POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering

Master's Degree Thesis

Robot pose calculation based on Visual Odometry using Optical flow and Depth map

Supervisors

Prof. Marcello CHIABERGE

Chiara BORETTI

Simone ANGARANO

Candidate

Sara Lucia CONTRERAS OJEDA

December 2022

Abstract

Visual Odometry (VO) is a technique that allows knowing accurately the position of a robot over time, useful, for instance, for motion tracking, obstacle detection, avoidance, and autonomous navigation. To do these tasks requires the use of images captured by a monocular or stereo camera on a robot. From these images, it is needed to extract features to figure out how the camera is moving. This can be done in three different ways: feature matching, feature tracking, and calculating the Optical Flow. Once the key feature points are found is possible to do a 3D to 3D, 3D to 2D, or 2D to 2D motion estimation.
Over the years many implementations of visual odometry have been done, a common denominator is that they need to be specifically fine-tuned to work in different environments and there is needed for prior knowledge of the space to recover all the trajectory done by the camera. In order to create a more generalized implementation, able to adapt to distinct environments, and improve the accuracy of the pose estimation, deep learning techniques have recently been implemented to overcome the limitations previously mentioned. Convolutional Neural Networks (CNNs) have proven to give good results for artificial vision tasks, however, VO is not a task that has been solved with this technique. On the other hand, CNNs have been able to solve with good results tasks such as feature detection and Optical Flow, these are included in some approaches to VO estimation, obtaining an improvement in the results. Considering this, for the purpose of this work CNNs were used for the estimation of the Optical Flow.
This work presents an approach to solving the Visual Odometry problem using Deep-Learning in one of the stages as a tool to calculate the trajectory of a stereo camera in an indoor environment. To achieve this goal there were implemented Convolutional Neural Networks such as RAFT and The Flownet to calculate the optical flow from two consecutive frames, also was calculated the depth map from the right and left camera images of each frame using an OAK-D camera. The aim of this procedure was to extract key feature points from the images over time. The key points of the left image in the first frame were found with a key points feature extractor that in this case was the Fast Algorithm for Corner Detection. Once gotten, the optical flow was used to find the same feature points of the previous left image in the left image of the consecutive frame. Then, from the depth map was obtained the disparity and with this value were located the same key feature points in the right images of the two frames. The key feature points were used to do triangulation and find the 3D points, with them was possible to obtain the transformation matrix that has the information on the pose of the camera along the period of time of the measure.

The proposed method has been implemented with a prototype robot that is in development at the Service Robotic Center of the Politecnico di Torino (PIC4SER), which will have the task of measuring the levels of CO2 in indoor environments with the aim to create an autonomous system capable of purifying the air of these spaces when is needed. The camera was put on the robot and with this system indoors courses were done. The movement of the robot was controlled by a person with a joystick and the odometry was captured using ROS2. The success of the Visual Odometry estimated from the proposed methodology in this work was compared with the odometry of the robot, obtained with ROS2, and plotted using MATLAB 2021. The two plots were compared to qualify the estimation of the implementation suggested in the different spaces.

Acknowledgements

First I want to thank my family for the emotional and economic support, especially my mom who always supports me and encourages me to achieve my dreams, without her nothing could be possible, I will always be grateful to you mom for giving me the best of you. I want to thank also to my grandmother "nona" Luisa that has been a second mom to me and a dad as she said, thank you for always showing me your love and support and for teaching me to never give up. I want to thank my second parents, my uncle Anyelo and my Aunt Duperly for always believing in me, loving me as your daughter, and allowing me to live a lot of wonderful experiences with you. I want to thank my Nana Maria and Aunt Amparo for loving and taking care of me since I was little and for teaching me to appreciate myself. Also, I want to thank my dad for his support in this journey. I miss you an love you all so much.

I would like to thank my beloved friends that are my second family here that I am far away from my home. I want to thank to my dear sensei, alias Mushu, Yitzak, for always making me laugh no matter how stressed I am, you always take off a smile from me, and also thank you for being loyal and following me, no matter if that make us walk for one hour. I want to thank to Cesar, for his tons of patience and for always being by my side supporting me, and doing his best to take care of me, and also of course thank you for literally not letting me die of hungry. I want to thank to Valentina to be like a sister to me, for pushing me to be better and more productive, and for always believe in me. I want to thank Sergio and Mayra to adopt me and making me feel welcome in a new country, thanks for all the matchas, talks, and saying yes to almost all the ideas to hang out that I have. And thank you to all the friends that I am not mentioning here by name but that have been present in this journey.

Finally, last but not least, I want to thank the members of the PIC4SER, especially professor Marcello Chiabergue for letting me participate in this project and opening the door from this lab. I want to thank Chiara for helping me, always being available, and for being the reference to develop this job.

Table of Contents

List of Tables

List of Figures

Acronyms

AI
Artificial Intelligence

ML
Machine Learning

DL
Deep Learning

VO
Visual Odometry

OF
Optical Flow

CNN
Convolutional Neural Networks

RAFT
Recurrent All-Pairs Field Transforms

FAST
Features from Accelerated Segment Test

Chapter 1

Introduction

Finding the position of a robot is one of the main challenges in mobile robotics because in order to achieve autonomous navigation is essential that the robot maintains knowledge of its position over time. For this purpose, several techniques have been proposed over the years, each one using different types of sensors depending on the approach taken. The most implemented type of odometry is wheel odometry which uses sensors to obtain the number and speed of the wheel rotations to calculate the translation of the robot. Despite being the most common and simplest, it suffers from position drift due to wheel slippage which could cause imprecision in the measures taken. In order to overcome these issues Visual Odometry (VO) showed up and started to be more used.

VO estimates the pose of a robot using images acquired from single or multiple cameras attached to the robot and had become one of the robust techniques for vehicle localization. Visual Odometry is considered more accurate even than GPS, INS, wheel odometry, and sonar localization. The aim of this technique is to estimate the camera pose and one of its several advantages is that has a good balance between cost, reliability, and implementation complexity, also has the advantage that since it uses a camera that in comparison with the other sensors use to do odometry, is inexpensive. VO can be considered a low-cost method to do the localization of a robot [1], [2].

Over the years several works have been developed in the area. For instance, Visual Odometry approaches based on classic geometry work with 2D-3D matches between 3D scene coordinates for pose estimation and the position of the pixels in 2D. These type of approaches needs a more complex pipeline because includes numerous steps that have to be implemented with attention, such as the calibration of the camera, feature detection, feature matching and tracking, outlier rejection, and motion estimation. Geometry-based Visual Odometry has been strongly

implemented in the last few years, something to take into account is that its reliability and accuracy depend on how well was done the setup. Due to the features mentioned before, the traditional correspondence search pipeline typically finds sparse feature points first before matching extracted features, producing a small number of high-quality correspondences. When solving Visual Odometry issues, the precision and variety of the correspondences are crucial. Zhan, H et.al suggested in their work to exploit the consistency constraint between bi-directional flows to extract precise correspondences in a variety of ways from the dense predictions of an optical flow network., and after the selection of the correspondence, these are taken to fed an Epipolar Geometry based tracker and PnP based for accurate and robust VO estimations [3]. Also, deep learning techniques have been implemented in Visual Odometry, an example is the work of Xicheng, B et.al that proposed a method called DL_Hybrid VO which uses deep learning for image processing and geometric localization methods use in pose estimation. The system is able to estimate the rotation and translation of each frame, and with that recover the camera trajectory with a good accuracy frame-by-frame [4]. Lim S, et.al in their work "Visual Odometry using Convolutional Neural Networks" used three types of CNN, each of them with different objectives. The first one was designed to predict the transnational motion in two consecutive frames from the camera, in this case the model is predicting the motion in a global coordinate frame. The second model predicted the camera's transnational motion between two consecutive frames and it used an inertial measurement system. Finally the third, predicted the change in orientation of a camera between two consecutive images, this last was the one that actually can apply for visual odometry approach, because it can bring the position if the agent only have cameras as a sensors [5].

1.1 Starting point and objective of the thesis

The starting point of this project was the need to localize a robot in an indoor environment to help in autonomous driving implementations. A topic that has been studied widely before is the estimation of the optical flow with Convolutional Neural Networks, but there is not enough information reported about works developed in indoor environments, not in the task of estimating the Optical Flow and either in the task of Visual Odometry. For this reason, the motivation for this work was to develop a methodology to estimate the pose of a robot with images captured by a stereo camera and using deep learning in a stage of the work, that for this purpose was decided to use in the Optical Flow estimation. Considering all this information the objectives of this work are:

- Obtain the pose of a camera on a robot, using stereo images in an indoor environment with the presence of different objects along the course.

- Estimate the optical flow using Convolutional Neural Networks that help to obtain information on where and how are moving the feature points identified in the image.
- Estimate the depth map to calculate the position of the key feature points in the right images that were previously found in the left images, to calculate the 3D feature points of each frame.
- Calculate the transformation matrix with less level of error possible in order to obtain a good approximation of the camera/robot pose along the time.

1.2 Organization of the thesis

This thesis work is organized following the workflow of this project along the time in which was developed. It contains seven chapters from which the Chapter 1 presents the motivation to do this thesis work and also its starting point. The Chapter 2 contains the state of the art, in which can be found the concepts needed to develop this work and also other research works taken as a reference. The Chapter 3 contains the explanation about the datasets used in this work and how each of them were handle. The chapter 4 is about the architecture of the neural networks used in this works to calculate the optical floe and also the results obtained. The Chapter 5 explain the process of the Key point feature extraction of the left and right images and how works the algorithm built. The Chapter 6 contains the methodology follow to calculate the pose of the camera and the strategies used. Finally the Chapter 7 includes the conclusions and Future work.

