis Miguel Bergasa Pascual

Tribunal:

President: Miguel Ángel García Garrido

1st Vocal: Esther Palomar González

2nd Vocal: Luis Miguel Bergasa Pascual

Deposit date: September 30th, 2021

"Nothing in life is to be feared; it is only to be understood"
Marie Curie

Acknowledgements

I would like to thank my tutor Luis Miguel Bergasa for giving me the opportunity to do this project, as well as the whole RobeSafe team, for helping me whenever I needed it. I would also like to express my gratitude to my family and friends.

Resumen

Conocer la posición exacta que ocupa un robot y la trayectoria que describe es esencial en el ámbito de la automoción. Durante años se han desarrollado distintos sensores y técnicas para este cometido que se estudian a lo largo del trabajo.
En este proyecto se utilizan dos cámaras a bordo del vehículo como sensores de percepción del entorno. Se propone un algoritmo basado únicamente en odometría visual, es decir, analizando la secuencia de imágenes captadas por las cámaras, sin conocimiento previo del entorno y sin el uso de otros sensores, se pretende obtener una estimación real de la posición y orientación del vehículo. Dicha propuesta se ha validado en el dataset de KITTI y se ha comparado con otras técnicas de odometría visual existentes en el estado del arte

Palabras clave: KITTI, Odometría Visual, Vehículos Autónomos, Percepción, Automoción.

Abstract

Knowing the exact position occupied by a robot and the trajectory it describes is essential in the automotive field. Some techniques and sensors have been developed over the years for this purpose which are studied in this work.
In this project, two cameras on board the vehicle are used as sensors for the environment perception. The proposed algorithm is based only on visual odometry, it means, using the sequence of images captured by the cameras, without prior knowledge of the environment and without the use of other sensors. The aim is to obtain a real estimation of the position and orientation of the vehicle. This proposal has been validated on the KITTI benchmark and compared with other Visual Odometry techniques existing in the state of the art.

Keywords: KITTI, Visual Odometry, Self-driving vehicles, Perception, Automotion.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1. Motivation

The objective of Visual Odometry is to obtain an estimation of the relative position and orientation of a vehicle, by analysing sequences of images obtained by cameras placed on board the vehicle without prior knowledge of the environment.
The use of GPS is important for the rough planning of the movement of the car to its target. However, the error is too large to guide a vehicle, so it is necessary to fuse this information with that provided by other sensors, such as in our case, the Visual Odometry.
Any estimated position is relative to the starting point. In cases where the signal provided by GPS is lost or is low, Visual Odometry helps us not to lose the vehicle's location and to continue estimating the trajectory.
In recent years, visual odometry has gained importance and is used in numerous applications, mostly in ground vehicles, space missions, underwater missions and air vehicles.

1.2. Objectives and organization of the project

The main objective of this project is to develop a Visual Odometry System as a Location aid for self-driving cars. To this propose we will study different Visual Odometry methods of the state of the art. The organization of this work is divided in the following chapters:

1. Visual Odometry Problem. State of the art and theorical explanation of Visual Odometry problem.
2. KITTI Benchmarking. This chapter explains what KITTI Benchmarking is all about, as it will be the dataset we will use to test our algorithm.
3. Comparison of Methods in KITTI. In this chapter we will explain some of the stereo Visual Odometry algorithms that are currently leading the KITTI ranking.
4. Algorithm. Structure of the proposed algorithm.
5. Results. Obtained results and comparison of the result with the studied algorithms.
6. Conclusion. A brief analysis of the proposed algorithm and its results.

Chapter 2

Visual Odometry Problem

2.1. Introduction

In the field of robotics, a large part of the investigation and development is aimed to increase the autonomy and versatility of robots.

In 1966, Shakey (Figure 2.1), considered the first inteligent robot in history, was born. It used a camera as an eye and combined aritificial inteligence and computer vision to estimate trajectories and avoid obstacles.The history of computer vision dates back to the 1980s with the development of computer engineering and the creation and evolution of microprocessors capable of capturing, processing and reproducing images taken by a camera.

Computer Vision is one of the technologies that is increasingly used in industry as it has multiple fields of application, for example:

- Pick and Place.
- Quality control.
- Assemblies.
- Object detection.
- Etc.

The aim of artificial vision is to use images to process information from the environment surrounding the robot so that the data obtained can be processed by the machine and, in this way, enable it to make decisions.

The term odometry comes from two Greek words *hodos* (path) and *metron* (measurement). Odometry is used by robots to estimate their position relative to their initial position. The term Visual Odometry was first used by David Nister because the concept of motion estimation by Computer Vision was similar to the concept of wheel odometry estimation. Wheel odometry uses the wheel motion data provided by encoders between a previous time and the current time to determine the new position of a robot, while visual odometry uses the data provided by a vision system analysing the change between a previous instant of time and the current time.

In the short term visual odometry provides good results, but its problem is that motion information is incrementally integrated over time and so error accumulation is inevitable. However, it is still very

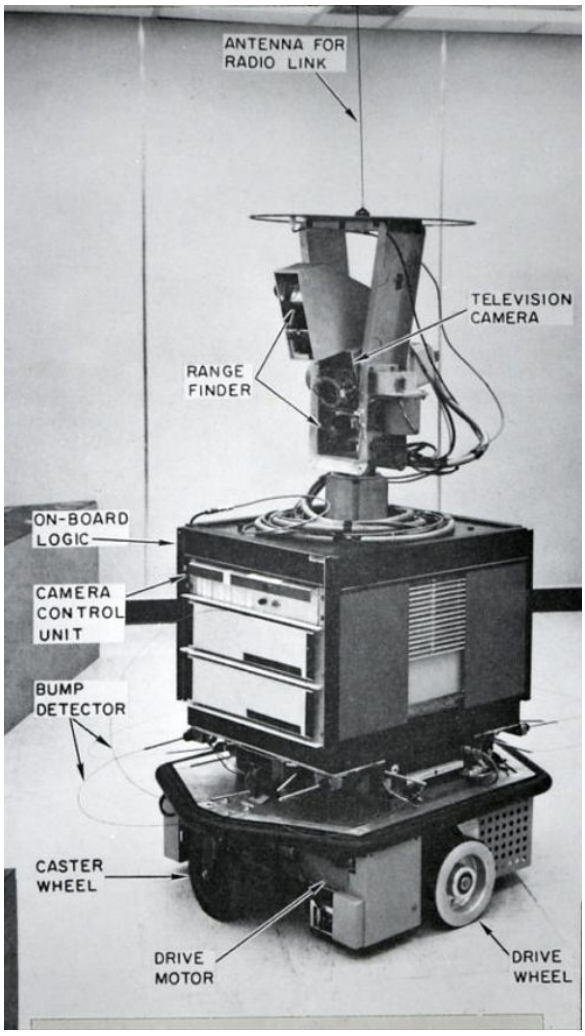


Figure 2.1: Shakey.

important in robotics to obtain a more accurate estimate when used in conjunction with absolute positioning measurements, for example, that provided by a good GPS signal. In places where such a signal is weak or absent, it is useful to have this vision system active.

2.2. State of the Art

With the growth of automation in various fields of engineering, mobile robots are becoming increasingly popular. There are several techniques for estimating the position of mobile robots. One of them, the simplest, is the estimation of the movement of a ground vehicle using Wheel Encoders. Other conventional localisation methods such as GPS, INS, SONAR or RADAR are used for many applications, however, they have several limitations. [1]

Accurate estimation of the robot's position is essential for autonomous driving, so visual odometry is used, which has become increasingly popular because it offers high accurary.[2] Visual odometry estimates vehicle position by analysing changes between images obtained through cameras on board the vehicle. [3]

Currently we can classify visual odometry into two main groups: geometric approach that uses information from projective geometry and Non-Geometric-Based-Approach that is based on machine learning.

In the prejective geometry group we have different techniques:

- Feature Based: In this technique, the points of interest are located and matched with the same points in the next image. Harris Corner Detector [4], SIFT features detector [5], SURF features detector [6], etc. are some of the methods used to find the points of interest. Once these points are matched between the different images, the position and movement of the robot is estimated.

- Appereance-Based: A descriptor of the appearance of the image is created without the need to draw keypoints. A map of the appearance of the environment is created and relationships between stored positions are established. [7]

- Hybrid of Feature and Apperance:This technique takes advantage of the advantages of the methods mentioned above: on the one hand, it uses feature-based methods, which provide a lot of relevant information, but at the same time lose important information for the reconstruction of the environment; while apperance-based methods provide enough information for the total reconstruction of the scene, but present problems in some areas.

  On the other hand, in the Non-Geometric-Based-Approach, with the growth of machine learning, visual odometry systems based on deep learning are starting to be developed. Most of these systems use supervised training models, which results in a very slow process to obtain real data [8], so some researchers propose visual odometry systems based on unsupervised learning, which is not based on completely real data, but is much easier to obtain [9].

Depending on the number of cameras, visual odometry can be referred to as monocular visual odometry or stereoscopic visual odometry, which produces less noise than the former because, using a simple triangularisation technique, the distance at which objects are located can be estimated by generating a three-dimensional image.

In recent years, different methods have been developed that combine visual odometry with information obtained by different sensors, for example, in [10] an algorithm is proposed that fuses data obtained by IMU (Intertial Measurement Unit), GPS and visual odometry for robot localisation. However, data fusion between different sensors increases the computational complexity. The LIBVISO method proposed by Andreas Geiger is based solely on stereoscopic visual odometry, which reduces the cost of the system and also allows its application in real time.

2.3. Explanation of Visual Odometry

Throughout this section the phases that make up a Visual Odometry System following a projective geometric approach will be explained.

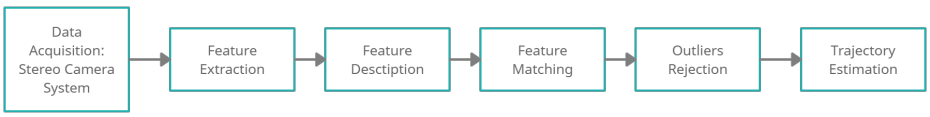


Figure 2.2: Diagram of main phases of Visual Odometry.

As we can see in Figure 2.2 after the images acquisition the next step is to detect the features and extract theirs descriptors, once we have these descriptors, the feature matching is performed. During the feature matching inliers are obtained, but some outliers are obtained too. An outlier filter is usually used to reject them. Finally, different mathematical calculations are used to estimate the trajectory.

2.3.1. Images Acquisition

Depending on the number of the vision cameras in the visual system we can talk about monocular system or stereoscopic visual system. The main difference between them is that in stereoscopic vision system we can have three-dimensional information of the environment and we obtain more stable information.

In a Visual Odometry system the images are only source of information we have about the environment, so the data we will extract will depend on the quality of them. The disadvantage of using cameras as sensor is that they are passive elements, so they are sensitives to weather conditions and light changes.

When estimating the movement of the vehicle, variations in the position of the keypoints will be observed, therefore, the more static the image is, the higher the quality of the keypoints found will be, since the keypoints in dynamic objects will not provide valid information.

In other hand, the image acquisition should be the most homogeneously as possible, and the images should overlap in order to be able to find the correspondences between them.

(a) Example image from KITTI Dataset

(b) Example image from KITTI Dataset

Figure 2.3: Example of two consecutive images extracted from KITTI Dataset

2.3.2. Camera Projective Geometry

Although other camera models exist, in this section the pin-hole model will be explained, as it is commonly used in machine vision systems. Figure 2.4 shows the geometrical model of pin-hole camera. Pin-Hole camera model describes the relationship between an O_{world} point and its corresponding

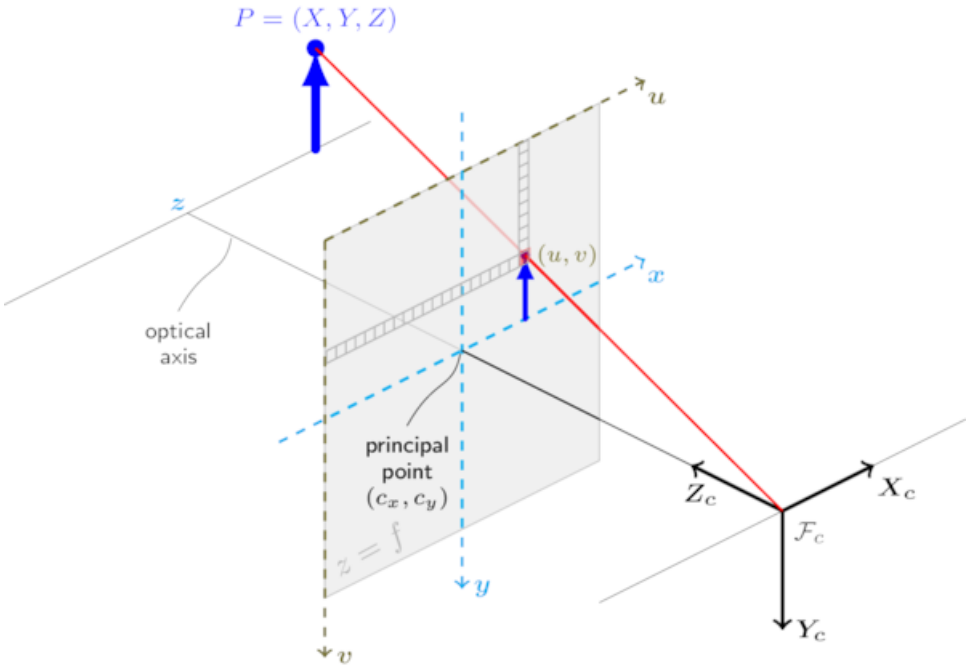


Figure 2.4: Pin-Hole Camera Model

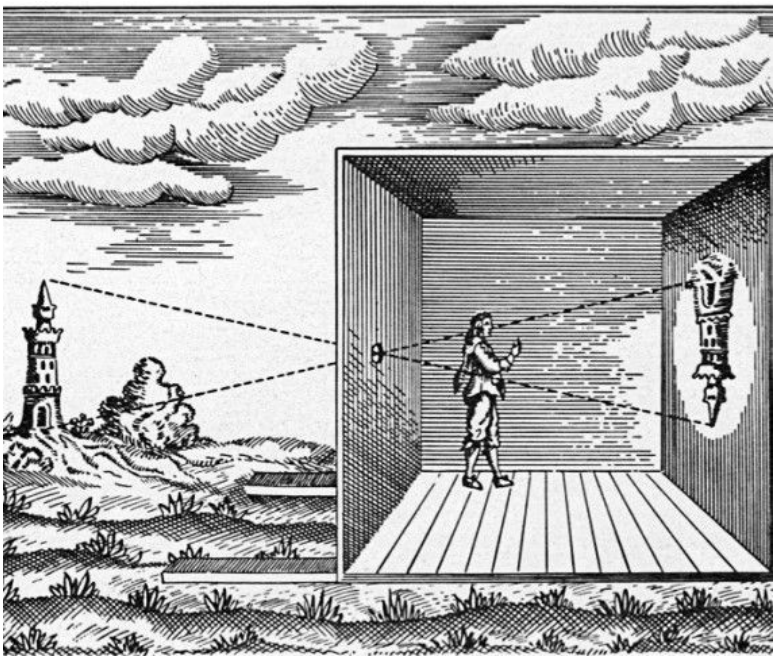


Figure 2.5: Camera Obscura Ilustration

projection in the image plane. The two most important parts in the geometry of the camera are:

- Focal length: it defines the distance between the pin-hole camera and the image plane. In Figure 2.5 we can observe a drawing of the first concept of a camera, the camera obscura. In this case, the distance between the wall, where the invert image appears, and the wall with the hole, corresponds to the focal length.
- Camera center: Pin-hole coordinates C_x and C_y.

The purpose of the camera is to transform the coordinates from the world frame coordinates into the coordinates of the camera plane. To do this, light travels through the camera center to the surface of the sensor. As we can see, a inverted image is obtained.

To simplified the problem we usually use a virtual plane in front of camera center, like in Figure 2.6.

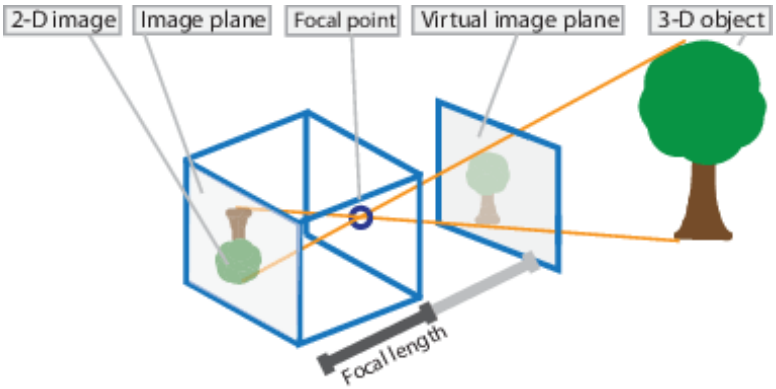


Figure 2.6: Virtual image plane in Pin-hole camera model.