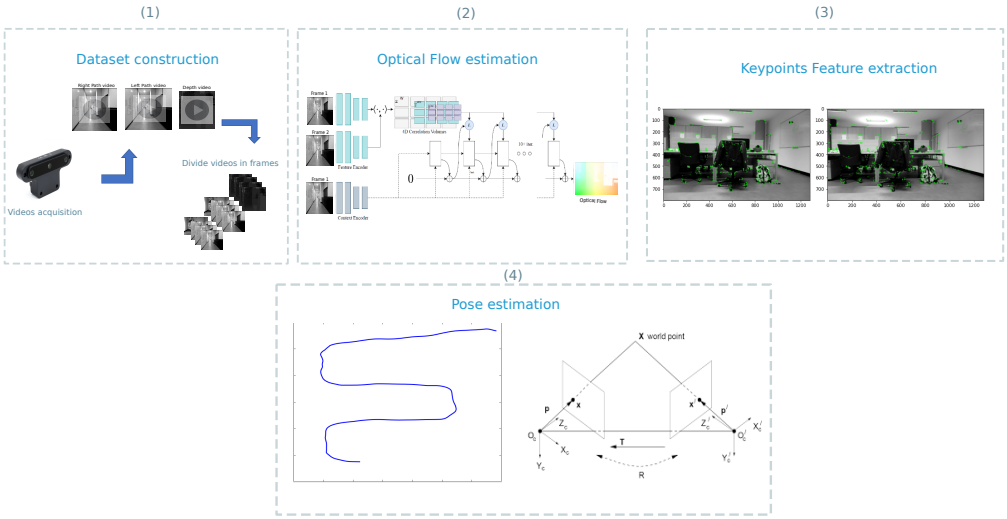

Figure 1.1: Methodology of the work

Chapter 2

State of the art

2.1 Visual Odometry

Visual Odometry estimates the ego-motion using only the input of single or multiple cameras attached to it. This methodology focuses on local consistency and estimates the path of camera/robot pose after pose, possibly performing local optimization. Can be divided into two methods: The monocular and Stereo camera methods [6]. These are divided in:

- Feature matching: matching features over a number of frames.
- Feature tracking: matching features in adjacent frames.
- Optical flow: based on the intensity of all pixels or specific regions in sequential images.

VO is the process of estimating the camera's relative motion by analyzing a sequence of camera images. To estimate the motion there are three techniques:

- 3D to 3D motion estimation: motion is obtained by triangulating 3D feature points observed in a sequence of images. The transformation between the camera frames is then estimated by minimizing the 3D Euclidean distance between the corresponding 3D points. The minimum number of points depends on the system DOF. More points, more computational cost, and better accuracy.
- 3D to 2D motion estimation: The 2D re-projection error is minimized to find the required transformation. The minimum number of points required is based on the number of constraints of the system.

- 2D to 2D motion estimation: used when there are not 3D data available. Use epipolar geometry. For example to estimate the relative transformation between the first two calibrated monocular frames where points have not been triangulated yet.

2.1.1 Visual Odometry Approaches

There are essentially three ways to approach estimating the position of a mobile robot using Visual Odometry: through a feature-based approach, an appearance-based approach, or a hybrid of feature- and appearance-based approaches.

Feature-based approach

The feature-based approach involves finding distinctive features among the extracted ones, matching or tracking them, and then predicting the motion. Distinctive features include corners, lines, and curves. According to this method, finding candidate-matching features involves comparing each feature in the two images and calculating the Euclidean distance of the feature vectors. When stereo Visual Odometry is used, the points in the second frame that corresponds to the extracted features in the first frame are matched, giving the 3D position of the points in space. The geometric transformation between two images captured by the camera using a collection of related feature points can be found in order to estimate the relative pose of the camera. The camera motion is estimated based on feature displacement. It is necessary to identify nearest neighbor pairings among the feature descriptors of two images in order to compute the matching between the feature points of the two images.
One of the crucial Bayesian filters used to enhance the precision and polish VO estimation results is the Kalman filter [7]. It predicts current feature positions using a previous vehicle state estimate, compares these predictions to recent observations, and then determines an updated vehicle state. Inertial measurement unit (IMU) and volumetric data are combined by the Kalman filter. The kinematic estimate and this combined estimate are then evaluated to ascertain whether and how much slippage has taken place. If slippage has taken place, a slip vector is created by comparing the current Kalman filter estimate to the kinematic estimate. This slip vector is then utilized to determine the required wheel speeds and steering angles to account for slip and maintain the target course.

Appearance-based approach

Using optical flow, the camera motion and vehicle speed may be calculated. The Optical Flow method calculates the displacement of brightness patterns from one image frame to the next using the intensity values of the surrounding pixels.

Dense OF techniques, such as the Horn-Schunck algorithm, which calculates the displacement at each pixel by employing global constraints, estimate the displacement for every picture pixel. Sparse optical flow techniques, like the Lucas-Kanade approach, are those that determine the displacement for a subset of the image's pixels. In contrast to sparse OF algorithms, dense algorithms avoid feature extraction yet are less noise-resistant. Therefore, for many VO applications, sparse OF algorithms are preferred to dense OF algorithms. When designing features for sparse algorithms, it is important to keep in mind that pixels in areas with a greater neighbor variance will result in more accurate displacement estimates.
The template matching method is one that is frequently employed in appearance-based approaches. The template matching method chooses a patch or template from the most recent frame of the image and tries to match it in the following frame. By matching a template between two successive image frames, the displacement and rotation angle of the vehicle are determined. The primary function of many computer vision programs is template matching. The method of locating a sub-picture or an object within a bigger scene image is known as template matching. The larger image is known as the search area, while the smaller image is known as the template. If the template is found in the search area, template matching decides where it is and establishes whether it is present. By shifting the template over the search area and determining the degree of similarity in each position using different similarity metrics, it determines how similar the template and search region are. The likely position of the template found in the search region is the shift position with the highest degree of similarity. [8]

Hybrid of feature and appearance-based approaches

Hybrid of feature and appearance-based approaches for textured contexts, such as urban and harsh landscapes, the feature-based method is appropriate. However, this method falls short when dealing with environments that have only one pattern and have little to no texture (e.g., sandy soil, asphalt, and concrete). The feature-based technique is ineffective in low-textured settings since there aren't many prominent features that can be found and monitored there. In low-textured environments, however, the appearance-based approach is more reliable and effective than feature tracking techniques. This method has a high likelihood of successfully matching two consecutive image frames since a big template can be used in the matching process. In some situations, a hybrid approach—which combines feature- and appearance-based approaches—is the ideal choice. They combine the tracking of important features across frames with the use of pixel intensity data from the entire or group of images. For instance, the hybrid technique was used in Scaramuzza and Siegwart (2008a) because the appearance-based strategy by itself was not highly resilient against picture occlusions. Therefore, in their work, the translation of the

vehicle was estimated using image features from the ground plane, whereas the rotation of the vehicle was estimated using the picture appearance.[8]

2.1.2 Sensors and Strategies for localization

In order to execute Visual Odometry approaches, robust and precise localization schemes incorporate information from IMUs, wheel encoders, GPS, laser, radar, ultrasonic, and vision software algorithms. The fusion might be restricted to a small number of sensors or all sensors, depending on the application and specification of navigation and object avoidance. The sensors commonly used to localize a robot in mobile robotics or in works related to autonomous driving are:

Inertial Measurement Unit

The Inertial Measurement Units (IMUs) are usually made by a number of accelerometers and gyroscopes, also is possible to find IMUs sensors with magnetometers and barometers. In a 3D space, this sensor can take the instantaneous pose which means position and orientation, velocity that can be linear or angular, and linear or angular acceleration of the robot.

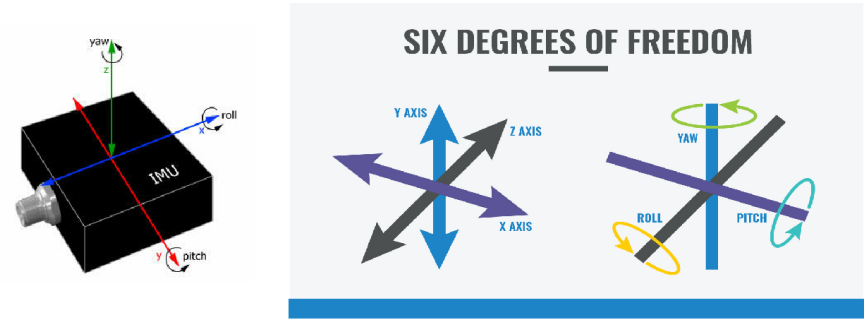

Figure 2.1: IMU

Laser sensors

Numerous positioning-related applications can make use of laser sensors. It is a technique for measuring distance through remote sensing that involves firing a laser at the target and examining the reflected light. Either phase-shift or TOF methods are required for laser-based range measurements. In a TOF system, which is similar to a sonar sensor, a brief laser pulse is emitted, and the amount of time it takes to return is measured. The terms "laser radar" or "light detection and ranging sensor" are frequently used to describe this sort of sensor (LIDAR). In contrast, a

continuous signal is sent using phase-shift systems. A reference signal produced by the same source is used to compare the phase of the signal that was returned [9]. The Laser sensor can be used for 2D and 3D analysis. The majority of these types of sensors are based on the time of flight concept. To output points with range and angle increments, signal processing is used. The price, performance, resilience, range, and weight of laser sensors might vary greatly depending on the type of robot—indoor or outdoor—and the robot's movement speed. The majority are based on the time of flight concept. To output points with range and angle increments, signal processing is used.