We use a transformation matrix between the world coordinate frame and the camera coordinate frame. The parameters that refer to the camera pose are called extrinsic parameters. The projection problem is then reduced to two steps:

- We project from the world coordinate frame to the camera coordinate frame.
- We project the camera coordinates to the image coordinates.

Finally the image coordinates are transformed into pixel coordinates. To transform from world to camera coordinates we use a transformation matrix $[R|t]$ and to transform from camera coordinates to image coordinates we define a matrix K_{3x3} that depends only on the intrinsic camera parameters.

$$O_{image} = \begin{bmatrix} f & 0 & u_o \\ 0 & f & v_o \\ 0 & 0 & 1 \end{bmatrix} O_{camera} = KO_{camera} \tag{2.1}$$

To transform from world coordinate frame to image coordinate frame we define the projection matrix P.

$$P = K[R|t] \tag{2.2}$$

Finally we obtain the following expression:

$$O_{image} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = K[R|t] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.3}$$

The last step is to transform the image coordinates into pixel coordinates, which is obtained with the following expression:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{z} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{2.4}$$

The world coordinates are transformed into camera coordinates using extrinsic parameters and the camera coordinates are transformed into image coordinates using intrinsic parameters (Figure 2.7).

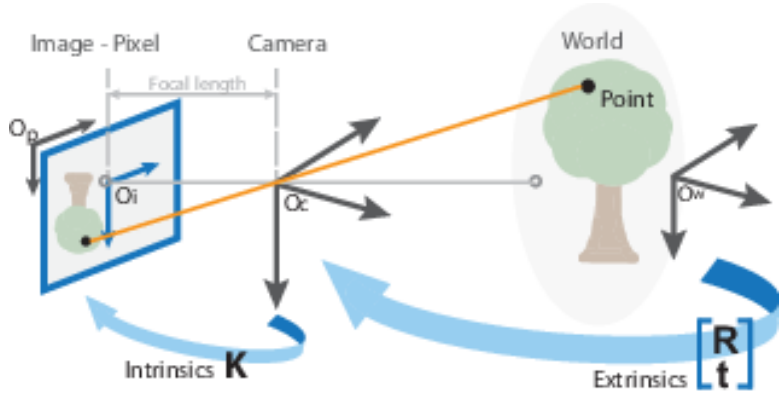


Figure 2.7: Intrinsic and Extrinsic parameters.

2.3.2.1. Camera Calibration

Camera calibration is the process by which the unknown intrinsic and extrinsic parameters of the camera are found. Camera often introduce distorsion in the image. The main distortions are radial distortion and tangential distortion.

To obtain these parameters we use a scene whose geometry is known, in this way we obtain the position of the three-dimensional points from the two-dimensional image. This scene is usually a chess grid that is placed in different positions and orientations.

Since the three-dimensional coordinates are known and the distances between the squares are also known, the intrinsic parameters of the camera are calculated, usually using a least squares optimization problem.

2.3.3. Depth Estimation

Stereopsis, stereo vision process, was defined for the first time by Charles Wheatstone. Is the process by which the mind perceives an object in three dimensions from two slightly different images of the physical world projected on the retina of the eye.

In machine vision, stereopsis is used to estimate the depth at which an object is located. Figure 2.8 shows a simplied model of a stereoscopic camera. There are two important parameters:

- Focal Length: it is the distance between the camera and the image plane,
- Baseline: It is the distances between the two camera centers.

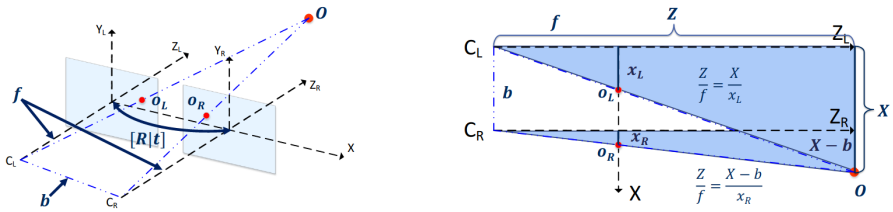


Figure 2.8: Stereo Camera Model

By observing the singular point O, and with a simple triangulation, we can estimate the depth at which it is located.

$$\frac{Z}{f} = \frac{X}{x_l} = \frac{X-b}{x_r} \tag{2.5}$$

Now, we define disparity. Disparity is the difference between x_l and x_r.

$$d = (x_l - x_r) \tag{2.6}$$

Then, we can get:

$$\frac{Z}{f} = \frac{X}{x_l} \rightarrow Zx_l = fX \tag{2.7}$$

$$\frac{Z}{f} = \frac{X-b}{x_r} \rightarrow Zx_r = fX - fb \tag{2.8}$$

$$Zx_r = Zx_l - fb \tag{2.9}$$

$$fb = Z(x_l - Zx_r) \tag{2.10}$$

$$fb = Zd \rightarrow Z = \frac{fb}{d} \tag{2.11}$$

OpenCV implements good tools for depth estimation. First step is to compute disparity map, as it is shown in Figure 2.9 and then, once we have the intrinsic values of the camera, we can calculate the depth map, as it is shown in Figure 2.10.

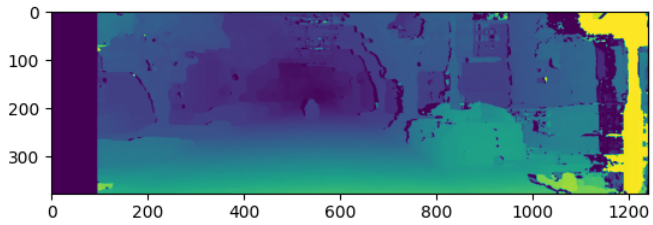


Figure 2.9: Disparity Map Example

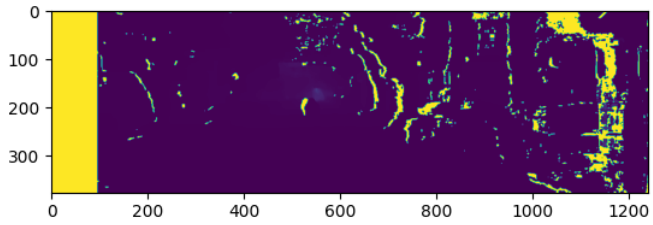


Figure 2.10: Depth Map Example

2.3.4. Feature Detection

Feature Detection process consists on finding points of interest in the acquired images. These points are called *features*. A feature is characterised by being easily distinguishable and comparable in an image. These features should have a high probability of being found in another image of the same scene.

Points of interest should have the next characteristics:

- Robust: Features should be insensitive to noise, compress processes, discretisation effects, or geometry and light changes.
- Salience: they must be easily distinguishable, identifiable and different from their immediate neighborhood.
- Repeatability: points of interest must be able to be found in multiples images.
- Locality: it must be precise in both position and scale.
- Quantity: it should be enough features in an image.
- Efficiency: minimal computational time.

2.3.4.1. Harris Corner Detector

Harris Corner detector [11] is one of the most common feature detector. It is a corner detector. It finds the difference in intensity for a displacement of (u,v) in all directions [12]. Depending on the changes, there are three types of detected regions, as we can see in Figure 2.11.

- Flat Region: no change in all directions.
- Edge Region: no change along the edge region.
- Corner Region: significant changes in all directions.

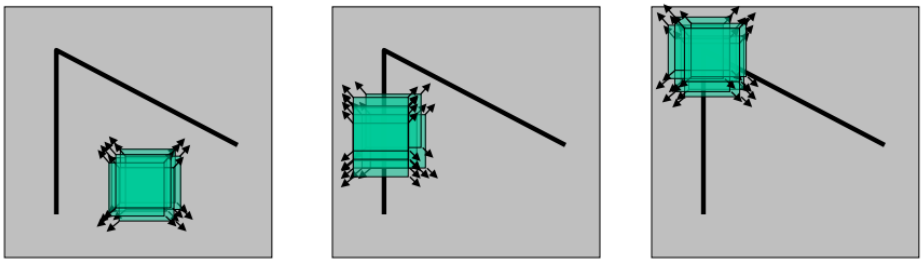


Figure 2.11: Harris Corner Regions

This is expressed as below:

$$E(u,v) = \sum_{x,y} \omega(x,y)[I(x+u,y+v) - I(x,y)]^2 \tag{2.12}$$

Where:

- $E(u,v)$ is the window function to maximize for corner detection.
- $\omega(x,y)$ is the window function.
- $I(x+u,y+v)$ is shifted intensity.
- $I(x,y)$ is intensity.

Applying Taylor Expansion to Equation 2.12 we obtain the next expression:

$$E(u,v) \approx [u,v]M \begin{bmatrix} u \\ v \end{bmatrix} \tag{2.13}$$

Where:

$$M = \sum_{x,y} w(x,y \begin{bmatrix} I_x I_x & I_x I_y I_y I_x & I_y I_y \end{bmatrix} \tag{2.14}$$

Where I_x and I_y are the intensity gradients of the image. To determine if the region is a corner or not they created a score:

$$R = det(M) - k(trace(M))^2 \tag{2.15}$$

Where:

- det(M) = $\lambda_1 \lambda_2$
- trace(M) = $\lambda_1 + \lambda_2$
- λ_1 and λ_2 are the eigen of M.

So the values of this eigen values will decide if the window is a flat, a edge or a corner.

- If λ_1 and λ_2 are small, SSD is small in all directions, so it is a flat region.
- When $\lambda_1 \gg \lambda_2$ or $\lambda_2 \gg \lambda_1$ the region is a edge.
- When R is large, it means that both λ_1 and λ_2 are large, so the region is a corner.

It can be represented as the Figure 2.12.

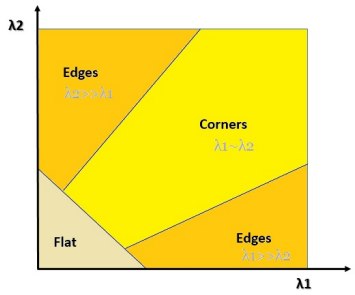


Figure 2.12: Type of region depending on λ_1 and λ_2 in Harris Corner Detector

In the figure below (Figure 2.13) we can see an example of feature detection using Harris Corner Detector.

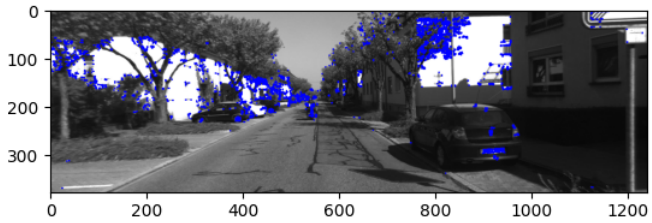


Figure 2.13: Harris Corner Detector used in an image from KITTI Dataset.

2.3.4.2. Shi-Tomasi Detector

Shi-Tomasi Detector [13] is based in Harris Corner Detector. Theirs authors realised that there is a better way of estimating the score. Shi-Tomasi proposed the next expression to calculate R:

$$R = min(\lambda_1, \lambda_2) \tag{2.16}$$

Then, if λ_1 and λ_2 are above the minimum value, it is considered a corner Figure 2.14. In the figure

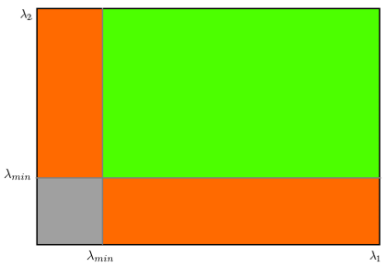


Figure 2.14: Type of region depending on λ_1 and λ_2 in Shi-Tomasi Detector

below Figure 2.15 we can see an example of feature detection using Shi-Tomasi Detector.

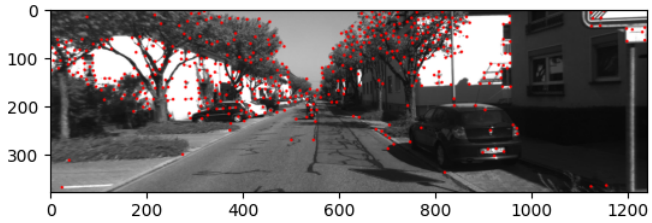


Figure 2.15: Shi-Tomasi Detector used in an image from KITTI Dataset

2.3.4.3. SIFT

One problem with the detectors explained above is they are not scale invariant, which means that A corner in a small image within a small window is flat when it is zoomed in the same window.

SIFT (Scale Invariant Feature Transform) is a feature detector and descriptor which considers scale and orientation changes [5]. The main advantage of the SIFT detector is that the features it obtains are very particular, so they will be easier to match even with drastic changes in illumination, scale or rotation. First of all the algorithm performs a convolution with a Gaussian function centred on each. The final value of each pixel in the new image depends on the weighting that the Gaussian function assigns to its neighbouring pixels. Using this filter, shown in Equation 2.17 the original image would be softened.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.17}$$

Where:

- σ is the standard deviation.
- x and y are pixel coordinates.

Laplacian of Gaussian is found for the image with different σ values. These differences are called Difference of Gaussians (DoG). SIFT calculates all Differences for each pair of successive smoothed images. Each pixel of a DoG is compared with its eight nearest neighbouring pixels on the same scale, with 9 from the previous scale and 9 from the next scale. If the pixel intensity turns out to be a maximum or a minimum it is considered as a point of interest. Once the keypoint is detected, the algorithm use a Taylor series expansion of scale space to get the exact position of the pixel, and if the intensity is below a threshold, it is rejected. Then, the orientation is assigned to each keypoint to achieve rotation invariance.

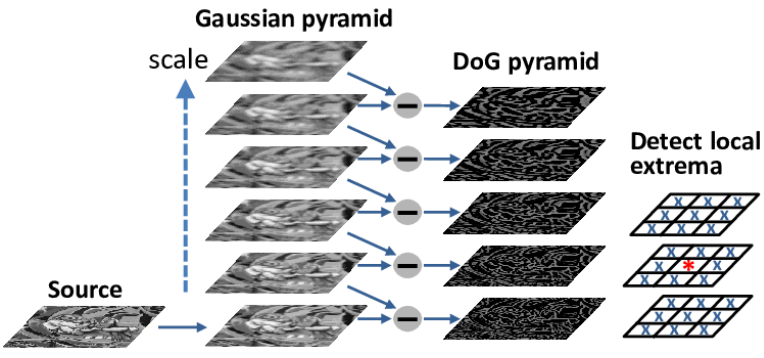


Figure 2.16: SIFT detector algorithm.

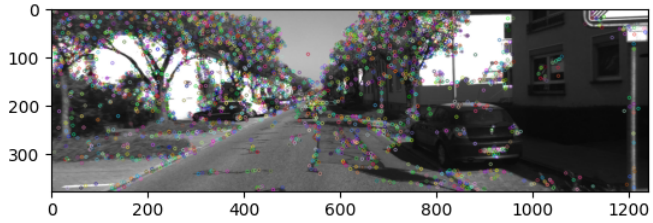


Figure 2.17: SIFT Detector used in an image from KITTI Dataset

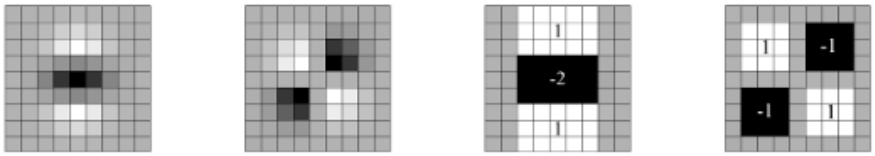


Figure 2.18: Gaussian second order partial derivates in y-direction and xy-direction (left) and approximations to them using box filters (right)

2.3.4.4. SURF

SURF (Speeded Up Robust Feature) [14] was introduced by first time in 2006, and it is considered as the speeded version of SIFT. It is a feature detector and descriptor.

As feature detector SURF is based on Hessian-Laplace detector. It is based on Hessian Matrix. To select the location and scale, a determinant of the Hessian is used. As we can see in Equation 2.18 given a point [x,y] in the image I, the Hessian Matrix is defined as follows:

$$H(x,y,\sigma) = \begin{bmatrix} L_{x,x}(x,y,\sigma) & L_{x,y}(x,y,\sigma) \\ L_{x,y}(x,y,\sigma) & L_{y,y}(x,y,\sigma) \end{bmatrix} \tag{2.18}$$

Where $L_{x,y}$ follows the Equation 2.19 and similarly for $L_{x,x}$ and $L_{y,y}$.

$$L_{x,y}(x,y,\sigma) = \frac{\partial^2}{\partial x \partial y} G(\sigma) * I(x,y) \tag{2.19}$$

SURF, instead of performing a convolution with the second derivative of the Gaussian, uses box filters. In Figure 2.18 we can observe a comparison between the Gaussian second order partial derivates in y-direction and xy-direction and the approximations to them using box filters.

The evaluation system, to determine if a point is actually a point of interest, follows the Equation 2.20 and if $c > 0$ then it is a possible point of interest.

$$c(x,y,\sigma) = D_{x,x}(\sigma)D_{y,y}(\sigma) - (0.9D_{y,y}(\sigma))^2 \approx [H(x,y,\sigma)] \tag{2.20}$$

$D_{x,x}$, $D_{x,y}$ and $D_{y,y}$ are the approximations previously calculated.

2.3.4.5. FAST

FAST (Features from Accelerated Segment Test) is an algorithm proposed in 2006 by Rosten and Drummond in their paper [15]. Other feature detectors are good, but they are nor fast enought and this is a problem in real time applications. FAST algorithm follows the next steeps:

1. One pixel p of the image is selected, and we will call to its intensity I_p.
2. Threshold value t is selected.
3. It is considered a circle of 16 pixels around p as we see in the image Figure 2.19.
4. The pixel is considerated as a corner if there is a number of pixels whose intensity is brighter than $I_p + t$ or darker than $I_p - t$.

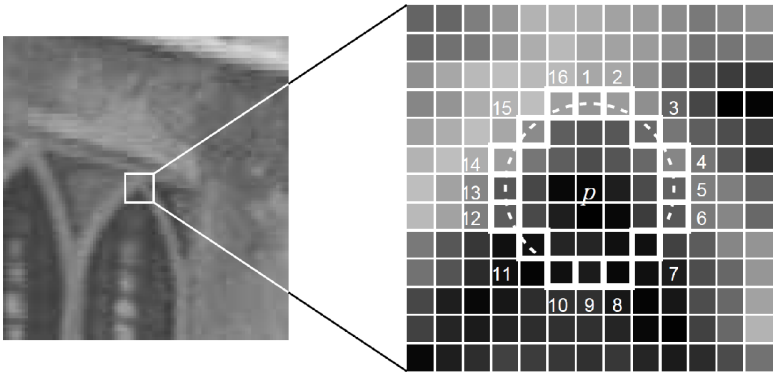


Figure 2.19: Pixels around P in corner detector FAST.

This detector itself has a high performance, but there are several weaknesses, one of which is that it finds many adjacent features, which is a problem in the matching process. We can see this problem in Figure 2.20 that is the test of FAST detector in an image of KITTI dataset.

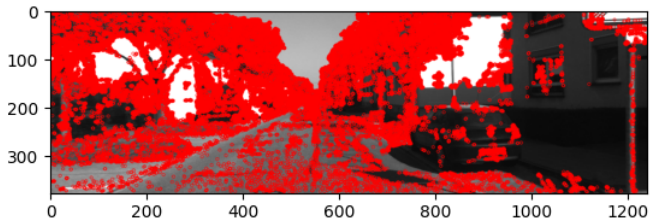


Figure 2.20: FAST Detector used in an image from KITTI Dataset.

2.3.5. Feature Description

A feature is defined by its pixel coordinates $[u, v]$ in the image frame. A descriptor is a n-dimensional vector associated to each feature, and it contains a summary of the image information in the feature vicinity.

Descriptors must comply with the following characteristics to be considered good:

- Repeteability: It means that, even if there are some changes in position, scale or illumination, the same point of interest must have approximately the same descriptor.
- Distinctiveness: Two nearby features must not have a similar descriptor.
- Efficiency: A descriptor must be computed in a reasonable computation time.

Throughout this section the main feature descriptors will be explained.

2.3.5.1. SIFT Descriptor

As we explain in subsection 2.3.4 SIFT (Scale Invariant Feature Transform) is a good feature detector and descriptor. In this section we explain SIFT feature descriptor algorithm.

- Around each feature SIFT takes a window of 16x16 pixels.
- This window is divide into 4 cells of 16 pixels.
- Edge orientation is calculated for each cell, and if the edge value does not pass a predefined threshold it is considered as weak, and it is suppressed.
- Finally an 32-dimensional histogram of orientation is construct for each cell and then concatenate to get 128 dimensional descriptor.

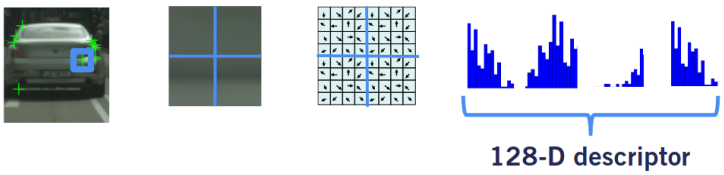


Figure 2.21: SIFT Descriptor Process

2.3.5.2. SURF Descriptor

SURF feature description process is divided in tow steeps. The first one is to identify a reproducible orientation for each keypoint and the second one is to construct the feature descriptor.

To orientation assignment Haar-wavelet response in x-direction and y-direction is calculated in a circular neighborhood of radius 6s, where s is the scale at the keypoint was detected. Then, the sum of vertical and horizontal responses is calculated in an area, the orientation is changed. Finally, the orientation with largest sum value is selected as the main orientation of the feature descriptor.

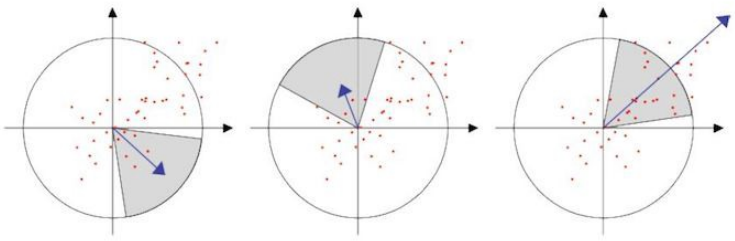


Figure 2.22: SURF Descriptor Orientation Assignment.

Once the orientation has been calculated, the feature descriptor is built. First, a square region with size 20s is construct around the keypoint and oriented along the orientation already calculate.

Each region is now divided into 4x4 square sub-regions. For each sub-region there are 25 uniformly distributed sampling points and circular areas with size 2s are selected around each point, as we can see in the picture Figure 2.23, where d_x is the Haar Wavelet response in horizontal direction and d_y the Haar Wavelet response in vertical direction.

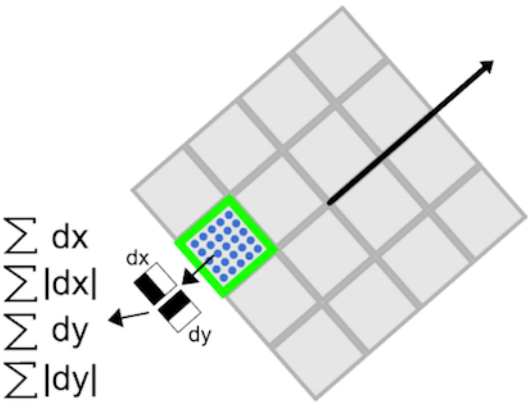


Figure 2.23: SURF Descriptor Components

Those response are sum in each subregion and form a first set of entries to the feature vector. Finally each sub-region has a 4-dimensional descriptor vector:

$$V = (\sum d_x, \sum d_y, |\sum d_x|, |\sum d_y|) \tag{2.21}$$

Finally, a 64-dimensional descriptor is construct.

2.3.5.3. BRIEF Descriptor

As we saw in previous sections SIFT descriptor is a 128-dimensional vector. This means that it occupies a memory of 512 bytes. SURF descriptor is a 64-dimensional vector, so the occupied memory is 256 bytes. When we try to construct the descriptors of thousand features a lot of memory is occupied, so this is not feasible for resource-constraint applications especially for embedded systems. In addition, the more memory we use, the longer the feature matching process will take. There are some methods to convert these descriptors in floating point numbers to binary strings.

BRIEF (Binary Robust Independent Elementary Features) [16] provides a shortcut to find the binary strings directly without finding descriptors.

It takes smoothed patch and select a set of location pairs- Then the intensity of the pixels of these location pairs is compared, for example, if we have two pairs p and q, if I(p) < I(q) the result is 1, else, the result is 0. This comparison is applied for all pairs and finally, a n-dimensional bit-string is obtained.

2.3.5.4. ORB Descriptor

ORB (Oriented FAST and Rotated BRIEF) is a fusion between the FAST feature detector and the BRIEF descriptor with some modifications.

First at all ORB uses FAST detector to detect the keypoints, then, it uses Harris Corner score to keep only the most visual and robust keypoints. To find the keypoint orientation ORB considers a region centred on the keypoint. In this region it calculates the intensity weighted centroid. Then, the orientation is given by the direction of the vector connecting the centre of the keypoint to the centroid.

At the end, ORB algorithm calculates the rotation matrix of each keypoint based on the orientation of each feature and drives the descriptor according to this orientation.

2.3.6. Feature Matching

Using the descriptors described in subsection 2.3.5 we can match features across one or more images. This process is called Feature Matching and it plays a very important role in Visual Odometry, as by observing the changes in features from one image to another, we can estimate motion.

The Feature Mathing process consists of finding the best match between descriptors of different images.

2.3.6.1. Brute-Force Matcher

Brute Force Matching is a simple solution for this problem. First at all a distance function is defined. It compares the distance between two descriptos. After that, for every feature in the first image, it computes the distance function and the closest feature will be considered a match. The most common distance functions are:

- Sum of Squared Differences (SSD): This is the most common distance function to compare descriptors. SSD is sensitive to large variations in the descriptors, but it fails in small ones.

$$d(f_i, f_j) = \sum_{k=1}^{D} (f_{i,k} - f_{j,k})^2 \tag{2.22}$$

- Sum of absolute differences(SAD): It has the same sensitivity for large and small descriptors.

$$d(f_i, f_j) = \sum_{k=1}^{D} |f_{i,k} - f_{j,k}| \tag{2.23}$$

- Hamming Distance: It is used when all descriptors are binary values.

$$d(f_i, f_j) = \sum_{k=1}^{D} XOR(f_{i,k}, f_{j,k}) \tag{2.24}$$

But if we select the closest feature as a match, we can find wrong matches because, if a feature is in the first image, but it is not found in the second one, the match should not exist.To solve this problem, a distance threshold is defined. Therefore, if the distance between the descriptors is greater than this parameter, then the match is rejected.

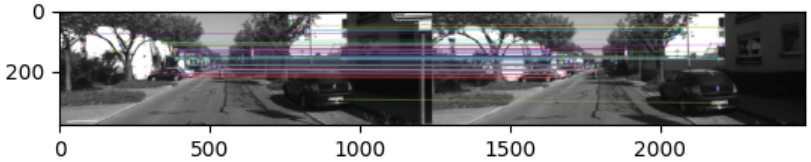


Figure 2.24: BF Feature Matching Example

2.3.6.2. FLANN Based Matcher

The main problem of Brute Force Matcher is that it is computacionally expensive. FLANN (Fast Library for Approximate Nearest Neighbors) [17] is a faster solution for matching problem but it slightly sacrifices the accuracy obtained with BFM. It contains a collection of algorithms, kd-tree based, for fast nearest neighbor search in large datasets and for high dimensional features.

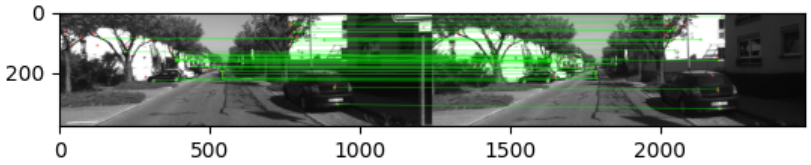


Figure 2.25: FLANN Based Matcher Example

2.3.7. Trajectory Estimation

Motion estimation is the most important task in Visual Odometry. Between two consecutive frames I_{k-1} and I_k the camera motion T_k will be estimated:

$$T_k = \begin{bmatrix} R_{k,k-1} & tk, k-1 \\ 0 & 1 \end{bmatrix} \tag{2.25}$$

Where $R_{k,k-1}$ is the rotation matrix and $t_{k,k-1}$ is the translation matrix.

Once we have the feature matches f_{k-1} and f_k between two consecutive frames I_{k-1} and I_k we can to estimate the camera motion in that time period. The way

- 2D-2D Motion Estimation: The features of both frames are defined in the image coordinate frame.
- 3D-3D Motion Estimation: The features of both frames are defined in the world coordinate frame.
- 3D-2D Motion Estimation: The features of the I_{k-1}, f_{k-1}, are defined in the world coordinate frame and the features f_k are defined in the image coordinate frame.

We will focus on 3D-2D estimation. Thanks to the stereo visual system, we can estimate the 3D coordinates of the features as we explained in subsection 2.3.3, because the feature matching has already been performed we have the correspondences in 2D of the f_{k-1}. This information is what we will use to estimate the matrix $[R|t]$.

The way we have to estimate the movement is using the algorithm "Perspective N Point" (PnP). knowing the 3D feature locations, their projections in 2D and the intrinsic matrix calibration, PnP can resolve the extrinsic transformations of the camera.

To estimate an initial guess of [R|t], PnP uses Direct Lineal Transform (DLT). This method requires a lineal model and it constructs a set of lineal equations to solve it using standard methods as SVD.

However, the results are not lineal so a non-lineal optimization technique is used, for example Levenberg-Marquardt algorithm.

PnP algorithm requires at least three features to solve R and t. Taken just three features, four possible results exist, and a fourth new point is used to decide which result is valid.

RANSAC (RANdom SAmple Consensus) is an algorithm that was proposed in 1981 for robust estimation of the model parameters in a presence of outliers that is, data points which are noisy and wrong. The algorithm follows the next steps:

1. Initialization: a set X of N point correspondences is taken.
2. The following steps are repeated until the maximum number of interactions is reached:

- A subset of S points of X is randomly selected.
- Using the information from S, a model M is constructed.
- The remaining points are evaluated with the model and the error is measured.
- Those points whose distance to the model is less than a predefined threshold will be part of a set of inliers that is stored.

3. The set with the highest number of inliers is selected.
4. The model is estimated using this inliers.

RANSAC algorithm can be incorporated into PnP. Assuming the transformation generated by PnP in four points is our model, we will use a new set of feature matches to evaluate the model and check the percentage of inliers that result to confirm the validity of the point matches selected.

Concatenating these single movements allows the recovery of the full trajectory of the camera.

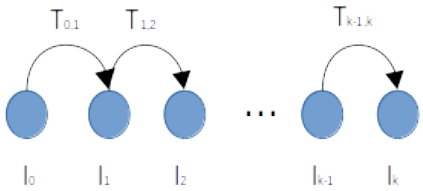


Figure 2.26: Concatenation of motion estimations between consecutive frames.

Chapter 3

KITTI Benchmarking

The KITTI Vision Benchmark [18] is a project of Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago.

The main objective of the platform is to develop a new real-world computer vision challenges applied to autonomous driving. Their task of interest are stereo, optical flow, Visual Odometry, 3D object detection and 3D tracking. For this proposal, they equiped a standard station waegon with two high-resolution color and grey-scale video cameras. They provide the ground truth that is estimated using a GPS localization system and a Velodyne laser scanner. Their datasets are captured by driving around the mid-size city of Karlsruhe, in rural areas and on highways.

3.1. Setup

The recording platform is a Volkswagen Passat B6 that has been modified with actuators for the acceleration and brake pedals and the steering wheel. The data is recorded using an eight core i7 computer equipped with a RAID system, running Ubuntu Linux and a real-time database. The following sensors are used:

- 1 Inertial Navigation System (GPS/IMU): OXTS RT 3003.
- 1 Laserscanner: Velodyne HDL-64E.
- 2 Grayscale cameras, 1.4 Megapixels: Point Grey Flea 2 (FL2-14S3M-C).
- 2 Color cameras, 1.4 Megapixels: Point Grey Flea 2 (FL2-14S3C-C).
- 4 Varifocal lenses, 4-8 mm: Edmund Optics NT59-917.

Figure 3.1 shows an schematic of the sensor setup and the dimensions of the vehicle.