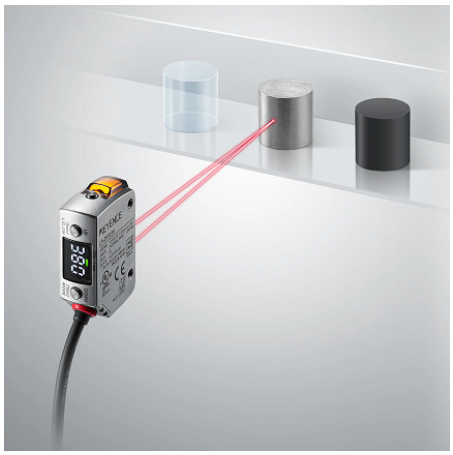

Figure 2.2: Laser Sensor

Encoders

The robot's sensors precisely tally the number of wheel rotations to calculate the distance traveled. When calculating distance using wheel encoders, the phrases odometry or dead-reckoning are employed. They must be paired with other sensors since they exhibit long-term drifts.

Vision Sensors

In visual odometry are very important the 2D, 3D, and depth cameras to obtain information. The use of computer vision and deep learning on sensor data can help with obstacle tracking, object recognition, and object detection. For autonomous robots operating in both indoor and outdoor areas where illumination conditions are adequate and can be maintained, visual odometry is becoming more important. Pose, or the location and orientation of an object in three dimensions, is provided via 3D cameras, depth cameras, and stereo vision cameras. Pose in combination

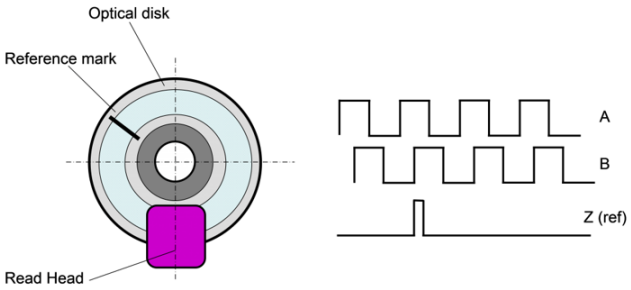

Figure 2.3: Encoder

with well-established machine vision techniques can help in industrial settings to address a variety of issues, including grasping, placement, and visual serving. When working in poor illumination situations, such as the dark or fog, thermal and infrared cameras are used.

Optical Cameras

In mobile robotic applications, cameras can be used for localization and a variety of activities. Since visual-based localization systems are more durable and dependable than other sensor-based localization systems, several academics have recently shown interest in these systems. Images captured by cameras can be used for interior and outdoor vehicle navigation, for example, to identify lane borders, lane transitions, and crossroads. A camera's pictures can record a lot of data that can be utilized for a variety of things, including geolocation. Optical cameras are inexpensive sensors that offer a lot more useful information than proximity sensors [10].

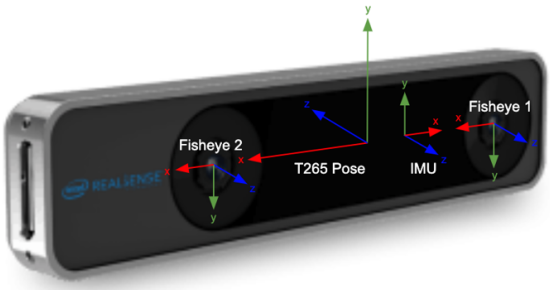

Figure 2.4: Optical camera

Pulsed and millimeter wave radars

The main objective of these sensors is to find distant objects and offer characteristics for their velocity, angle, and bearing, which are commonly calculated in relation to the centroid of the object. While most other sensors malfunction in challenging situations like rain, fog, and changing lighting, they operate in all weather conditions. However, they don't have the same level of resolution as lidar or laser.

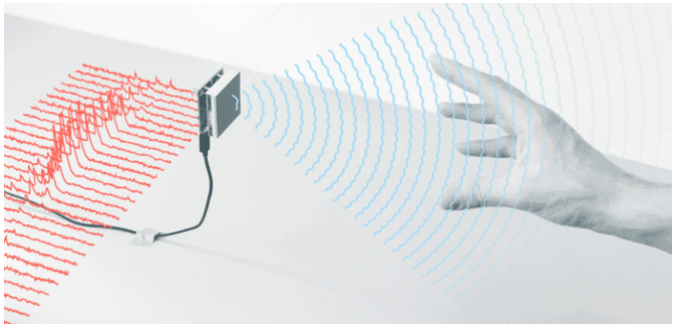

Figure 2.5: Optical camera

For localizing a robot had been developed different strategies along the years, the most common are:

Wheel Odometry

Wheel odometry is the quickest and most used technique for determining the location of moving robots. By counting the rotations that the wheels make in contact with the ground, the position of a wheeled vehicle can be estimated. Wheel rotations can be precisely converted into a linear displacement with respect to the ground [11].
An approach to relative location is wheel odometry. Wheel slippage causes position drift and inaccuracy, which causes error accumulation over time [12]. Wheel odometry translation and orientation errors rise in direct proportion to total distance traveled. Wheel odometry is straightforward and affordable, permits high sampling rates, and has good short-term accuracy [13].

Inertial Navigation System

The location and orientation of an item with respect to a known beginning point, orientation, and velocity are provided by the Inertial Navigation System (INS), a relative positioning approach. It is a navigation aid that, uses a computer,

motion sensors such as accelerometers, and rotation sensors as rate gyroscopes to continuously calculate the position, orientation, and velocity of a moving vehicle. This moving vehicle could be a surface ship, a submarine, an airplane, a spaceship, or any of the following: a rocket, a spaceship, or a ground vehicle. The benefit of INS is that it is self-contained, meaning that it does not need references from outside sources [14]
Due to the fact that INS implements repeated mathematical acceleration integrations with regard to time, it is extremely susceptible to drift accumulation. While rate-gyro data only need to be integrated once to track the orientation, accelerometer data must be combined twice to produce the location. As a result, any little inaccuracies in the measurement of acceleration and angular velocity are combined to create bigger velocity errors, which are then added together to create even larger position errors [14]. The mistakes add up over time and get worse. As a result, the position needs to be updated on a regular basis using data from another navigation system.

GPS

The satellite-based navigation system called GPS enables users to precisely pinpoint their location anywhere on or just above the Earth's surface [15]. The application of GPS extends beyond straightforward outdoor navigational activities to include geology, agriculture, landscaping, construction, and public transit. Anyone with a GPS receiver can access accurate position, navigation, and timing data for free. A notional constellation of 24 operational satellites that orbit the Earth and broadcast Radio Frequency (RF) signals packed with data makes up the GPS system. To maintain constant global coverage, they are set up so that four satellites are deployed in each of the six orbital planes [16].

GPS offers the absolute position with a known error ratio. Its long-term stability and resistance to error accumulation over time are its key benefits. A groundbreaking tool for outdoor navigation, GPS is ineffective in enclosed, subterranean, or underwater locations and works best where there is a clear view of the sky. GPS has a number of drawbacks, including intermittently high noise content, multi-path effects, poor bandwidth, interference, and jamming. Urban canyons, tunnels, and other GPS-denied environments and tight spaces are common locations for GPS outages [17].

2.1.3 Advantages and Challenges

Challenges

Robot localization in indoor contexts has been effectively done, however, the issue of robot localization in outdoor settings is still difficult to solve. Localization is challenging in outdoor contexts due to a number of elements, such as the fact that terrains are typically not flat, direct sunshine, shadows, and dynamic changes in the environment brought on by wind and sunlight [18]. The primary issues with VO systems relate to the cost of computing, as well as lighting and image circumstances [19]. In order for VO to be effective, the environment must have enough illumination and a static scene with enough texture to allow apparent motion to be extracted [20].
Another challenge for visual odometry is that in the case of monocular vision the system can suffer of scale uncertainty which means that the image scale can variate if the surface is uneven, so the image scaling factor can have difficulties to be estimated. Based on what the works done in this thematic said, such as Kitt et. al, when a significant change in the road's slope takes place, the scaling factor may be estimated incorrectly, which could result in an incorrect calculation of the trajectory that results.
The biggest difficulties with VO systems are relate to computing costs as well as lighting and imaging conditions, such as directional sunshine, shadows, picture blur, and image scale or rotation variance. The majority of the VO systems suggested in the literature currently in circulation either fail or are ineffective in outdoor settings with shadows and directing sunlight. The calculation of pixel displacement between camera frames is adversely affected by shadows and directed sunlight, which might result in mistakes in estimating the position of the vehicle. [8]