3.2. Odometry Benchmark

The Odometry Benchmark consists of twenty-two stereo sequences. Of these twenty-two sequences KITTI provides eleven sequences with ground truth for training and eleven sequences without ground truth for evaluation. Apart from colour or greyscale images, KITTI provides the camera calibration data, so if KITTI dataset is used, it is not necesary to perform the camera calibration process.

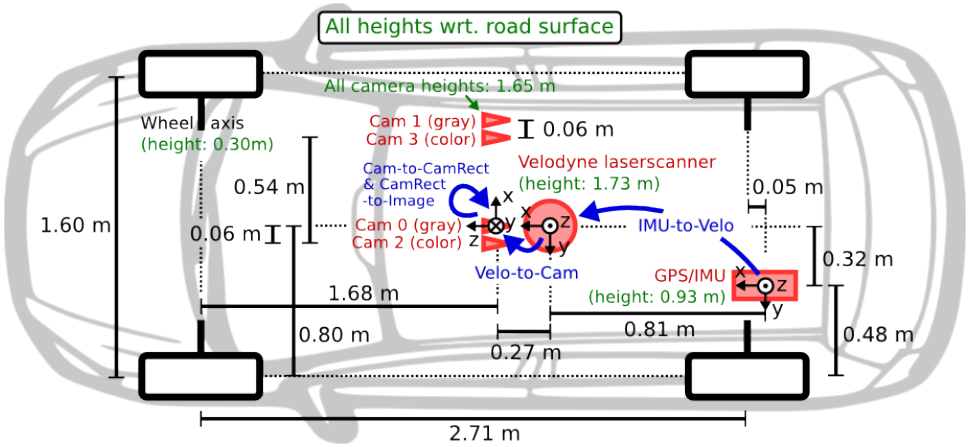


Figure 3.1: KITTI setup top view.

For evaluation KITTI computes the traslational and rotational errors, the evaluation table (Table 3.1) ranks the methods according to the average of those values.

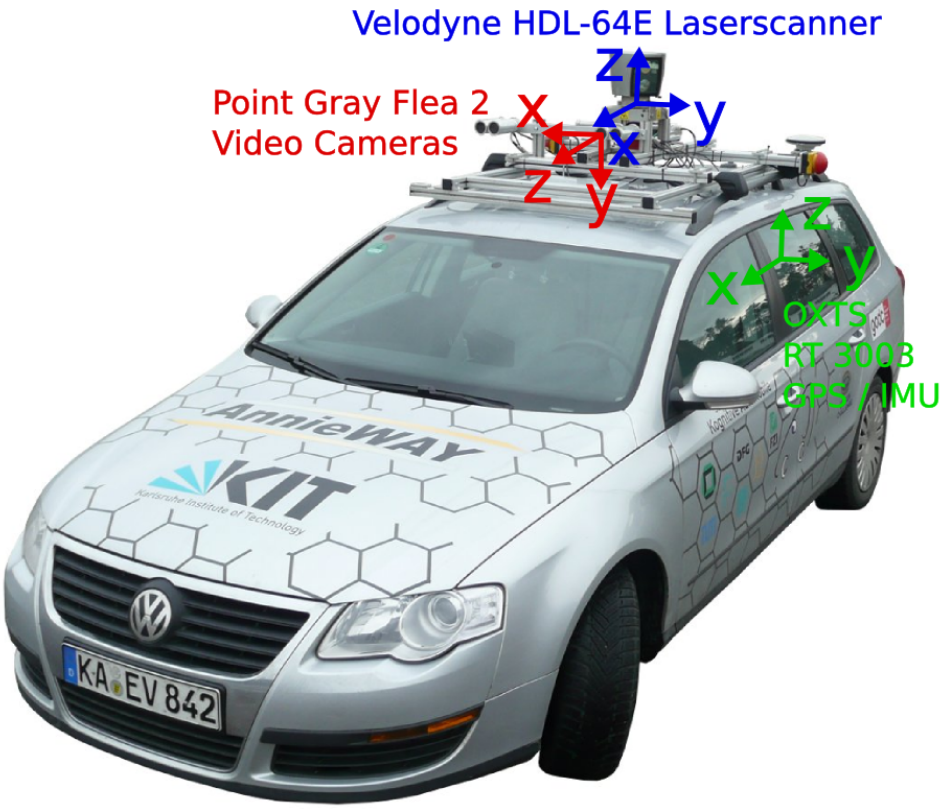


Figure 3.2: Vehicle used by KITTI to record the sequences.

Method	Setting	Translation	Rotation	Runtime
SOFT2	st	0.53 %	0.0009 [deg/m]	0.1 s / 4 cores
V-LOAM	la	0.54 %	0.0013 [deg/m]	0.1 s / 2 cores
LOAM	la	0.55 %	0.0013 [deg/m]	0.1 s / 2 cores
TVL-SLAM+	st la	0.56 %	0.0015 [deg/m]	0.3 s / 1 core
GLIM	la	0.59 %	0.0015 [deg/m]	0.1 s / GPU
HELO	la	0.61 %	0.0018 [deg/m]	0.1 s / 8 cores
zPICP	la	0.62 %	0.0016 [deg/m]	0.1 s / 1 core
HMLO-whu	la	0.63 %	0.0014 [deg/m]	0.2 s / 1 core
HMLO	la	0.64 %	0.0015 [deg/m]	0.2s / 1 core
filter-reg		0.65 %	0.0016 [deg/m]	0.1 s / 1 core
SOFT-SLAM	st	0.65 %	0.0014 [deg/m]	0.1 s / 2 cores
wPICP	la	0.65 %	0.0018 [deg/m]	0.1 s / 1 core
MULLS	la	0.65 %	0.0019 [deg/m]	0.08 s / 4 cores
MULLS-SLAM	la	0.67 %	0.0020 [deg/m]	0.1 s / 4 cores
PICP	la	0.67 %	0.0018 [deg/m]	0.05 s / 1 core
ELO	la	0.68 %	0.0021 [deg/m]	0.005 s / GPU
IMLS-SLAM	la	0.69 %	0.0018 [deg/m]	1.25 s / 1 core
MC2SLAM	la	0.69 %	0.0016 [deg/m]	0.1 s / 4 cores
FLOAM	la	0.71 %	0.0022 [deg/m]	0.05 s / 4 cores
ICD		0.71 %	0.0018 [deg/m]	0.2 s / 8 cores
ISC-LOAM	la	0.72 %	0.0022 [deg/m]	0.1 s / 4 cores
PSF-LO	la	0.82 %	0.0032 [deg/m]	0.2s / 4 cores
RADVO	st	0.82 %	0.0018 [deg/m]	0.07 s / 1 core
LG-SLAM	st	0.82 %	0.0020 [deg/m]	0.2 s / 4 cores
DSV-LOAM	la	0.83 %	0.0025 [deg/m]	0.1 s / 4 cores
F-pose		0.83 %	0.0024 [deg/m]	0.02 s / 1 core
RotRocc+	st	0.83 %	0.0026 [deg/m]	0.25 s / 2 cores
LIMO2_GP	la	0.84 %	0.0022 [deg/m]	0.2 s / 2 cores
CAE-LO	la	0.86 %	0.0025 [deg/m]	2 s / 8 cores
GDVO	st	0.86 %	0.0031 [deg/m]	0.09 s / 1 core
LIMO2	la	0.86 %	0.0022 [deg/m]	0.2 s / 2 cores
CPFG-slam	la	0.87 %	0.0025 [deg/m]	0.03 s / 4 cores
SOFT	st	0.88 %	0.0022 [deg/m]	0.1 s / 2 cores
RotRocc	st	0.88 %	0.0025 [deg/m]	0.3 s / 2 cores
D3VO		0.88 %	0.0021 [deg/m]	0.1 s / 1 core
PNDT LO	la	0.89 %	0.0030 [deg/m]	0.2 s / 8 cores
Lidar-DSO		0.90 %	0.0027 [deg/m]	0.1 s / 4 cores
DVSO		0.90 %	0.0021 [deg/m]	0.1 s / GPU
LIMO	la	0.93 %	0.0026 [deg/m]	0.2 s / 2 cores
Stereo DSO	st	0.93 %	0.0020 [deg/m]	0.1 s / 1 core
OV2SLAM	st	0.94 %	0.0023 [deg/m]	0.01 s / 1 core
PLO	la	0.95 %	0.0021 [deg/m]	0.1 s / 1 core
OV2SLAM	st	0.98 %	0.0023 [deg/m]	0.01 s / 8 cores
ROCC	st	0.98 %	0.0028 [deg/m]	0.3 s / 2 cores
SuMa-MOS	la	0.99 %	0.0033 [deg/m]	0.1s / 1 core
IsaacElbrusSLAM	st	1.02 %	0.0019 [deg/m]	0.018 s / 3 cores
MESVO		1.03 %	0.0033 [deg/m]	0.5 s / 1 core
SuMa++	la	1.06 %	0.0034 [deg/m]	0.1 s / 1 core
ULF-ESGVI		1.07 %	0.0036 [deg/m]	0.3 s / GPU and CPU
cv4xv1-sc	st	1.09 %	0.0029 [deg/m]	0.145 s / GPU
VINS-Fusion	st	1.09 %	0.0033 [deg/m]	0.1s / 1 core
MonoROCC	st	1.11 %	0.0028 [deg/m]	1 s / 2 cores
icp_lo		1.14 %	0.0037 [deg/m]	0.05 s / 1 core
DEMO	la	1.14 %	0.0049 [deg/m]	0.1 s / 2 cores
ORB-SLAM2	st	1.15 %	0.0027 [deg/m]	0.06 s / 2 cores

Table 3.1: KITTI evaluation table - 25 august 2021.

Chapter 4

Comparison of Methods in KITTI

4.1. Stereo Visual Odometry KITTI

As we saw in chapter 3 thanks to KITTI Benchmarking [18] we can evaluate different odometry methods. In Table 4.1 we observe the current best methods of stereo visual odometry. Throughout the chapter we will explain some of them.

Method	**Setting**	**Translation**	**Rotation**	**Runtime**
SOFT2	st	0.53 %	0.0009 [deg/m]	0.1 s / 4 cores
SOFT-SLAM	st	0.65 %	0.0014 [deg/m]	0.1 s / 2 cores
RADVO	st	0.82 %	0.0018 [deg/m]	0.07 s / 1 core
LG-SLAM	st	0.82 %	0.0020 [deg/m]	0.2 s / 4 cores
RotRocc+	st	0.83 %	0.0026 [deg/m]	0.25 s / 2 cores
GDVO	st	0.86 %	0.0031 [deg/m]	0.09 s / 1 core
SOFT	st	0.88 %	0.0022 [deg/m]	0.1 s / 2 cores
RotRocc	st	0.88 %	0.0025 [deg/m]	0.3 s / 2 cores
Stereo DSO	st	0.93 %	0.0020 [deg/m]	0.1 s / 1 core
OV2SLAM	st	0.94 %	0.0023 [deg/m]	0.01 s / 1 core
OV2SLAM	st	0.98 %	0.0023 [deg/m]	0.01 s / 8 cores
ROCC	st	0.98 %	0.0028 [deg/m]	0.3 s / 2 cores
IsaacElbrusSLAM	st	1.02 %	0.0019 [deg/m]	0.018 s / 3 cores
cv4xv1-sc	st	1.09 %	0.0029 [deg/m]	0.145 s / GPU
ORB-SLAM2	st	1.15 %	0.0027 [deg/m]	0.06 s / 2 cores
RPGSLAM	st	1.16 %	0.0026 [deg/m]	0.1 s / 2 cores
NOTF	st	1.17 %	0.0035 [deg/m]	0.45 s / 1 core
FPS-DS-SLAM	st	1.18 %	0.0025 [deg/m]	0.07 s / 2 core
S-PTAM	st	1.19 %	0.0025 [deg/m]	0.03 s / 4 cores
S-LSD-SLAM	st	1.20 %	0.0033 [deg/m]	0.07 s / 1 core
VoBa	st	1.22 %	0.0029 [deg/m]	0.1 s / 1 core

Table 4.1: Stereo Visual Odometry Methods in KITTI Benchmarking.- 25 august- 2021

4.2. SOFT-SLAM

As we can see in Table 4.1, SOFT-SLAM method [19], developed at the University of Zagreb (Croatia), is currently one of the first stereo visual methods evaluated in the KITTI benchmarking.

Figure 4.1 shows the scheme of the algorithm.

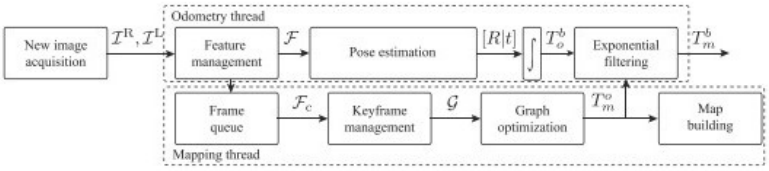


Figure 4.1: SOFT-SLAM diagram.

4.2.1. Algorithm

SOFT-SLAM proposes a solution consisting in the separation of two threads, one for odometry and one for mapping. The two threads are finally connected through a filter and the mapping output is merged with the information obtained from the odometry.

4.2.1.1. Odometry Thread

There are two verions ot the algorithm used, one of them uses the information provided by an IMU sensor, and the other one is only vision based. We will only explain the one which is entirely visual odometry.

This method is based on careful feature selection and tracking and it will be explained in section 4.3.

4.2.1.2. Mapping Thread

Mapping thread is independent of Odometry thread, but it uses the feature managment process output.

The main reason of the independence between the two threads is because Mapping thread is used for drift evaluation. Assuming that drift is raised on the line of a Wiener process, once an output from this thread is available, it can be used regardless of how old the frame is where that mapping was made. The Mapping Thread is shown in Figure 4.2

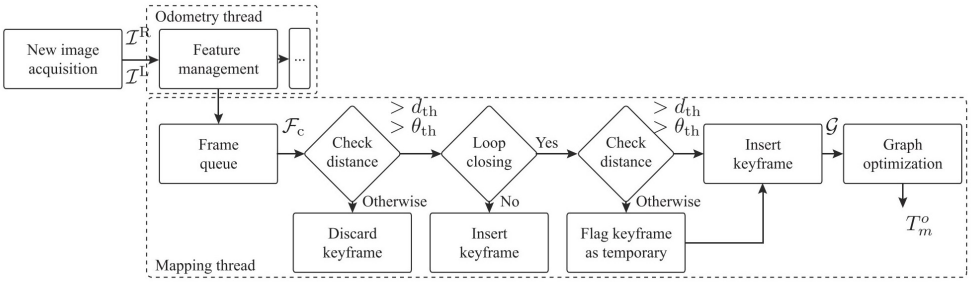
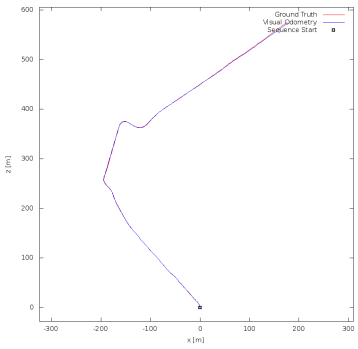


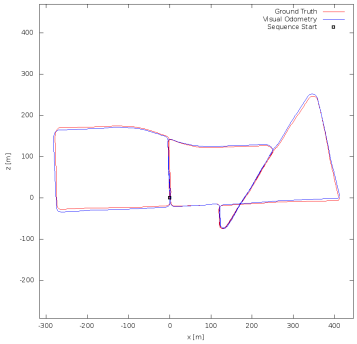
Figure 4.2: Mapping Thread diagram.

4.2.2. Resuls

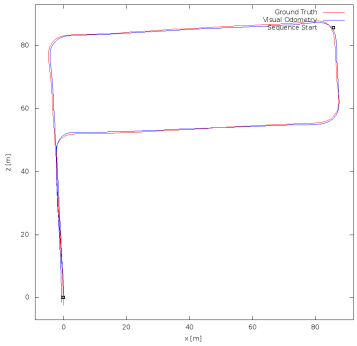
SOFT-SLAM algorithm is evaluated in KITTI benchmarking and it has a translation error of 0.65% and a rotation error of 0.0014 [deg/m]. Its execution time is 100ms. The following image shows the trajectories 11, 13 and 15 of the KITTI dataset estimated with the SOFT-SLAM method.



(a) Sequence 11.



(b) Sequence 13.



(c) Sequence 15.

Figure 4.3: KITTI Sequences estimated by SOFT-SLAM.

4.3. SOFT

SOFT method [20] is an algorithm developed at the University of Zagreb (Croatia) and is the preliminary work to the SOFT-SLAM method. It is based on careful feature selection and the subsequent separate estimation of translation and rotation. The following sections explain the algorithm followed by the method.

4.3.1. Algorithm

Figure 4.4 shows the Algorithm Diagram, and its parts will be explained in the next sections.

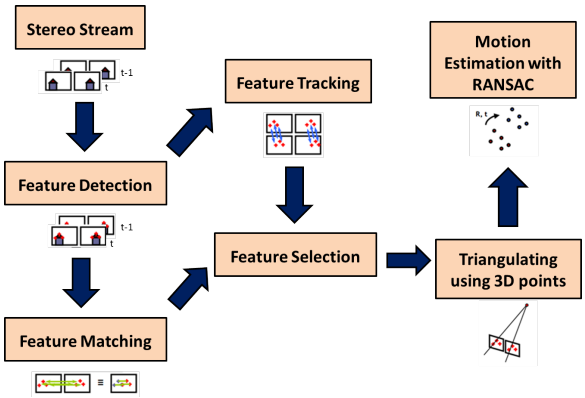


Figure 4.4: SOFT Algorithm Diagram

4.3.1.1. Feature Matching

The first step of this algorithm is the extraction of features from both the image on the left and the image on the right.

Once the features have been extracted "Non-Maximum-Suppresion" is applied. To perform the feature matching "Sum of absolute differences" (SAD) is applied. This method is simple but susceptible to outliers. To reject the outliers "Circular Matching" is used.

Circular matching implies that each feature has to match between the left image and the right image in two consecutive times as Figure 4.5 shows.

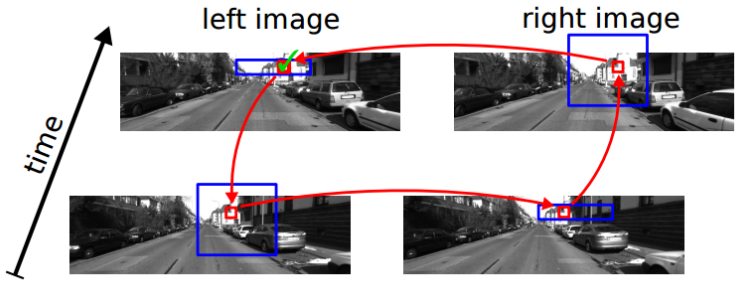


Figure 4.5: Circular Matching.

If this circle does not close, then that feature is considered an outlier and therefore rejected. If the cycle is closed, an additional check is performed with the cross-correlation (NCC) 25x25 pixels around the position of each feature point. However, this is a slow process, to maintain real-time performance, NCC is used only after SAD Circular Matching. The remaining outliers will be discarded by applying RANSAC.

4.3.1.2. Feature Selection

Using only a small set of well-selected feature points reduces the computational time of the algorithm, allowing it to be applied in real time.

For an accurate estimation of the movement of the vehicle, it is necessary that these feature points are evenly distributed. Therefore, the image is divided into rectangles of 50x50 pixels. In each bucket there is a limited number of inliers and for their selection the following steps are performed.

- The points are divided in four classes: "corner-max", "corner-min", "blob-max" y "blob-min"
- These points are ordered by strength in each class.
- The strongest feature point is placed at the end of the list.
- The previous step is repeated until all feature points are in the list.
- Only a number of feaures is taken from the list, they are selected to the trajectory estimation.

4.3.1.3. Feature tracking

Each feature point is represented by the following properties:

- ID,

- age,
- refined current position in the image,
- feature strength,
- belonging class,
- initial descriptor.

When a feature is found in the next image, the position of the pixel is updated with respect to the previous one and the age is increased by one. This position is stored and will be used in the next image. As long as a feature remains, its initial descriptor will be used, reducing the error. Feature points that have been present the longest are unlikely to be outliers, so the oldest one should always be chosen in the next interaction. However, very strong feature points can also appear, therefore the feature selection is estimated as follows:

$$select(f1, f2) = \begin{cases} stronger(f1, f2) & \text{if } age(f1) = age(f2) \\ older(f1, f2) & \text{if } age(f1) \neq age(f2) \end{cases} \tag{4.1}$$

4.3.1.4. Trajectory Estimation

This process is divided into two parts, rotation estimation and translation estimation.

- **Rotation**: The "five point" method normally used in monocular visual odometry is employed to estimate rotation. However, it has been found to give better results for rotation. This method is used in conjunction with RANSAC.
- **Traslation**: The "three point" method is used to estimate the translation. Two 3D point clouds are reconstructed, one from the previous time and one from the concurrent time. Triangularisation is then performed between the two. And it is calculated by iterative minimisation regarding the translation only.

4.3.2. Results

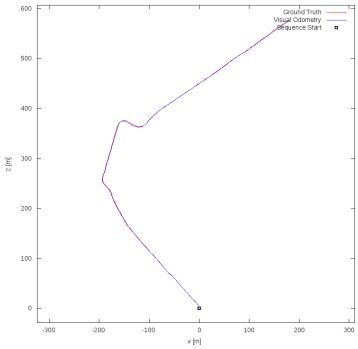
SOFT algorithm is evaluated in KITTI benchmarking and it has a translation error of 1.03% and a rotation error of 0.0029 [deg/m]. Its execution time is 100ms. Figure 4.6 shows the trajectories 11, 13 and 15 of the KITTI dataset estimated with the SOFT method.

4.4. OV2SLAM

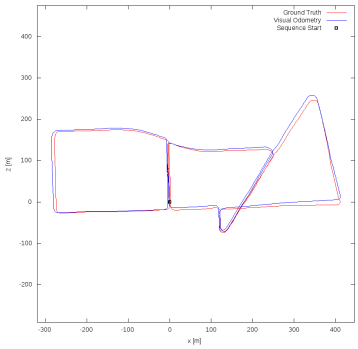
OV2SLAM [21] method is an open source algorithm that can be used for both monocular and stereo applications. It combines numerous recent contributions from visual odometry within an efficient multi-treated architecture.

4.4.1. Algorithm

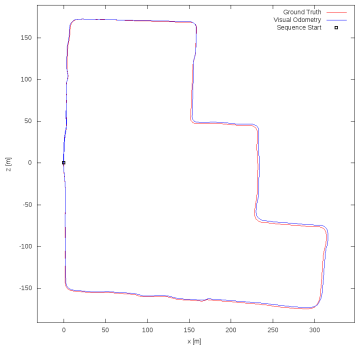
Figure 4.7 shows the scheme of OV2SLAM algorithm. It has three main threads: "Visual front-end Thread", "Mapping Thread" and "State Optimization Thread".



(a) Sequence 11.



(b) Sequence 13.



(c) Sequence 15.

Figure 4.6: KITTI Sequences estimated by SOFT.

4.4.1.1. Front-End Thread

The main task of this thread is to estimate the vehicle pose in real time. Although OV2SLAM is a method that can be used in both stereo and monocular applications, Front-End Thread uses only the frames provided by the left camera. It has five different tasks that we explain below.

1. **Image pre-procesing**: When a new image is received, contrast is applied to increase the dynamic range and limit intensity changes.

2. **Keypoint tracking**: Keypoint tracking is performed by means of a guided coarse-to-fine optical flow method, Keypoints are tracked individually using a pyramidal implementation of the inverse compositional Lucas-Kanade (LK)

   - 2D Keypoints: The position is set to its previous position and a four-level pyramid is used.
   - 3D Keypoints: When triangulation has already been completed, its initial position is computed using its projection, its 3D position and the prediction of the current position by a constant velocity motion model.

   Finally the tracked keypoints are updated and its coordinates are calculated from the intrinsic calibration of the camera.

3. **Outlier filtering**: To the outlier rejection a RANSAC filter is applied using the essential matrix and not the fundamental one (as in other cases). This is because the essential matix is more efficent and its estimation is robust. This matrix is estimated just from 3D keypoints.

4. **Pose estimation**: Pose estimation is performed by minimization of the 3D keypoints reprojection errors.This is a non-linear optimization and it is applied using Levenberg-Marquardt algorithm.If

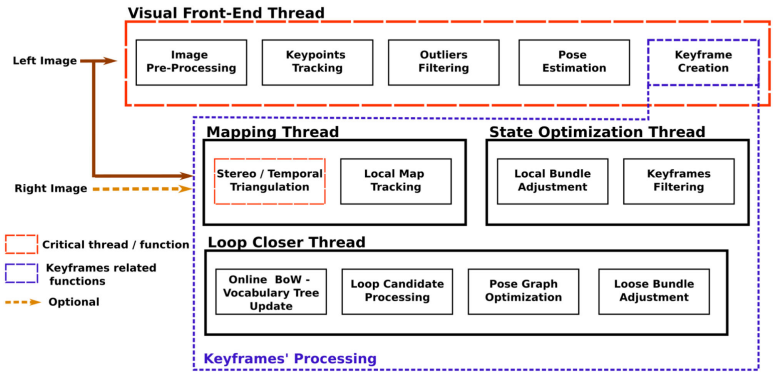


Figure 4.7: OV2SLAM scheme

the prediction is not correct, it is rejected and a new pose is sought by applied P3P RANSAC before error minimation.

5. **Keyframe Creation**: Finally, front-end thread is used to create the keyframes. If the required conditions are accomplished a new keyframe is created. Feature detection is performed by using Shi-Tomasi algorithm, after dividing the image in 35x35 buckets. BRIEF descriptors are then found for all keypoints.

4.4.1.2. Mapping Thread

This thread is in charge of processing each keyframe to create the 3D point cloud by triangulation to minimise the deviation. First the triangulation is performed and then the map traking until a new keyframe is available.

4.4.1.3. State Optimization Thread

A local Bundle Adjustment is performed to refine the keyframe and point cloud poses. Then a keyframe filtering is done in order to eliminate redundant keyframes.

4.4.2. Results

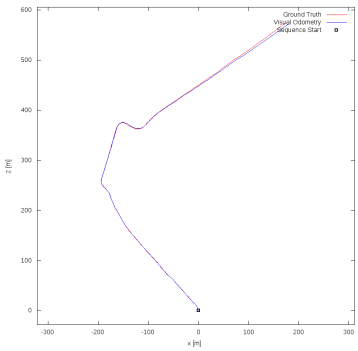
OV2SLAM algorithm is evaluated in KITTI benchmarking and it has a translation error of 0.94% and a rotation error of 0.0023 [deg/m]. Its execution time is 100ms. Figure 4.8 shows the trajectories 11, 13 and 15 of the KITTI dataset estimated with the OV2SLAM.

4.5. ORB-SLAM2

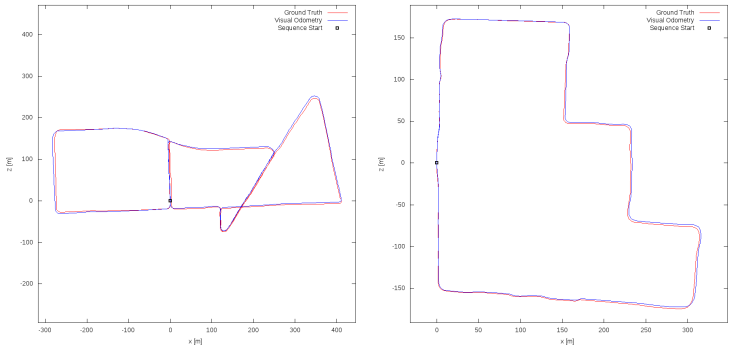
ORB-SLAM2 is a an Open-Source simultaneous localization and mapping (SLAM) system for Monocular, Stereo, and RGB-D Cameras.[21]

4.5.1. Algorithm

ORB-SLAM consists of three main threads working in parallel: tracking thread, loop-closing and local mapping. At the end, another thread is created to perform "bundle adjustment" in order to optimize the results. Figure 4.9 shows the architecture of the algorithm, which will be explained in more detail below.



(a) Sequence 11.



(b) Sequence 13. (c) Sequence 15.

Figure 4.8: KITTI Sequences estimated by SV2SLAM.

The system uses ORB feature points for tracking, mapping and place recognition. These feature points can be quickly extracted, which supports real-time execution.

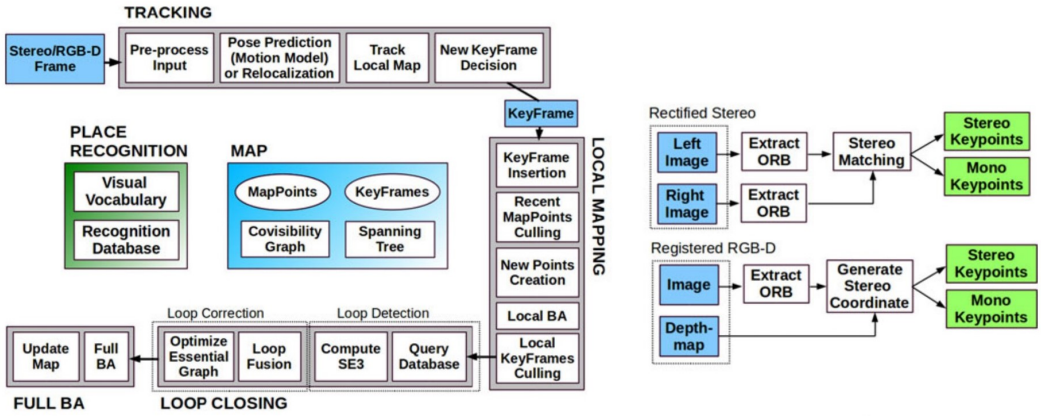


Figure 4.9: ORB-SLAM2 Diagram

4.5.1.1. Tracking thread

Tracking thread is the responsible of camera position location in each image. This thread decides when a new keyframe is required.

When the system loses tracking, there is a module that is in charge of representing a global relocation. Once an initial estimation of the position of the camera has been obtained and the feature points found

have been associated, the local map is retrieved using a covisibiliy graph of the keyframes that were stored.

This thread applied "motion-only bundle adjustment". **Motion-only bundle adjustment** optimizes the rotation and the position of the camera minimizing the reprojection error between the 3D points in the world coordinates and the keypoints.

4.5.1.2. Local Mapping Thread

The second thread is responsible for processing the new keyframes and rendering the local bundle adjustment to achieve an optimal reconstruction of the camera position environment. In this module, the keyframes are filtered by removing redundant keyframes.

Local mapping thread perform "local bundle adjustment" to optimize the map.

4.5.1.3. Loop-closing thread

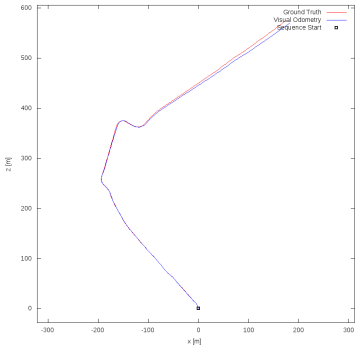
The third thread is in charge of looking for the existence of closed loops whenever a new keyframe is found. If it exists, this keyframe is incorporated into the global map graph.

This thread launches a fourth thread to perform full BA after the pose-graph optimization, to compute the optimal structure and motion solution **Full-bundle adjustment** is a particular case of the local bundle adjustment.

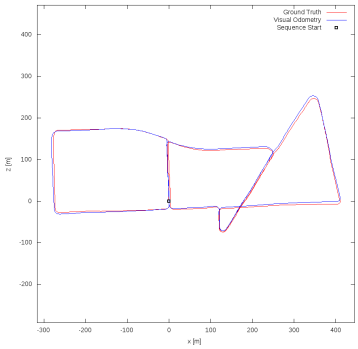
In this method all keyframes and points on the map are optimised except for the original keyframe, which is fixed.

4.5.2. Results

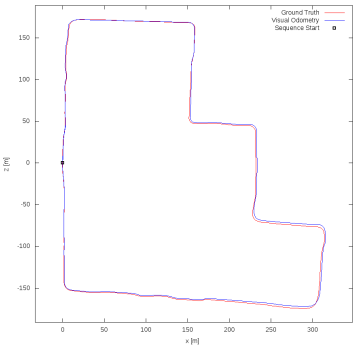
ORB-SLAM2 is evaluated in KITTI benchmarking and it has a translation error of 1.15% and a rotation error of 0.0039 [deg/m]. Its execution time is 60ms. Figure 4.10 shows the trajectories 11, 13 and 15 of the KITTI dataset estimated with the ORB-SLAM2.



(a) Sequence 11.



(b) Sequence 13.



(c) Sequence 15.

Figure 4.10: KITTI Sequences estimated by ORB-SLAM2.

4.6. Comparison of results in the studied algorithms.

In Table 4.2 we can see the traslation and rotation errors in the trajectories 11, 12, 13, 14 an 15 of KITTI dataset estimated by SOFT-SLAM, SOFT, OV2-SLAM and ORB-SLAM2.