Advantages

An important advantage to consider is that in comparison to more traditional methods like GPS, and wheel odometry, vision odometry is a less expensive alternative technology that is comparatively more accurate. Cost, dependability, and implementation complexity are all appropriate trade-offs for VO. Another feature to consider is that Visual odometry can be implemented with monocular or stereo vision, which means that it is possible to use one or more cameras to capture the images or the trajectory done by the robot, and this can be chosen based on the application that is going to be developed. Besides, the challenges presented in the previous section Visual odometry is a technique that can be implemented in indoor and outdoor environments with good results, so it can be used as a tool in autonomous driving projects in different conditions. Due to the advance in the tasks of image processing and artificial vision of deep learning, Visual Odometry

has been benefited, because different strategies have been implemented to improve the estimation of the path or trajectory done by the object. Another advantage that should be considered is that with visual odometry is possible to integrate different sensors to improve the results, for instance, the first sensor integrated is a camera, but can also be used as an IMU, or a LIDAR sensor depending on how is the approach to calculating the odometry in the work.
An important feature to consider is that wheel slippage on uneven ground or in other difficult circumstances has little impact on VO. Additionally, VO is efficient in areas without GPS [21]. Also, VO does not release any observable energy into the surroundings, in contrast to laser and sonar localization techniques. Additionally, unlike GPS, VO does not require the presence of additional signals [22]. The advantages of using cameras for robot localization over other sensors include cost savings, the ability to easily integrate ego-motion data into other vision-based algorithms, such as obstacle, pedestrian, and lane detection, and the absence of the need for sensor calibration [23]. Cameras are compact, affordable, light, low-powered, and adaptable. Thus, they can be used for additional robotic activities and in any kind of vehicle that could do trajectories on land, sea, or air to do tasks such as object detection and recognition.
A binocular camera consists of two lenses, each with its own image sensor. Since the beginning of 2004 it has been used on Mars to calculate robot motion (Nistér et al. 2006). The image scale may be instantly and instantly obtained because the size of the stereo baseline is set and known, leading to an effective and accurate triangulation process. Information on the third dimension, or depth, can be derived from a single frame. Additionally, both types of cameras' varied features improve tracking in succeeding frames (Gonzalez et al. 2012; Nistér et al. 2006). Stereo cameras cost more money than regular cameras, though. Additionally, binocular cameras need more work to calibrate than monocular cameras, and calibration errors

2.1.4 Cameras used in Visual Odometry

Visual Odometry has numerous methodologies to be implemented which can include different types of cameras, for instance, stereo, monocular, stereo, or monocular omnidirectional, and also RGB-D can be used for works with Visual odometry. The most common methodologies that can be found in the literature are with stereo or monocular cameras, so it can be classified as Stereo or Monocular Visual Odometry. The systems that use binocular cameras can be considered also Stereo Visual Odometry [19].
Something to consider is that Stereo cameras are more expensive than regular cameras, though. Additionally, the calibration process for binocular cameras is more labor-intensive than for monocular cameras, and errors in calibration have a

direct impact on the motion estimation process [24]. The two photos of the stereo pair must also be collected at the exact same time interval for stereo VO, which is crucial. According to Kreo et al. [25], this can be done by synchronizing the shutter speeds of the stereo vision system's two cameras or by synchronizing the cameras with an external trigger signal supplied by the controlling PC through a serial or parallel port.
The impact of calibration mistakes in motion estimation is lessened when using a monocular camera. The key reasons for employing the monocular camera in many common applications, such as cell phones, and laptops, are its low cost and ease of deployment. A disadvantage of monocular vision is that it can present scale uncertainty, so if the surface is uneven, the image scale will fluctuate, and for that reason, the image scale factor could be difficult to estimate. If there is a significant change in the road's slope, the scaling factor estimation may be inaccurate, which could result in an inaccurate prediction of the trajectory that results. Small robotics generally benefit more from monocular VO systems than stereo VO systems because they save the space of the baseline between the two stereo cameras. Additionally, stereo cameras are more challenging to interface with and synchronize with than monocular cameras.
The omnidirectional monocameras are also used in Visual Odometry applications and have the advantage that giving more information than common monocameras. For instance, Seok, H and Lim, J develop a work name "ROVO: Robust Omnidirectional Visual Odometry for Wide-baseline Wide-FOV Camera System" in which proposed a methodology with an omnidirectional monocamera to maximize the stability and accuracy of the pose estimation. They used four cameras with 220° Field Of Vision (FOV), to be able to observe a full 360° angle around the robot, which allows doing the stereo triangulation calculation possible because the environment visible had more than two views. The authors added a hybrid projection model, online extrinsic calibration and a multi-view P3P RANSAC algorithm [26].

Scale uncertainty

Scale uncertainty can negatively affect the monocular visual odometry systems. In the case of stereo systems the scale motion can be recovered by using the baseline between the two cameras as a reference. A disadvantage of the monocular Vision Odometry is that if the camera motion is constrained, the scale ambiguity is unsolvable. Based on several works when happens that large changes in road slope occurs, do the estimation of the scaling factor may become mistaken that can cause a estimation of the trajectory with high level error. Because the absolute image scale is unknown, the relative scale with respect to the prior frames is derived using either understanding of the 3D structure or the trifocal tensor. As a result, the

absolute scale can be established through direct measurements, such as estimating the size of an object in a picture, mobility restrictions, or integration with other sensors like an inertial measurement unit (IMU) and range sensors.
When the camera motion is restrained to a surface, the image scale can be recovered. Several works implemented different methodologies to solves this issue, for instance, Kitt et.al b[24] By utilizing the Ackermann steering model and assuming that the car travels on a planar surface, the problem of picture size ambiguity was resolved.In certain circumstances, a hybrid strategy that blends feature- and appearance-based strategies are the best option. They integrate the usage of pixel intensity data from the complete set of images with the tracking of significant features over frames. For instance, Scaramuzza and Siegwart employed the hybrid technique because the appearance-based strategy by itself was not very resistive against picture occlusions. As a result, in their work, the vehicle's rotation was calculated using the picture appearance, whereas the vehicle's translation was estimated using image features from the ground plane.
The RANSAC method was employed for outlier elimination, and the Kanade-Lucas-Tomasi (KLT) feature tracker was used to extract features. Through SVD decomposition, the relative stance between two successive frames was retrieved from the crucial matrix. The scale ambiguity problem was resolved by employing the restrictions of camera mounting and the ground planar assumption because the absolute size of the translation cannot be determined through monocular motion estimate. The image was divided into regions in order to identify barriers and distinguish between the ground and obstacle areas. According to three factors—homography constraint, feature point distribution, and boundary point reconstruction—each region was categorized as either on the ground or off the ground.

2.1.5 Monocular Vision Odometry

In Monocular Visual Odometry the feature points need to be observed in at least three different frames. A problem of this methodology is that the transformation between the first two consecutive frames is not fully known and is usually set to a predefined value, so the global scale can be obtained using sensors such as IMUs, wheel encoders, or GPS. The procedure to obtain the camera pose in this case is:

1. Extract features in the first frame and assigns descriptors to them
2. Extract features in the next frame and assign descriptors
3. Match features between the two consecutive frames
4. Estimate a transformation between the first two frames and triangulate the corresponding points using this transformation.

Figure 2.6: Cameras use in Visual Odometry

5. Extract features in the following frame, match them with the previously extracted features from the previous frame
6. Use RANSAC to refine the matches and estimate the transformation that gives the minimum sum of square differences between the observed features in the current and the re-projected 3D points
7. Triangulate the matched feature pairs into 3D points using the estimate transformation.

2.1.6 Stereo Vision Odometry

In Stereo Visual Odometry the motion is estimated by observing features in two successive frames, the procedure is the following [4]:

1. Extract and match features in the right and left frame (in time), reconstruct points in 3D by triangulation.
2. Match these features with their corresponding features in the next frames.
3. Estimate the transformation that gives the minimum sum of squares differences between the observed features in one of the camera images and the re-projected 3D points that were reconstructed in the previous frame.
4. Use RANSAC type refinement step to recalculate the transformation based on inner points only.

5. Concatenate the obtained transformation with previously estimated global transformation

2.1.7 Deep learning in Visual Odometry

The estimation of Visual odometry is a strategy used when is needed to calculate the path that an object is following. Several works have been developed regarding this task, for instance, Holder J,C and Breckton T,B in their work Learning to Drive: Using Visual Odometry to Bootstrap Deep Learning for Off-Road Path Prediction, wanted based on a single image of the surroundings captured by a forward-facing vehicle-mounted camera, anticipate the route a human driver would take in an off-road scenario. For this problem, they made a visual, end-to-end autonomous driving approach suggestion. A CNN was trained to directly map the path of an upcoming car to pixels in an image taken by a forward-facing camera. The training was carried out using stereoscopic visual odometry to track the motion of a human-driven vehicle through a series of images and map this motion into the image space of the initial frame so that pixels that the vehicle traverses are labelled as "path" (This approach allows to predict a path taking into account of future changes in direction and does not rely on a direct link to driver inputs. They train three cutting-edge CNN architectures, each of which is built to handle a specific segmentation or classification task, using automatically labeled data. The CNN architectures used were Segnet, U-Net, and Fully Convolutional Networks (FCN). The VGG16 network on which the encoder was built has its fully connected layers removed. The encoder comprises 5 max-pooling layers with a down-sampling ratio of 2, 13 convolution layers with 3x3 kernels, batch normalization, and rectified linear units in between [27].
Ruihao L, et al proposed a UnDeepVo which is a monocular VO system based on an unsupervised deep learning scheme. In this work, the system was composed of a pose estimator and depth estimator and used monocular images with a VGG foundation. The 6-Dof transformation between two consecutive monocular pictures is predicted using the input of two monocular images. To create dense depth maps, the depth estimator primarily uses an encoder-decoder architecture. The one used here is intended to accurately forecast depth maps. Left and right pictures are fed to the network during training. They employed the input stereo images, taking into account both the spatial and temporal geometric constancy of a stereo image, to develop the loss function [28].
Mohamed et.al in their work “Towards dynamic monocular visual odometry based on an event camera and IMU sensor” proposed a hybrid technique that is composed by first using the number of events for frame to achieved the minimum quantity of data processed for a scene but keeping the accuracy of pose estimation, and the other is to use a frame based camera as a reference for a even-based camera output,

also they used the velocity of the camera to determines how fast the environment changes using a IMU and extracting the acceleration data [2]