Sequence	SOFT-SLAM		SOFT		OV2-SLAM		ORB-SLAM2	
	trel (%)	Rrel (deg/m)	trel (%)	Rrel (deg/m)	trel (%)	Rrel (deg/m)	trel (%)	Rrel (deg/m)
11	0.407	0.00191	0.584	0.00345	0.591	0.00295	0.904	0.00475
12	0.333	0.00131	0.587	0.00149	0.659	0.00187	0.717	0.00363
13	0.695	0.00357	0.926	0.00509	0.954	0.00405	0.805	0.00421
14	0.435	0.00786	0.337	0.00921	0.497	0.01151	0.37	0.0099
15	0.705	0.00245	0.848	0.00311	0.884	0.00341	0.869	0.00348

Table 4.2: Comparison of relative errors by sequence

Chapter 5

Algorithm

As shown in Figure 5.1 the algorithm that we have implemented follows the same schematic as we saw in section 2.3 but in this chapter the specific tools we used for each step will be explained. We will try to use the best techniques of the studied methods to obtain the best Visual Odometry model as possible.

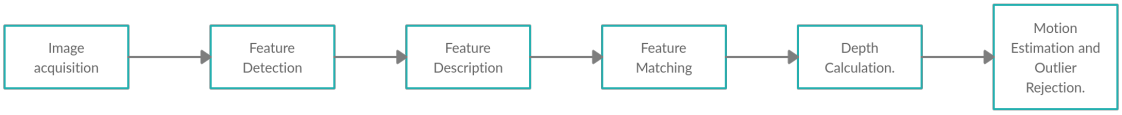


Figure 5.1: Algorithm Diagram

5.1. Feature Acquisition

We will use the images of the sequences that are part of the KITTI dataset. Because we are using this dataset, the camera calibration process will not be necessary because the projection matrices are provided.

5.2. Feature Detection and Description.

In this section, the feature detection of the images is performed. In order for the keypoints to be distributed uniformly throughout the image, the image is subdivided in a grid (60x60 pixels) and each section of the image that the kps are detected. Below, you can see the difference between kps detection when the grid is applied and when it is not. In order to obtain different results for further analysis we will use two different feature detectors: "Shi-Tomasi Corner Detector" [13] and "FAST Feature Detector"[15]. Their main characteristics are explained in chapter 2. In Figure 5.3 we can see the results of both feature detectors. Using both detectors we conclude that the FAST detector is faster than the Shi-Tomasi detector.

As a descriptor we use ORB-Descriptor, because it is more efficient than SIFT or SURF. Keep in mind that we have to get a limited number of features, because if we get too many, the computation time will be too long, but if we do not get enough, the results will not be good enough.

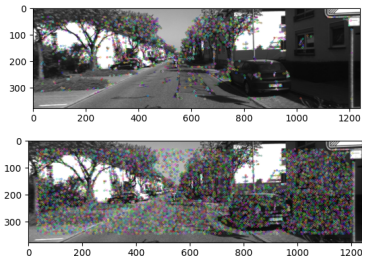
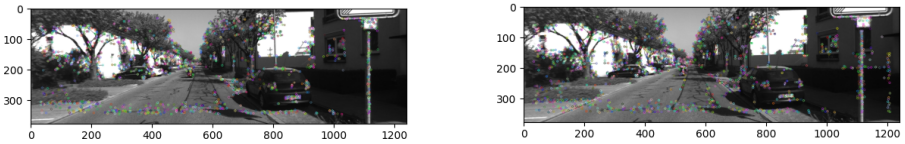


Figure 5.2: Feature Detection without dividing the image (first image) and dividing the image (second image).



(a) FAST Feature Detection.

(b) Shi-Tomasi Corner Detection.

Figure 5.3: Feature Detection

5.3. Feature Matching

After test our algorithm using FLANN Based Matcher and Brute-Force Matcher, we decided to choose the second one, because although the computation time is slightly longer, the results obtained are better.

OpenCV implementes the function "knnMatch()" that returns k best matches. Then, we applied a ratio test explained by D.Lowe in his paper [5]. We set k = 2, then, for every feature in the first image we obtain the two best matches, if the distance between the two matches is sufficiently different, then we keep the match, but if it is not, the keypoint is eliminated.

Brute-Force Matcher is used to perform Feature matching. To eliminate outliers, a filter has been used. It calculates the angle between the keypoints that have been matched. If this angle is bigger than a specific threshold, it is taken as an outlier, and therefore, it is not taken into account when estimating the position.

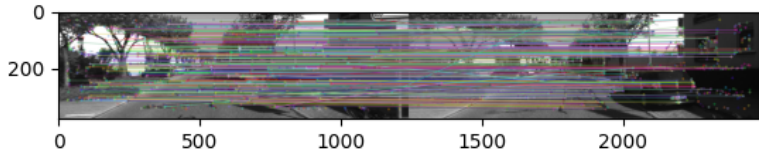


Figure 5.4: Matches without filtering.

5.4. Motion Estimation

For pose estimation, stereo Visual Odometry is used, however, for rotation estimation, the results of monocular visual odometry have been found to be better, as we see in section 4.3. Therefore, a distinction is made between the two cases:

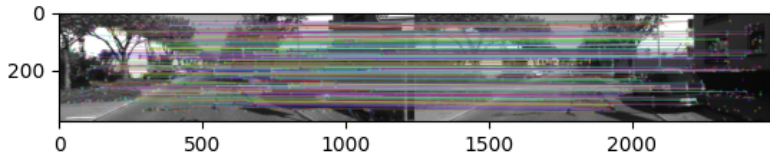


Figure 5.5: Matches with angle filter.

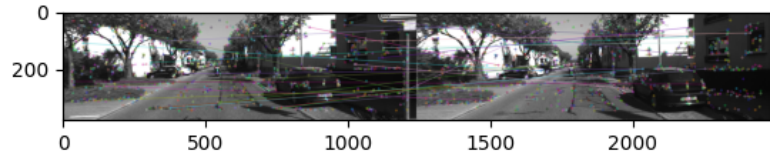


Figure 5.6: Outliers rejected.

5.4.1. Rotation

As we see in the algorithm SOFT (section 4.3) we obtain better results when estimate the rotation only using the frames from the left camera.

To accomplish this task we first calculate the essential matrix. To compute this matrix we decided to use RANSAC method. OpenCV [12] implements the function "findEssentialMat()" to calculate Essential Matrix. After this step we have to recover the relative camera rotation from this essential matrix. To do this we use the function "recoverPose()".

5.4.2. Translation

As we explain in subsection 2.3.7 there are three ways to estimate the motion, 2D-2D, 3D-3D and 3D-2D. 2D-2D estimation is usually used in monocular systems, this is because depth information is not available to calculate 3D-points, as we are using a stereo system, we can obtain the 3D points. The main reason to use 3D-2D estimation instead 3D-3D is because 3D-3D motion estimation methods will drift much more quickly than 3D-2D methods.

To use 3D-2D estimation we have to calculate the 3D-points. To achieve this we perform Depth estimation, as we explained in subsection 2.3.3. We take the keypoints from the image 1, then, using depth estimation, we take calculate the coordinate z of each one, in this way we obtain the 3D keypoints. Finally, we employ the fuction "solvePnPRansac()" implemented by OpenCV [12], this function finds the pose from 3D-2D point correspondences using the RANSAC Scheme.

Chapter 6

Results

6.1. KITTI Evaluation.

We test our algorithm in KITTI Benchmarking. First at all, to validate the algorithm we used the test sequences (0-11), in this way, specific parameters could be adjusted to obtain better results. After this test, we evaluate the algorithm in KITTI Benchmarking. To carry out this task it was necessary to register on the KITTI Benchmarking website. There, you were asked to provide a short description of your project in order to be allowed to evaluate it. Once this permission was granted, it was necessary to upload the results obtained in the estimation of the evaluation sequences (12-21).

6.1.1. KITTI Results

When the algorithm has been evaluated, KITTI gives us the average of the results, and provides us with the specific results of the first five evaluation sequences. We obtain a translation error of 4% and a rotation error of 0.01[deg/m]. The execution time takes 108ms (1 core 3GHz).

As we can observe in Figure 6.1 the translation error tends to increase with the length of the trajectory, therefore, better results are obtained when the path length is shorter. We can see, specially in Figure 6.6 or Figure 6.10, the drift error increases considerably on turns.

We just can see the translation error obtained in sequences 11, 12, 13, 14 and 15. In this set of sequences the average of the translation error is 2.525%, almost the same was expected when we estimated the trajectories of the test sequences, where the results are reflected in Table 6.3, and the average of the translation error is 2.47%. Probably, in some of the sequences 16, 17, 18, 19, 20 or 21, the error has been considerably high, thus increasing the average translation error, as in sequence 1.

High translation error can be due to different causes, in the case of sequence 1, it is due to the presence of multiple moving objects and few static objects in an environment where the keypoints found are very similar to each other. Almost the same occurs in sequence 21, both trajectories are taken at a high speed on a road, we can observe this in Table 6.2 and Table 6.3. The velocity in sequence 21 is about 82 Km/h, and if we look at the graph of translation error as a function of speed (Figure 6.1), at that speed the error increases.

Position	**Method**	**Setting**	**Translation**	**Rotation**	**Runtime**
122	CPC	st	4.01 %	0.0100 [deg/m]	0.10 s / 1 core

Table 6.1: Algorithm proposed in KITTI Benchmarking

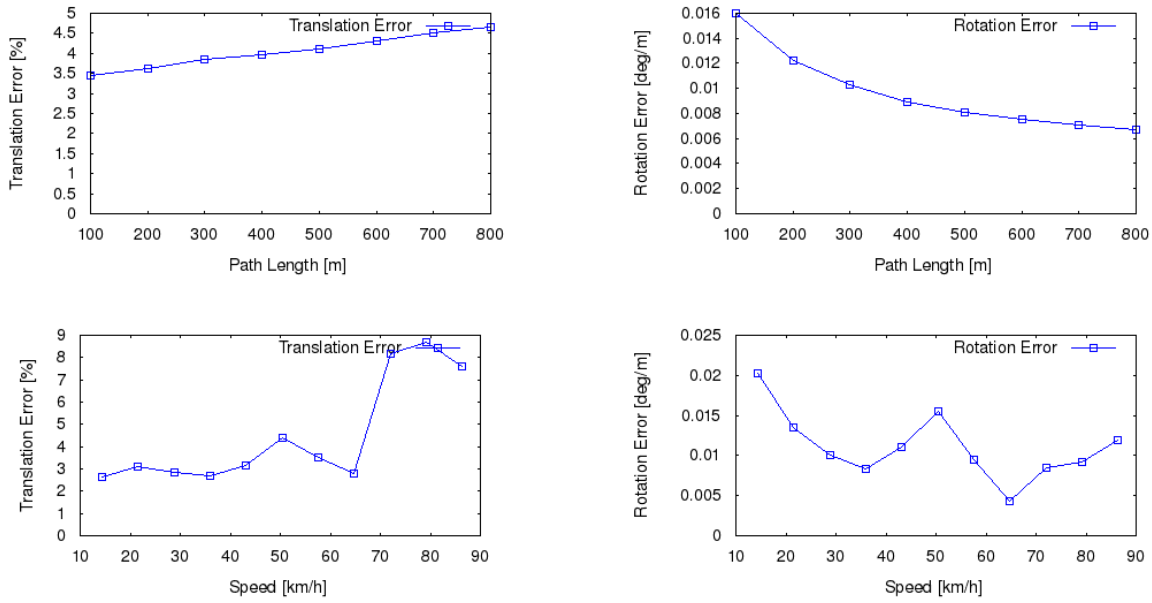


Figure 6.1: Test Set Average

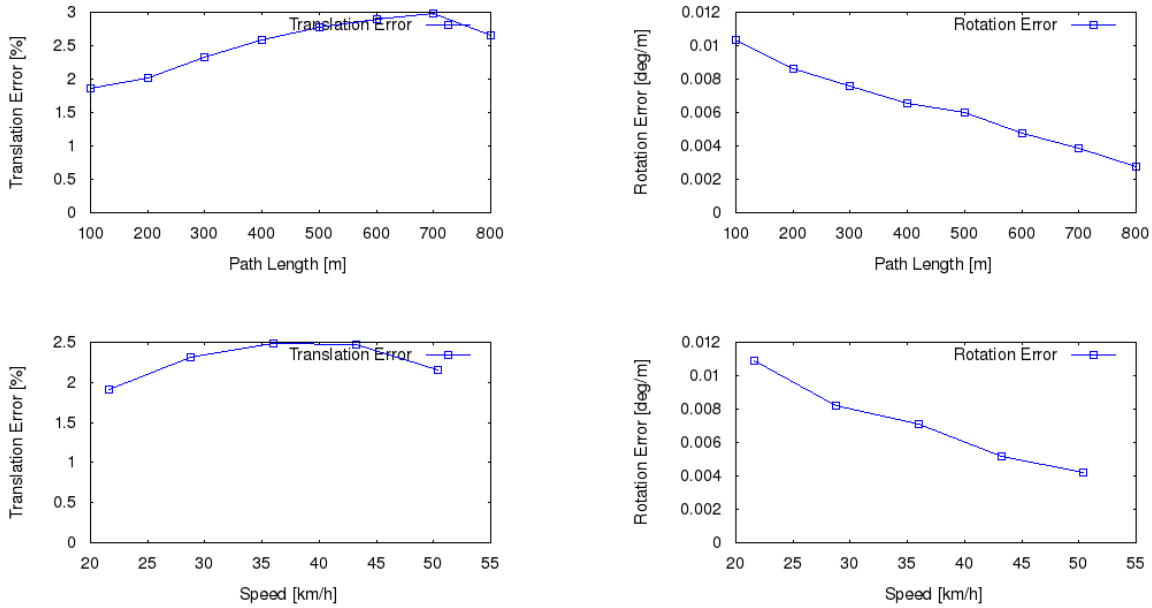


Figure 6.3: Evaluation sequence 11.

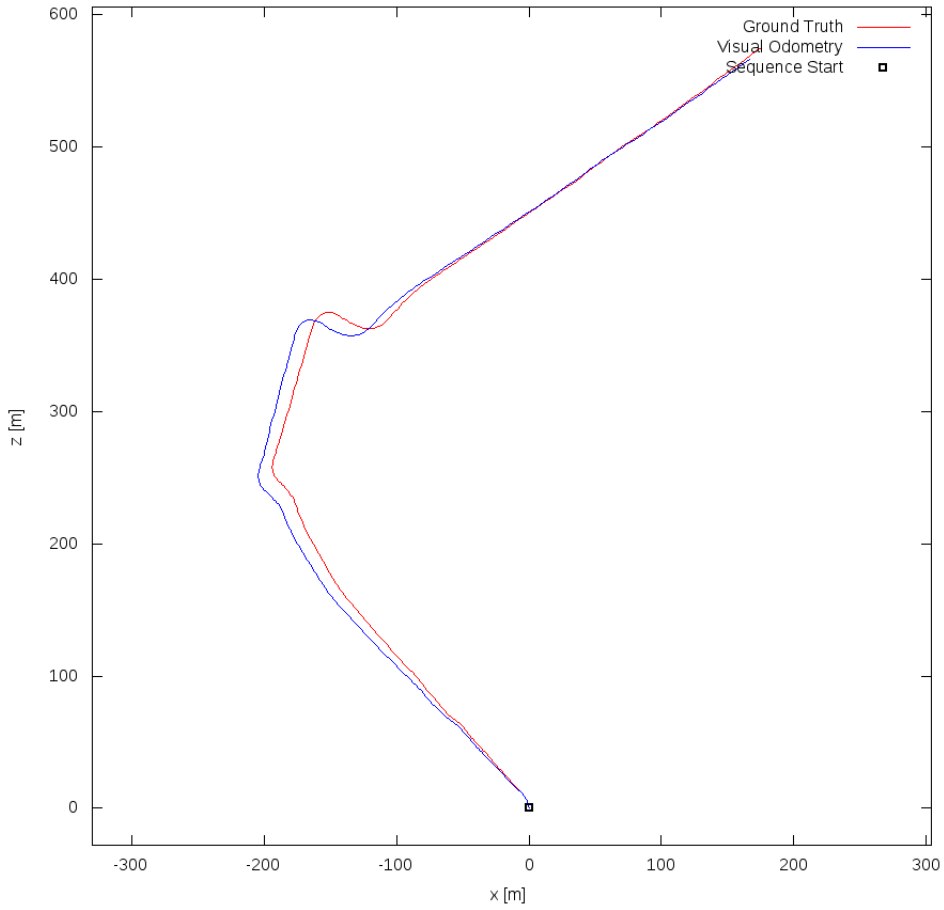


Figure 6.2: sequence 11

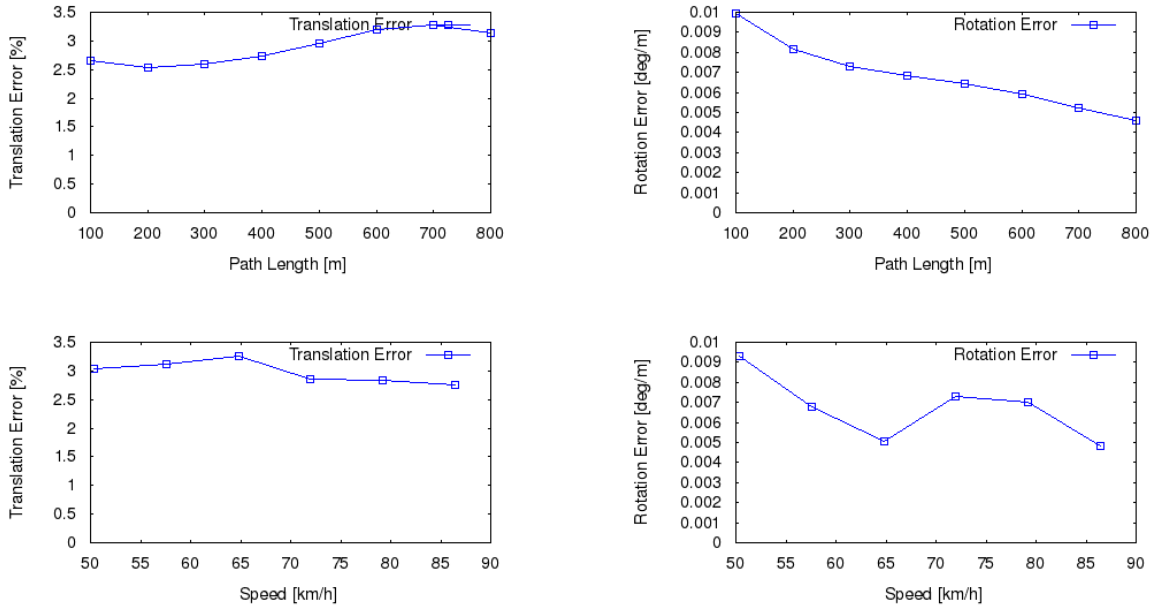


Figure 6.5: Evaluation sequence 12.

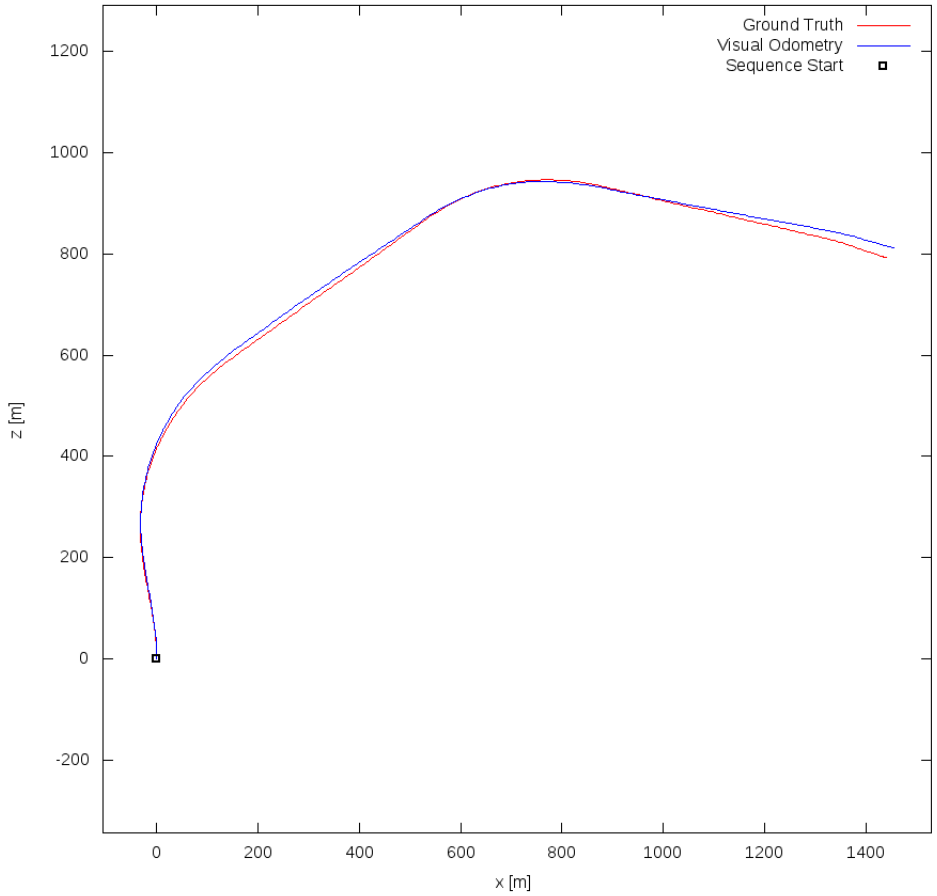


Figure 6.4: sequence 12

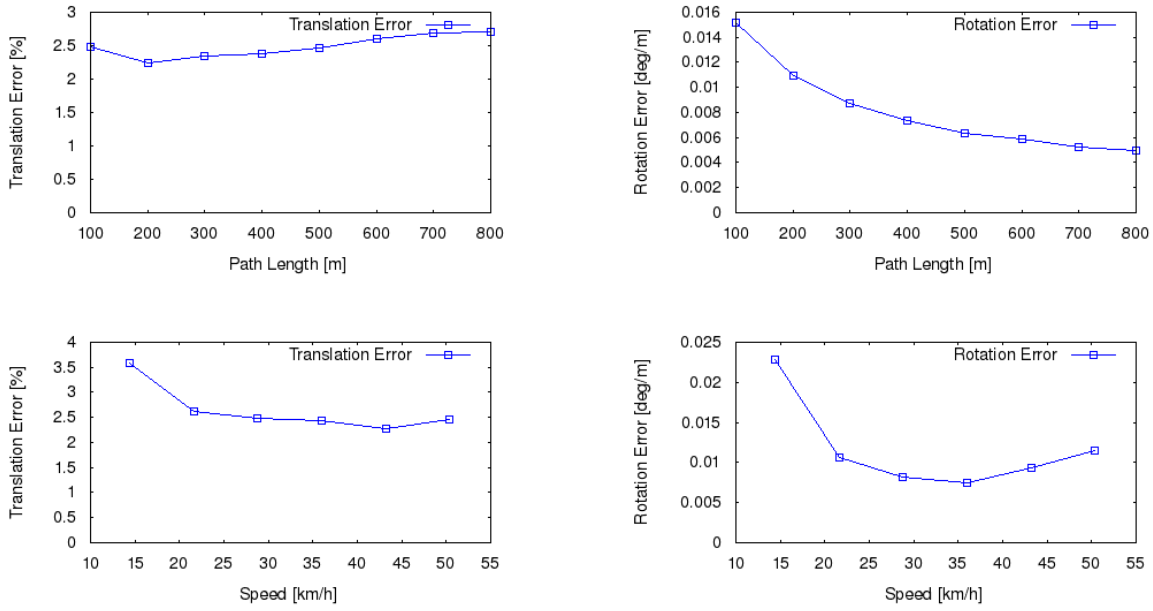


Figure 6.7: Evaluation sequence 13.

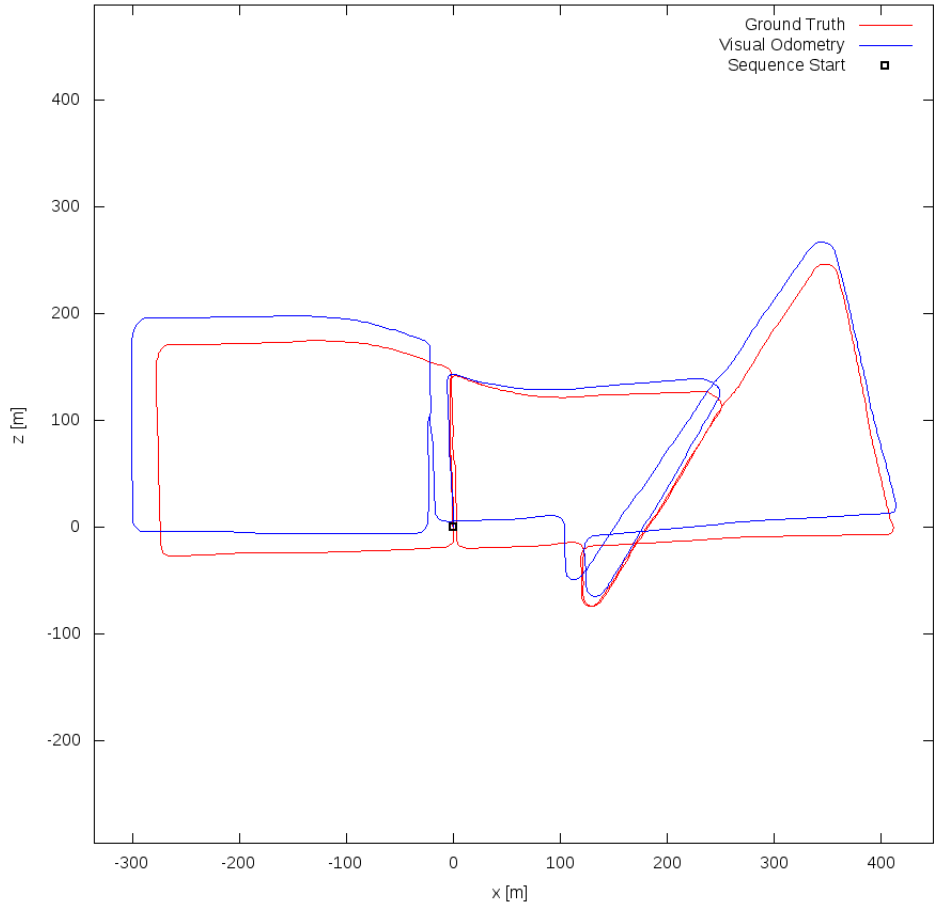


Figure 6.6: sequence 13

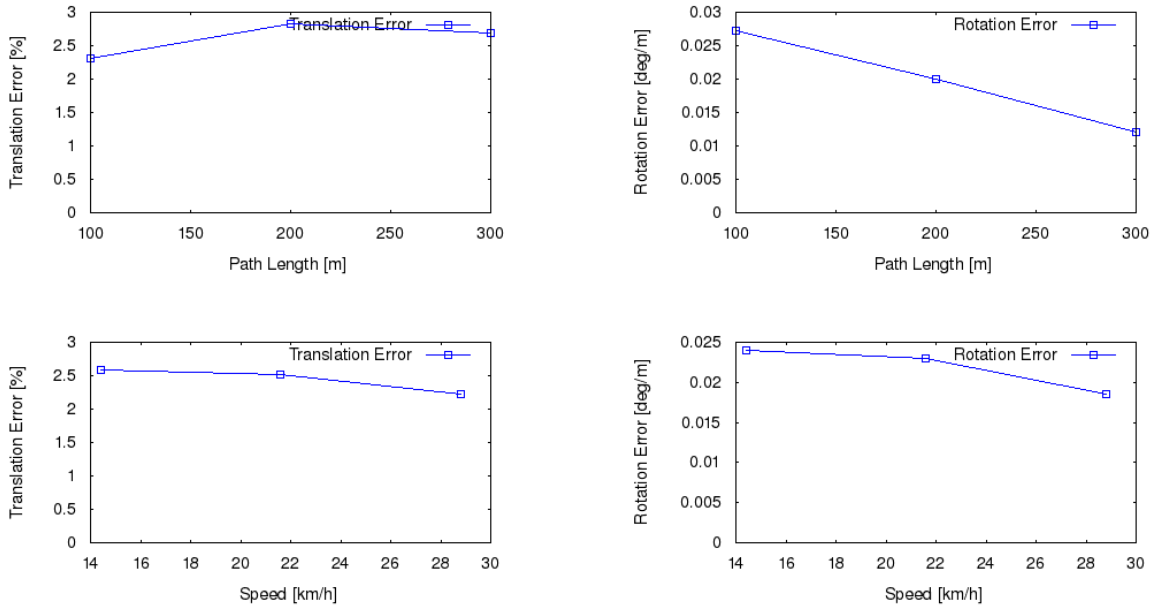


Figure 6.9: Evaluation sequence 14.

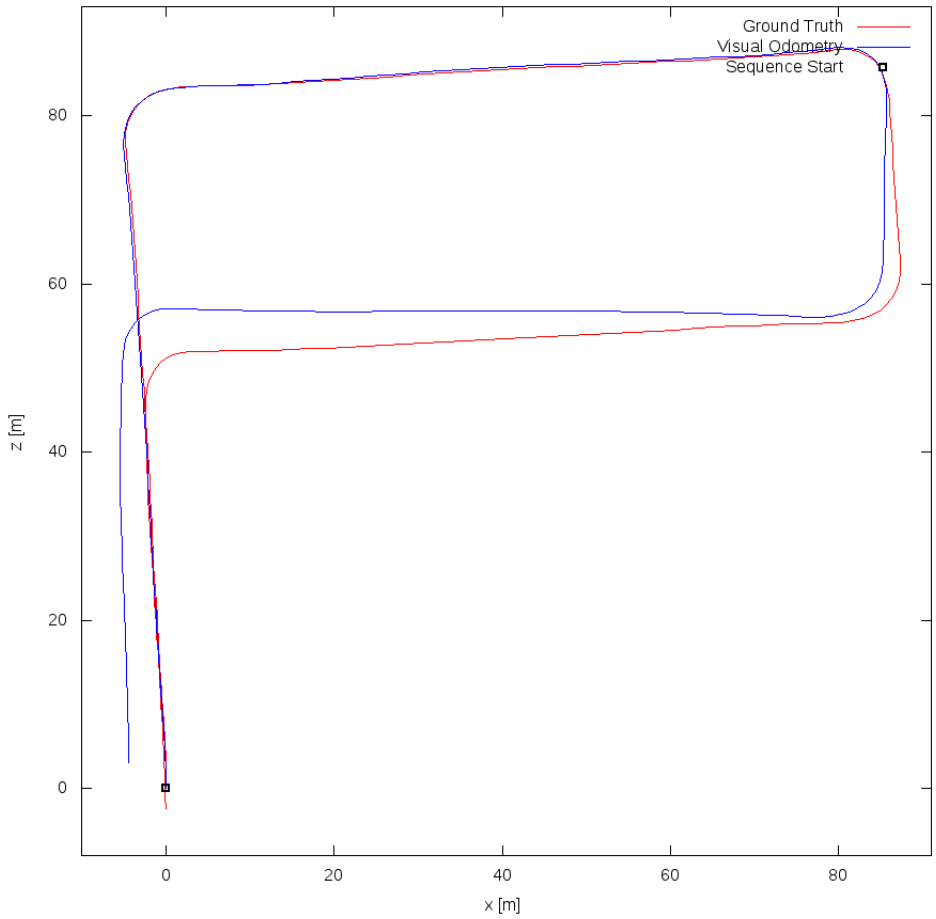


Figure 6.8: sequence 14

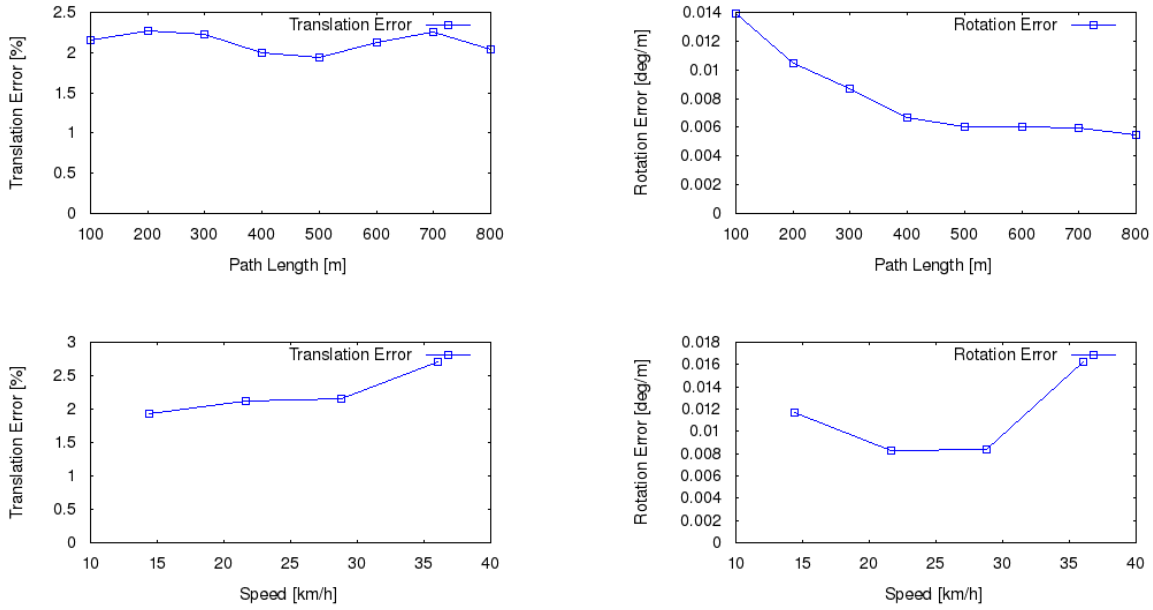


Figure 6.11: Evaluation sequence 15.