Muller P, et.al suggested a method in which the odometry is calculated in three steps basically. The first is calculate the optical flow with FlowNetS, then use this as input to the FlowNetS again which will be referred to as Flowdometry, to produce inter-frame odometry estimations. Finally, the estimations are accumulated to generate maps and accuracy measurements [6]

M He, et. al in their work "A review of monocular visual odometry" studied different strategies to estimate the Visual Odometry [29]

2.2 Optical Flow

The optical flow is calculated by analyzing the projected Spatio-temporal patterns of moving objects in an image plane and its value at a pixel specifies how much that the pixel was moved in the sequential image, so basically it allows measuring the relative motion between objects and the viewer. To work with the optical flow, it was used the Based on Intensity Coherence assumption that states that the image brightness of a point projected on two successive images are constant or nearly constant, or almost constant. Basically to be able to work with Optical Flow is necessary to do two important assumption:

- The intensities of the pixel of an object do not change between consecutive frames.
- The pixels that are neighbors have a simirlar motion.

To find the Optical flow equation, it should be considered a pixels $I(x, y, t)$, that moves a distance (dx, dy) in the enxt frame after dt time. Considering the two assumptions exposed before, is possible to say that the intensity of those pixels is the same and the intensity does not change, so:

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \tag{2.1}$$

If the taylor series of right-hand side, is possible to remove common terms, then if the expression is divided by dt. The next expression shows up:

$$f_x u + f_y v + f_t = 0 \tag{2.2}$$

$$f_x = \frac{\partial f}{\partial x}; f_y = \frac{\partial f}{\partial y}; \tag{2.3}$$

$$u = \frac{dx}{dt}; v = \frac{dy}{dt}; \tag{2.4}$$

The equation 2.2 is called the Optical Flow equation. f_x and f_y are the image gradients and f_t is the gradient along time. The problem is that in this expression (u, v) are not known or is not possible to find a solution with traditional methods. Several methods have been proposed to solve this problem and the most used is the Lucas-Kanade method.

2.2.1 Types of optical flow

It is possible to classify optical flow into two main types: Sparse Optical Flow and Dense Optical flow. While dense optical flow provides the flow vectors of the full frame, which means all pixels, sparse optical flow only provides the flow vectors of a selection of important characteristics, for example, a small number of pixels showing an object's borders or corners. Dense optical flow has more precision but is slower and more computationally expensive, as you could have anticipated.

Sparse Optical Flow

In order to track its velocity vectors, sparse optical flow chooses a sparse feature set of pixels, for instance, attractive features like edges and corners motion. To guarantee that the same spots are being tracked, the extracted features are given into the optical flow function from frame to frame. Sparse optical flow can be implemented using a variety of techniques, such as the Lucas-Kanade approach, the Horn-Schunck method, the Buxton-Buxton method, and others. The Lucas-Kanade approach will be put into practice using OpenCV, an open-source library of computer vision algorithms.

Implementation of the Sparse Optical Flow

Only a feature set of pixels are tracked in order to perform sparse optical flow. Points of interest in pictures are known as features, and they provide rich image content information. Corners, for instance, are points in the image that are resistant to changes in translation, scale, rotation, and intensity. The renowned Harris Corner Detector, which may be used via the following three processes, is quite similar to the Shi-Tomasi Corner Detector.

1. Find windows as small image patches that, when translated in both the x and the y axes, have strong gradients which can be traduced in variations in image intensity.

2. For each patch, compute R

3. Classified R as a flat, edge, or corner, according to the value R

The difference between Harris Corner Detector and Shi and Tomasi is in the equation on which the score R is calculated because in the Harris Corner Detector, the scoring function is given by:

$$R = detM - k(traceM)^2 \tag{2.5}$$

$$traceM = \lambda_1 + \lambda_2 \tag{2.6}$$

An what Shi-Tomase implemented is:

$$R = min(\lambda_1, \lambda_2) \tag{2.7}$$

This expression essentially says that a corner is considered if R exceeds a certain threshold.

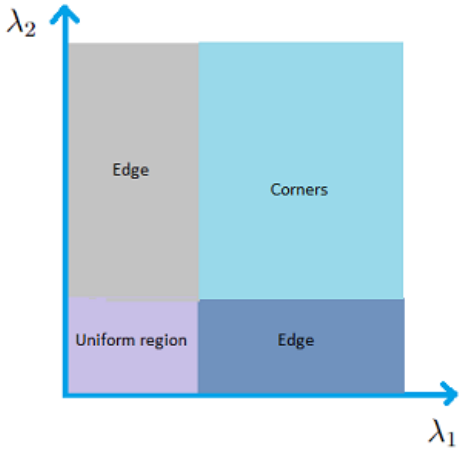

Figure 2.7: Scoring function of the Shi and Tomase

As is shown in the previous graphic in the space of λ_1 and λ_2, the window can be classified as a corner just when λ_1 and λ_2 are over a minimum λ that as can be seen in the plot is in the limits of the Uniform region.

Dense Optical Flow

Computing the dense optical flow, the optical flow vector is calculated for each pixel in each frame. While the computation may take longer, the outcome is more accurate and dense, making it useful for applications like video segmentation and learning structure from motion. Prior and current photos are analyzed by the

algorithm, which then adds an estimate of motion to an output image. There are numerous implementations of the Dense Optical Flow, but the most common is the Farneback method

Implementation of the Dense Optical Flow

In the work Two-Frame Mobility Estimation Based on Polynomial Expansion, Farneback, G proposed a useful method to estimate the motion of important characteristics by comparing two successive frames [30].
First, the method uses a polynomial expansion transform to approximate the windows of image frames using quadratic polynomials. Second, a method to estimate displacement fields from polynomial expansion coefficients is defined by examining how the polynomial transforms under translation (motion). Dense optical flow is computed following a number of improvements.
This algorithm could be implemented in OpenCV, and what is done is that it calculates the direction and magnitude of the Optical Flow from 2-channel array of flow vectors $\frac{dx}{dt}$, $\frac{dy}{dt}$, that is the optical flow problem. The next step is to depict the flow's angle that is the direction using color and its distance that is the magnitude using HSV (Hue, Saturation, Value) color values. For the best visibility, HSV's strength is always set to a maximum of 255.

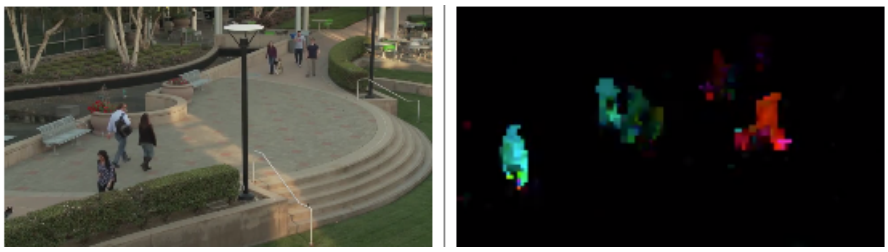

Figure 2.8: Example of Dense Optical Flow estimation

2.2.2 Lucas-Kanade method

In their study titled An Iterative Image Registration Technique with an Application to Stereo Vision [31], Lucas and Kanade suggested a useful method to predict the mobility of important features by comparing two successive frames. The following presumptions govern how the Lucas-Kanade approach operates:

- The separation between the two consecutive frames of interest should of a small time variation, to do not have a large displacement of the Objects, which means that this method has a better behaviour in situations of slow-moving objects.

- A frame shows a natural scene with textured items that have smoothly varying grayscale tones.

In the Lucas-Kanade approach, the point is surrounded by a 3x3 patch or window. The motion is assumed the same at each of the nine places, so it is possible to find (f_x, f_y, f_t) for these 9 points. In the figure 2.9 is shown the process in which is selected the window in the image and then how can be localized the 9 points from which the Optical Flow is going to be estimated.

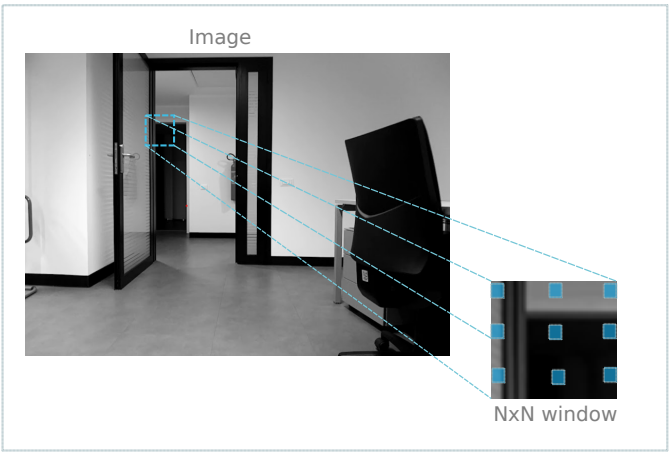

Figure 2.9: Lucas-Kanade points estimation

This can be represented as:

$$I_x(q_1)V_x + I_y(q_1)V_y = -I_t(q_1)$$

$$I_x(q_2)V_x + I_y(q_2)V_y = -I_t(q_2)$$

$$...$$

$$I_x(q_n)V_x + I_y(q_n)V_y = -I_t(q_n)$$

Where $q_1, q_2, ..., q_n$ are pixels inside the window shown in the figure 2.9, that for the purpose of the Lucas-Kanade algorithm $n = 9$ and the size of the window in $3x3$. $I_x(q_n), I_y(q_n)$ and $I_t(q_n)$ represent the partial derivatives of the image, respect with the position (x, y) and time, for each pixel at the current time. So, the problem transforms into the solving of 9 equations with two unknown variables that makes this problem into an over-determined. The least square fit method yields a better result. The final answer, which consists of two equations and two unknown problems that must be solved, is shown below.

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_i I_x(q_i)^2 & \sum_i I_x(q_i) I_y(q_i) \\ \sum_i I_y(q_i) I_x(q_i) & \sum_i I_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i I_x(q_i) I_t(q_i) \\ -\sum_i I_y(q_i) I_t(q_i) \end{bmatrix}$$

Figure 2.10: New Optical Flow equation

In this new expression $V_x = u = \frac{dx}{dt}$ means the displacement of x over time and $V_y = v = \frac{dy}{dt}$ is the displacement of y over time. If the value of this two variables is found the Optical Flow problem is complete

2.2.3 Metric maps commonly use

The most common metrics maps used for the optical flow are:

Feature maps

The feature maps represent the environment as points and straight lines. Each feature is described by location and geometric shape. To localize a point it is used a feature detection technique and then the features are compared with the map features already stored. The biggest weakness of this method is the sensitivity to false data association.

Occupancy grids

The occupancy grids represent each region of the environment with an array of cells.

Topological maps

The topological maps are used when the environments are large because it reduce the size of the information to the most important. This metric is more interested in adjacency information between objects, they are typically represented by a graph, where nodes are points that define locations or landmarks, contain information about them, and connect other points.

2.2.4 Convolutional Neural Networks for the estimation of the Optical Flow

In general the Optical Flow estimation problem has been an optimization problem. The classical energy-based models, which treat optical flow estimate as an energy minimization problem, have previously dominated the field. In recent years have

been developed different strategies to calculate the optical flow to improve the results obtained. The adoption of Convolutional Neural Networks (CNNs) have been increased in the context of motion estimation, to the point where CNNs approaches are now the state of the art in terms of accuracy, as the practical advantages of CNNs over conventional methods have become apparent in many areas of computer vision and beyond. For this reason, one of the strategies used to innovate and try to improve the results obtained for the optical flow is the implementation of deep learning. Usually the approach taken is to consider two video frames as input to obtain as the output the Optical Flow, this can be express as:

$$(u, v) = f(I_{t-1}, I_t) \tag{2.8}$$

In this expression u is the motion in the x direction, v is the motion in the y direction, and f is the neural network that takes in two consecutive frames I_{t-1} and I_t as input.
Deep neural networks need a lot of training data, which is particularly difficult to come by, in order to compute optical flow. This is due to the fact that categorizing video material for optical flow necessitates precisely determining the motion of every single point in a picture down to the subpixel level. Researchers used computer graphics to build vastly realistic worlds in order to address the problem of classifying training data. Since the worlds are created through instructions, it is possible to calculate the motion of each and every point in an image during a video sequence. Examples of this include Moving Chairs, a dataset of several chairs flying across random backgrounds, and MPI-Sintel, an open-source Computed Generated Imagery (CGI) movie with optical flow labeling created for multiple sequences.

2.2.5 Recent Convolutional Neural Networks for Optical Flow estimation

Flownet 2.0

The Flownet 2.0 was developed by Mayer, E.I et.al in 2016. They did several contributions that helps to improve the results in the estimation of the Optical Flow. For instance, it was introduced a training schedule that was made with different datasets, also a warping operation was introduced, that was used to stack multiple networks to improve the results. Another important contribution was that to obtain numerous variants of the networks, the authors made variations in the depth of the stack and size of the components, which allows controlling the ratio between accuracy and computational resources. In this work was important to improve the results in different scenarios, for that reason the authors focused on small, subpixel motion and real-world data, so they created a new dataset to train

the network, then to obtain a better performance in arbitrary displacements, it was added an additional network that learns to fuse the former network with the small displacement network in the correct form. [32]

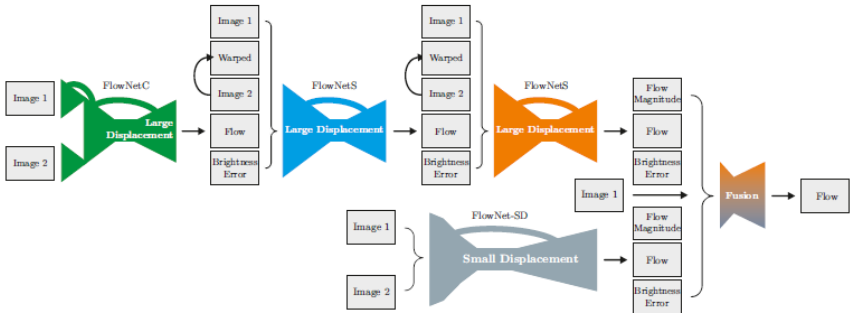

Figure 2.11: FlowNet 2.0 architecture

As is shown in the image below the architecture of the CNN compos

f three main operations that are: computation of the optical flow of large displacement, of small displacement, and the fusion between these two. For the calculation of the optical flow in large displacement, there were used the FlowNetC and two FlowNetS, for the optical flow for small displacement was used the FlowNet-SD, and finally the Fusion network to calculate the final estimate.

RAFT

The Recurrent All-Pairs Field Transform for Optical Flow was developed by Teed Z, and Deng J, in 2020. The Raft has numerous strengths over other works done in the past. Regarding the accuracy of the estimation of the optical flow in two very important datasets in this area, that are: KITTI and SINTEL, the RAFT obtained an error reduction of the 16% and 30% respectively. Another advantage of the RAFT is that has high efficiency, for instance, on a 1080Ti GPU, RAFT processes 1088436 videos at 10 frames per second. Compared to other designs, it trains with 10X less iterations. On Sintel, a scaled-down version of RAFT that uses only a fifth of the parameters can still outperform all earlier techniques by a factor of 20. Traditional approaches centered on optimization are the driving forces behind the RAFT design. Per-pixel features are extracted using the feature encoder. The correlation layer determines how visually similar two pixels are. The iterative optimization algorithm's steps are mimicked by the update operator. However, in contrast to conventional methods, features and motion priors are learned by the feature encoder and the update operator, respectively [33].

STARFLOW

A SpatioTemporal Recurrent Cell for Lightweight Multi-Frame Optical Flow Estimation (StarFlow) was developed by Godet, P et. al in 2020. The work is based on the doubly recurrent network over spatial scales and time instants. The network within a single processing cell known as a STaR cell (for SpatioTemporal Recurrent cell), which stands for SpatioTemporal Recurrent cell, it explicitly accounts for the knowledge from earlier frames and the redundancy of the estimation at each network size. The STaR cell outputs the Optical Flow and occlusion map at the current image scale and time instant based on data from the past and from a lower scale. The STaRFlow model is produced by continually invoking this cell over scales in a coarse-to-fine scheme and over sets of N subsequent frames. We achieve a lightweight model with this doubly recurrent structure and weight sharing between processes for flow estimates and occlusion detection. [34]

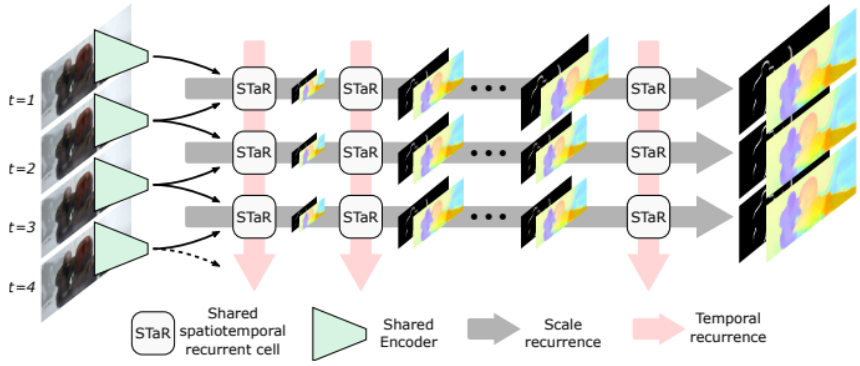

Figure 2.12: Starflow architecture

The architecture of this network consists in the basic application of the same SpatioTemporal Recurrent cell in repetition in regard to the features retrieved from each image of the series in terms of time and spatial scale. As is shown in the figure below a shared encoder, from which architecture is derived, is used for feature extraction and is represented with the green box. The scale recurrence, as horizontal gray arrows, involves supplying the STaR cell with features taken from the current frame and with the OF and occlusions from the previous scale at each scale. Vertical pink arrows are used to represent the data flow associated with temporal recurrence, which transports learn features from one time instant to the next.

2.2.6 Optical Flow applications

To know in which situations is useful to estimate the Optical Flow as an approach to solve the problem first, is important to know that since the Optical Flow is

the projection of the motion of an object onto an image plane. It is needed to be considered that exist numerous real-world motions that can cause the velocity vectors. Translation and rotation are the two main motions to be considered. In the next figure 2.15 is shown the Optical Flow vectors that can be obtained based on the movement made by the camera:

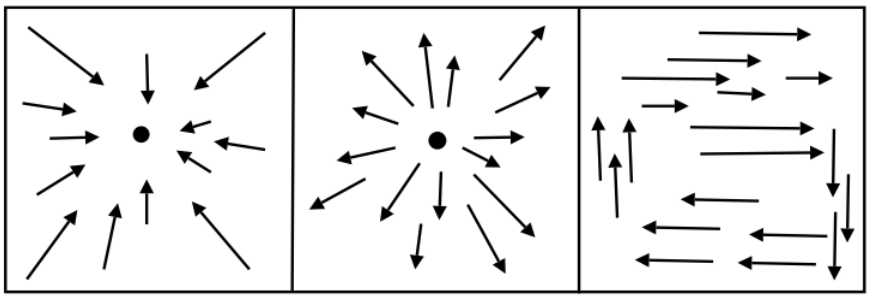

Figure 2.13: Types of Optical Flow vectors

In the left side are shown the Optical Flow vectors when the camera is moving in reverse, that cause all the vectors converge around a point that is call Focus Of Contraction (FOC). In the middle are show the Optical Flow vectors that are the results when the camera is moving ahead, so all the vectors diverge of the FOC. In the right picture are the vectors that represents the movement of a field of parallel flow vectors created by four moving objects perpendicular to the image plane converge at infinity.

Time to Contact and Focus of Expansion

The calculation of the time-to-contact or time-to-collision is one of the most useful uses of an optical flow field. Knowing how long it will take an autonomous robot to arrive at a specific location, assuming constant velocity, is helpful if you can picture a robot navigating through an environment. It's interesting to note that this can be discovered without any knowledge of the required distance or robot movement speed. However, the rotating component of the flow field will not be relevant to our computations; we only require access to the translation component. To do this, It is needed first be able to determine the flow field's focus of expansion. Theoretically, there is only need two vectors because it can easily find the points where their respective lines intersect. The expansion will be concentrated on this. Obviously, inaccurate optical flow vectors will be observed in reality due to noise and other flaws resulting from the numerous steps required to get there.

Segmentation

The aim of semantic segmentation, for instance, is to divide a picture into a number of regions belonging to distinct item classes, however single frame segmentation approaches frequently struggle with closely spaced objects with identical textures. However, if the objects are arranged independently, the different motions of the objects may be quite useful where discontinuity in the dense optical flow field correlate to boundaries between objects.
However, the issue is significantly simplified if the items are traveling independently of one another. There are various broad strategies, much like single picture segmentation. Segmentation based on local regions is comparable to edge-based segmentation in several ways. These result from the finding that borders between objects frequently correspond to flow field zones of discontinuity. Additionally, there are global algorithms that isolate entire sections of the image with identical motion, much like region-based segmentation does.

Figure 2.14: Example of segmentation using Optical Flow [35]

Tracking and Object Detection

Due to the fact that sparse optical flow relies on tracking points of interest, such real-time systems may be carried out via feature-based optical flow approaches from either a stationary camera or cameras mounted on vehicles. Thanks to features that can be extracted from the Optical Flow, the works that can be developed could be from basics applications to a more complex work. An area in development is the use of Optical Flow for traffic analysis and vehicle tracking, for instance, Smith and Brady developed a system for identifying and following moving objects, a clearly essential component of an autonomous navigation system [36]. Their technology tracks moving objects from a camera that is either stationary or mounted on a moving object itself. It is used for traffic studies. The system operates in three main stages: cluster tracking and filtering, feature-based optical flow estimates, and flow segmentation into clusters. Additionally, it allows for vehicle occlusion.
Calculating the optical flow field for a single frame is the initial step. The feature detector is used to detect the flow vectors using a feature-based approach. Due to

time limits, feature tracking—a important field in and of itself—which matches detected features—is not covered in this work. Feature tracking is the process of tracking features across several frames by estimating motion models for each feature. Additionally, models are regularly updated. The feature tracker then returns a flow field of constant flow vectors, either the feature's instantaneous displacement or the displacement over the previous n frames. Giving the displacement over several frames makes it less likely that the characteristics will be mistaken for noise or false signals.
The flow field will then be divided into distinct clusters in the following phase. An motion model for each cluster is fitted using a straightforward least squares method. A flow vector is used to create a new motion and launch a new cluster for a specific portion of the image. The present affine model is used to calculate the error from adding neighbors of that flow vector. The expression that the authors used to calculated the error is:

$$E = \frac{|v - v_a|}{\frac{|v|+|v_a|}{2} + \omega} \tag{2.9}$$

In this expression v is the current vector flow, v_a is the vector that the model estimates for the position and ω is the error of the current estimate.
The procedure is that if the error is less that the threshold selected, the vector will be add to the cluster and the current model updated. The number of clusters increase depending on the neighbors and the Optical Flow vectors from the list when they are assigned to a cluster. When the moment in which it can not been created more clusters happens. The clusters must be passed to the tracker. A list of prior clusters is compared with the clusters from the current frame by the cluster tracker and filter. The motion model, the bounding box's shape, and the cluster centroid are all used in the matching process. We assign the cluster to an earlier one and update the data if the matching error is under a certain limit. If not, the cluster is included in the list. A higher level is then given a filtered list of clusters to understand. [37]

2.3 Depth Map

The depth map is a picture or an image channel that shows how far away the surfaces of scene objects are from the perspective. For that reason to know the geometric relationships inside a scene it is require some degree of depth estimation. One of the most crucial problems in image processing and computer vision is the extraction of depth information from 2D pictures. The depth information can be used for depth-based image manipulation, scene interpretation, scene reconstruction, image refocusing, and 2D to 3D conversion. The depth information can be obtain with several techniques as: depth from motion, depth from focus and stereo vision.

Figure 2.15: Example of object tracking and detection [38]

Depth map from Stereo images

It is possible to calculate the information of the depth map of two stereo images, for that is necessary to consider the following:

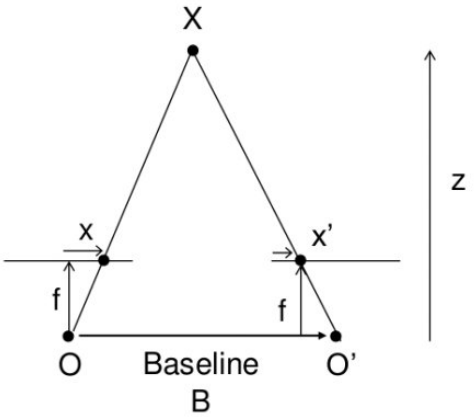

Figure 2.16: Diagram to calculate the Depth map

The previous figure 2.16 contains three equivalent triangle and applying geometry it is possible to obtain the next result:

$$disparity = x - x' = \frac{Bf}{Z} \tag{2.10}$$

In this expression x and x' are the separations between the image-plane points that correspond to the 3D scene points and their camera centers. B is the distance

between the two cameras that should be known and f is the focal length of the camera that also should be known. Therefore, the above equation essentially states that the distance between corresponding picture points and their camera centers and the depth of a point in a scene are inversely related. It can be therefore calculate the depth of each pixel in an image using this information.

Depth map from monocular images

The estimation of the depth map with monocular images has been a critical problem since time ago in numerous applications of real-world scenarios. The statistics of the depth information, for instance, can be effectively used to estimate horizontal limits or the location of the vanishing point, which is highly helpful to immediately comprehend a given image. Inferring the depth information has recently become crucial in the field of autonomous driving systems since these hints frequently provide exceptional advantages of understanding the 3D geometrical layout. Due to the abundance of options, many academics have worked very hard to find a solution to the monocular depth estimation problem.
Considering the success of the deep learning, many works started to find a methodology to be able to estimate the depth map. In order to do this, the problem of the depth estimation was formulated as the problem of image translation. Which refers to the translation from the color image to the depth one. The convolution neural network (CNN) has been widely embraced as the foundational architecture of the generative model to extract underlying features pertinent to the depth information. The most common approach to take is used a 3D sensors, such as LiDAR, Kinect, etc to scanned the depth information, and use this as a ground truth for the Convolution Neural Network selected.

LiDAR sensor

The term LiDAR stands for Light Detection And Ranging. In LiDAR, laser light is emitted from a transmitter and reflected by the scene's objects. The system receiver picks up the reflected light, and the Time Of Flight (TOF) is utilized to create a distance map of the scene's objects. LiDAR is essentially a ranging tool that calculates the distance to a target. Sending a brief laser pulse and timing the interval between it and the detection of the reflected (back-scattered) light pulse are used to determine the distance.
To "scan" the object space, a LiDAR system may employ a scan mirror, many laser beams, or other techniques. LiDAR can be used to address a variety of issues since it can deliver precise distance measurements. LiDAR systems are employed in remote sensing to assess atmospheric particle or molecule scatter, absorption, or re-emission. The systems may have certain demands regarding the laser beams' wavelength for various uses. It is possible to detect the amount of a particular

molecular species in the atmosphere, such as methane and aerosol loading. The size of raindrops in the atmosphere can be used to gauge a storm's distance and rate of precipitation. [39]

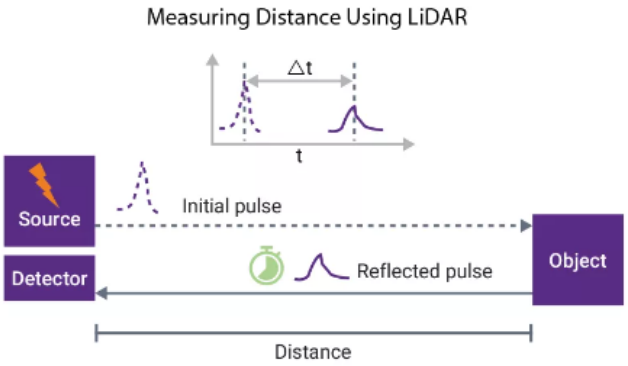

Figure 2.17: LiDAR sensor work

Challenges for LiDAR

LiDAR is a ranging tool that calculates the separation from a target. Sending a brief laser pulse and timing the interval between it and the detection of the reflected (back-scattered) light pulse are used to determine the distance. Operational LiDAR systems face a number of well-known difficulties. Depending on the type of LiDAR system, certain difficulties exist:

- The separation and rejection of the beam's output signal The probing beam typically has a substantially higher brightness than the return beam. To prevent the detector from becoming saturated and unable to detect external targets, care must be taken to ensure that the probing beam is neither reflected or dispersed by the system back into the receiver.
- Limitations on the amount of optical power that can be used - A system that has more power in the beam offers more precision but costs more to operate.
- When the laser source is operating at a frequency that is hazardous to human eyes, scanning speed-Safety can become a problem. Other methods, including flash LiDAR, which illuminates a vast area all at once and operates at eye-safe wavelengths, are reducing the severity of this problem.
- Device crosstalk signals from surrounding LiDAR devices could obstruct the desired signal. The current problem is figuring out how to distinguish signals from surrounding LiDAR systems. Different strategies including signal crackling and isolation are being developed.

Time of Flight

Time of Flight (ToF) is a term used in computer vision to describe the idea of timing the passage of light over a specific distance. Since the necessary time is directly proportional to the distance, the distance between the emitter and receiver—which are typically merged into a single device—can be estimated using the speed of light. Typically, LEDs or lasers emit the infrared spectrum of light. There are numerous conceivable implementations; for example, direct ToF and indirect ToF systems are differentiated from scanning-based light detection and ranging (LiDAR) systems, and flash-based Time of Flight cameras from indirect ToF systems.
Cameras utilizing the ToF concept are susceptible to interference from other cameras or outside light sources that produce in the same wavelength, similar to structured light. This can be fixed with multi-camera setups by synchronizing the cameras. The ability to obtain depth information from surfaces with little to no roughness as well as the high precision and independence from outside light sources are all advantages. There are two types of ToF:

- Direct ToF: refers to the practice of producing a single pulse and measuring the distance using the time interval between that pulse and its reflection that is received.

- Indirect ToF: employs a continuous stream of light that is manipulated or coded. The difference in phase between the emitted and received reflected light is then used to determine the distance.

The direct ToF is more implemented in scanning based LiDARs and the indirect are more use in flash-based cameras. An advantage of the last one is that has a higher accuracy without use to high sampling rates of the laser light pulse. As a result, it is possible to capture at minimal cost at higher resolutions and fields of view. In the figure 2.18 is shown the two types of ToF [40].

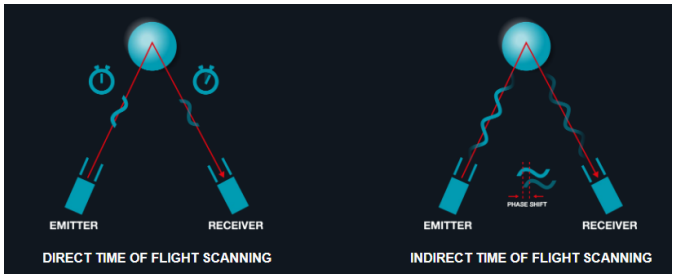

Figure 2.18: Direct and Indirect Time of Flight

2.4 Key Feature point detectors

Locating unique key points, or the positions of the most recognizable elements on each image, is a generic and fundamental way to finding features. Once the content has been normalized around the important locations, a local descriptor can be computed. The key point's outward appearance is described by the local descriptor, a vector of numbers. These can then be used to compare and contrast important details between several photos. Finding stable and distinguishable local regions from images is the goal of a good detector. Furthermore, despite the fact that the local regions have been altered by viewpoint, illumination, scale, blur, and compression, these detectors continue to locate or identify them. The design of conventional methods is based on the previous mathematical theory, known as a customized operator and divided into corner, binary corner, and blob. The most use and the key point feature points detectors that show better results are going to be explain in the following sections.

2.4.1 Harris Corner detector

The Harris Corner detector is classified into the gradient-based detector and intensity-based detector. This method was proposed by Harris C, Shi M, and Tomasi to identify a point as a candidate of a corner points by the computation of the gradient. The detector frequently searches for corners since there are noticeable fluctuations in intensity in all directions. This method is divided in three main steps:

1. Change the intensity for the shift $[u, v]$

$$E(u,v) = \sum w(x,y)[I(x+u, y+u) - I(x,y)]^2 \tag{2.11}$$

   In the equation the element $w(x,y)$ is the window function, $I(x+u, y+u)$ is the shifted intensity, and $I(x,y)$ is the intensity. The window function can be Gaussian filter or rectangular and the interval inside is a constant and outside is zero.

2. The change of intensity can be measure by the equation below:

$$\begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_x^2 \end{bmatrix} \tag{2.12}$$

   Where M is going to be equal to:

$$M = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \tag{2.13}$$

   The elements of M correspond if the region is a flat region, corner or edge.

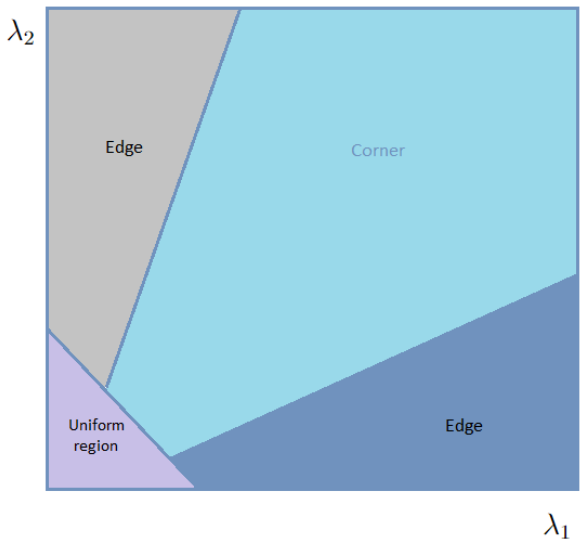

Figure 2.19: Scoring function for Harris detector

3. The values of λ_1 and λ_2 are classified based on the thresholds. As is possible to see in the image 2.21, if both λ_1 and λ_2 are small, the region is uniform, the region is an edge if λ_1 is small and λ_2 is large or vice versa, and if both are large the region is a corner

4. The harris detector used the next equation, to find the key points in the image

$$R = det(M) - k(trace(M))^2 \tag{2.14}$$

2.4.2 SIFT detector

Scale-invariant Feature Transform (SIFT) is a detector with the properties of rotation-invariant and scale-invariant. The advantage of this method is that, in order to identify the same key points independently in each image, SIFT maximizes the Difference of Gaussians (DoG) in size and in space. DoG is essentially the variation in the amount of Gaussian blurring applied to an image depending on its standard deviation. The image is scaled and each octave is blurred using Gaussians with various scaling factors' standard deviations. DoG is determined from the discrepancies between neighboring Gaussian-blurred images. For every octave of the scaled image, the operation is repeated.
Once that the DoG is found, the Scale-invariant Feature transform detector is going to search the DoG over scale and space for local extremas, which can be potential

keypoints. A pixel is a local extreme and a potential keypoints in that scale if it is larger or smaller than all of its neighbors [41].

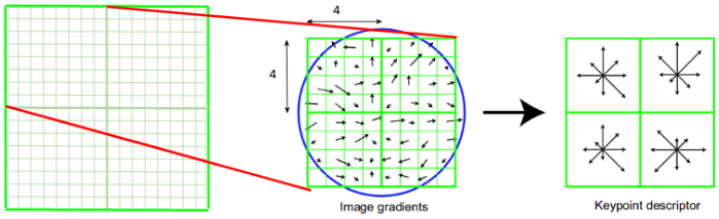

Figure 2.20: Key-point descriptor estimation by SIFT

SIFT Descriptor

Once the keypoint has been identified, the next step is to build a descriptor that contains details about the visual traits surrounding the keypoint while being insensitive to rotation and picture illumination. to build the SIFT descriptor is necessary to follow the next steps

1. Use Gaussian blurred with the image previously associated with the scale of the key point
2. Take an array over 16x