In comparison with other algorithms of the state of the art, the results obtained are not good enought in translation, but they are similar in rotation and better in computational time (comparing the number of

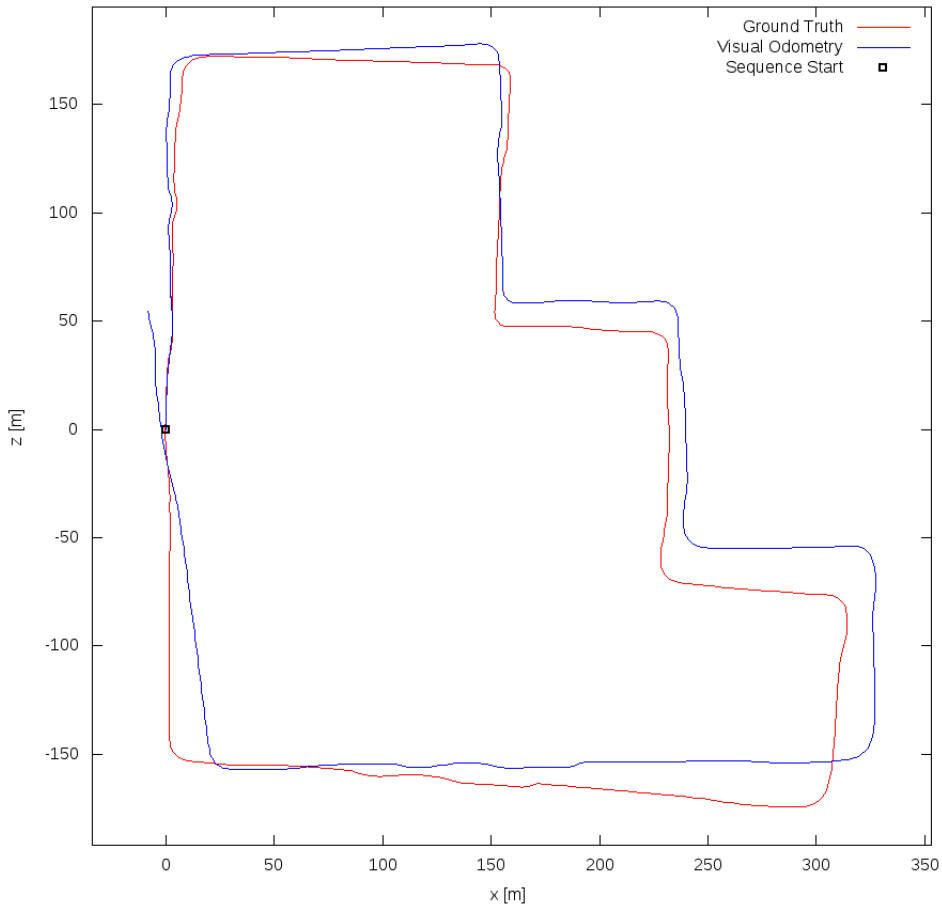


Figure 6.10: sequence 15

cores). Other algorithms uses more than one core to estimate the trajectories, this increases computational complexity and can be a disadvantage in case of limited resources, however, it allows more computations to be performed while ensuring real time. These computations, in the previous algorithms, have been invested in optimisation and error improvement processes allowing better results in translation.

6.2. Time analysis

The time taken by the algorithm to estimate the movement between two images is 108ms using one core at 3GHz. In this section we study how long it takes for each of the tasks, we can see the time taken for each task in Table 6.5

As we can see in Table 6.4 the computational time is almost the same as in the algorithms in the state of the art, the main difference is that only one core is used. Therefore, it can be said that the presented algorithm is the fastest in the state of the art.

As it is reflected in Table 6.5 each task takes, on average, a certain computational time. If more than one core is used it is possible to improve the computation time by running some of the tasks in parallel. The result is that, if two cores are used, the estimated computational time should be, approximately, 70 ms.

Sequence	Translation error (%)	Rotation Error(deg/m)	Distance(m)	Velocity (km/h)
11	2.515	0.011	861	50
12	2.886	0.012	2089	87
13	2.49	0.014	2835	46
14	2.606	0.035	340	22
15	2.128	0.014	1316	33
16	-	-	1577	42
17	-	-	866	72
18	-	-	1335	30
19	-	-	4323	42
20	-	-	679	50
21	-	-	5346	82

Table 6.2: Results in evaluation sequences

Sequence	Translation error (%)	Rotation Error(deg/m)	Distance(m)	Velocity (km/h)
0	2.424	0.0051	3724	46
1	5.566	0.0082	2453	96
2	2.66	0.005	5067	49
3	1.43	0.004	560	31
4	1.02	0.011	393	56
5	2.554	0.009	2205	40
6	2.086	0.012	1232	51
7	2.02	0.0075	694	39
8	2.436	0.01	3222	43
9	2.602	0.014	1705	52
10	2.38	0.012	919	51

Table 6.3: Results in test sequences.

As we see in the graphic Figure 6.12 the most computationally expensive processes are Feature Detection and Description and Depth Estimation. This is because, as we explain in previous sections, these processes involve a lot of mathematical calculations.

The goal of a good Visual Odometry algorithm is to be able to run in real time. To achieve this, the computational time should be as homogeneous as possible and, since the images that form the KITTI sequences are taken every 100ms, to accomplish this goal it should be less than this value.

We find that there are certain points where the computation time is slightly higher along the trajectory, for example, in places where there is a higher number of features. As an example of the distribution of the time we can observe the graphic Figure 6.13. Here we can see that the normal time required to estimate the position is between 100ms and 120ms, however, we can see that there are values that exceed these times and could cause problems in a real time application.

	SOFT-SLAM	SOFT	OV2-SLAM	ORB-SLAM2	PROPOSED ALGORITHM
trel (%)	0.65	1.03	0.94	1.15%	4.01
Rrel [deg/m]	0.014	0.0029	0.0023	0.0039	0.01
t	0.01s	0.01s	0.01s	0.06s	0.01s
cores	2c @ 2.5 Ghz	2c @ 2.5 Ghz	8c @ 3.0 Ghz	2c @ >3.5 Ghz	1c @ 3 Ghz

Table 6.4: Comparison of average results.

Process	Time
Data acquisition	13ms
Feature Detection and Description	32,8ms
Feature Matching	11,9ms
Depth Estimation	32,8ms
Translation Estimation	11,26ms
Rotation Estimation	5,65ms

Table 6.5: Time Analysis

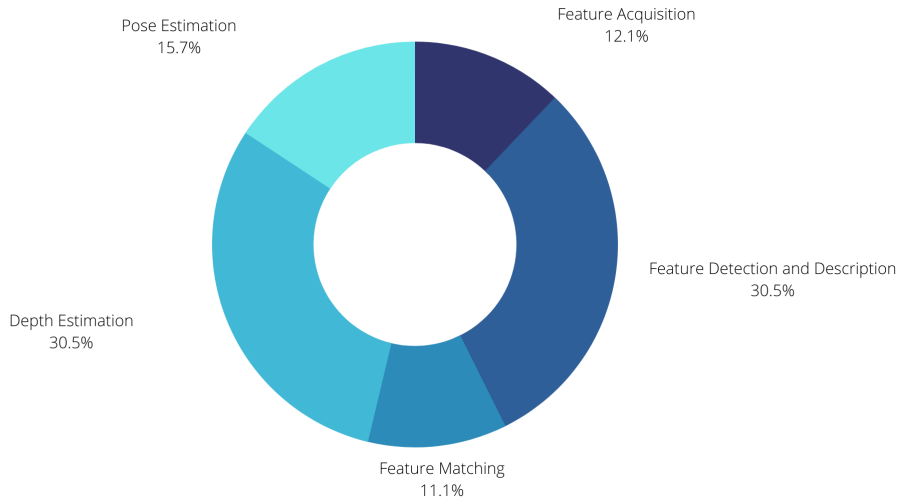


Figure 6.12: Percentage graph of time spent on processes

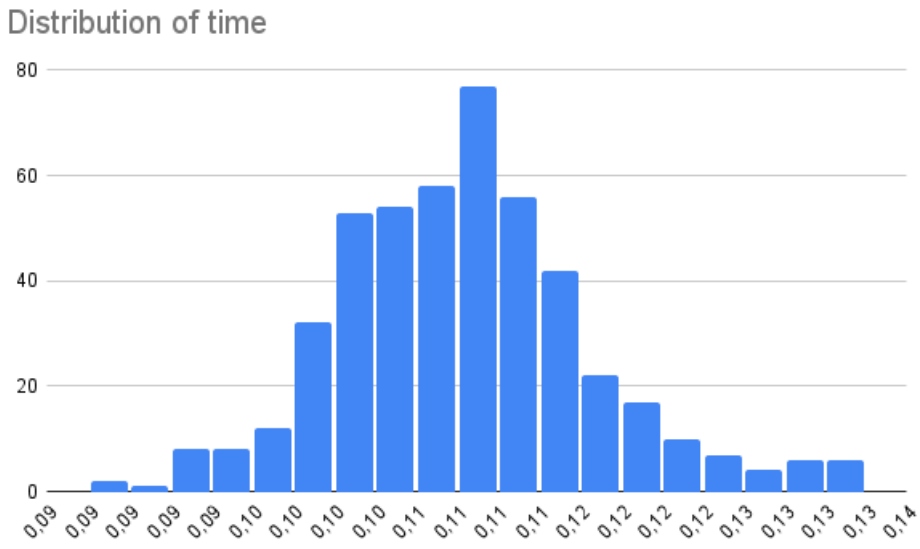


Figure 6.13: Time Distribution in Sequence 07 from KITTI Dataset.

Chapter 7

Conclusions

Throughout this project, different stereo Visual Odometry algorithms have been studied, which are currently at the top of the KITTI Benchmarking ranking. The main objective of the methods studied is to ensure real-time trajectory estimation.

One of the main problems with odometry is that the error is accumulative, meaning that the error increases as the vehicle or robot moves forward. For this reason, odometry is not useful for long distances. However, on short distances, where the GPS signal may have been lost, Visual Odometry provides good results and thanks to it, the location of the vehicle is not lost.

Throughout this project we have encountered different challenges. When developing the algorithm, the main problem has been to find a solution to improve the translation error and avoid the execution time to be too high.

To evaluate this algorithm objectively, KITTI dataset has been used, thanks to this test we have been able to compare the results with other existing algorithms, obtaining as of today (september 2021), the 122^{nd} position in KITTI Benchmarking, with an average translation error of 4.01% and a rotation error of 0.01[deg/m].

Chapter 8

Specifications

This chapter will describe the conditions necessary to be able to execute the algorithm explained before.

8.1. Hardware

A personal computer with the following characteristics has been used to develop this project:

- Intel(R) Core(TM) i7-9700F CPU @ 3.00GHz (8CPUs) 3.00 GHz
- RAM Memory: 16GB
- Hard Disk HDD 1TB
- Video Card: NVIDIA GeForce RTX 2060 Super.

8.1.1. Minimal requirements

- CPU 2.5GHz or faster.
- RAM Memory: 100MB
- At least 22GBs of free hard disk memory, this is due to the use of the KITTI grayscale odometry data set. If the color odometry data set is used, then, 65GBs of free hard disk memory will be necesary.

8.2. Software

- Linux Operating System: Ubuntu 18.04.5 LTS.
- Python 3.6.9.
- OpenCV 4.5.1.

Chapter 9

Budget

9.1. Hardware Cost

In this part of the project, an estimate of the cost of carrying out the work will be made.

Material	Unit Price	Units	Total
Desktop PC	1085€	1	1085€

Table 9.1: Hardware Costs

9.2. Software Cost

The software material used for the Final Degree Project is specified below.

Material	Price
Ubuntu 18.04.5	0€
OpenCV 4.5.1	0€
Python 3.6.9	0€

Table 9.2: Software Costs

9.3. Personnel costs

An average wage of 15 euros per hour is assumed. Working 5 hours a day.

Task	Days	Cost
Study of state of the art and software tools.	60	4500€
Algorithm development.	30	2250€
Optimisation adjustments and test.	60	4500€
Report writting	45	3375€
Total	195	14625€

Table 9.3: Software Costs

9.4. Total Costs

Material	Total
Hardware Costs	1085€
Software Cost	0€
Personnel costs	14625€
Gross Cost	15710€
IVA (21%)	3299,1€
Total	19009,1€

Bibliography

[1] M. O. A. Aqel, M. H. Marhaban, M. I. Saripan, and N. B. Ismail, "Review of visual odometry: types, approaches, challenges, and applications," *SpringerPlus*, vol. 5, no. 1, p. 1897, Oct 2016. [Online]. Available: https://doi.org/10.1186/s40064-016-3573-7

[2] H. Badino, A. Yamamoto, and T. Kanade, "Visual odometry by multi-frame feature integration," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshops*, June 2013.

[3] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," vol. 18, no. 4, 2011, pp. 80–92. [Online]. Available: https://doi.org/10.1109/MRA.2011.943233

[4] C. Harris and M. Stephens, "A combined corner and edge detector," in *In Proc. of Fourth Alvey Vision Conference*, 1988, pp. 147–151.

[5] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," vol. 60, no. 2, Nov 2004, pp. 91–110. [Online]. Available: https://doi.org/10.1023/B:VISI.0000029664.99615.94

[6] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, Jun 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1077314207001555

[7] D. Valiente García, L. Fernández Rojo, A. Gil Aparicio, L. Payá Castelló, and O. Reinoso García, "Visual odometry through appearance- and feature-based method with omnidirectional images," *Journal of Robotics*, vol. 2012, p. 797063, Sep 2012. [Online]. Available: https://doi.org/10.1155/2012/797063

[8] Q. Liu, H. Zhang, Y. Xu, and L. Wang, "Unsupervised deep learning-based rgb-d visual odometry," 2020. [Online]. Available: https://doi.org/10.3390/app10165426

[9] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, "Unsupervised learning of depth and ego-motion from video," 2017.

[10] G. Cai, H. Lin, and S. Kao, "Mobile robot localization using gps, imu and visual odometry," in *2019 International Automatic Control Conference (CACS)*, ser. 2019 International Automatic Control Conference (CACS), 2019, pp. 1–6. [Online]. Available: https://doi.org/10.1109/CACS47674.2019.9024731

[11] C. G. Harris and M. Stephens, "A combined corner and edge detector," in *Alvey Vision Conference*, 1988.

[12] A. Huamán, "Harris corner detector." [Online]. Available: https://doi.org/10.1016/j.cviu.2007.09.014

[13] J. Shi and Tomasi, "Good features to track," in *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 1994, pp. 593–600.

[14] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008, similarity Matching in Computer Vision and Multimedia. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1077314207001555

[15] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Computer Vision – ECCV 2006*, A. Leonardis, H. Bischof, and A. Pinz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 430–443.

[16] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *Computer Vision – ECCV 2010*, K. Daniilidis, P. Maragos, and N. Paragios, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 778–792.

[17] M. Muja and D. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *VISAPP*, 2009.

[18] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3354–3361.

[19] I. Cvisic, "Soft-slam : Computationally efficient stereo visual slam for autonomous uavs," 2017.

[20] I. Cvisic and I. PetroviÄ, "Stereo odometry based on careful feature selection and tracking," in *2015 European Conference on Mobile Robots (ECMR)*, 2015, pp. 1–6.

[21] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras," *CoRR*, vol. abs/1610.06475, 2016. [Online]. Available: http://arxiv.org/abs/1610.06475

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá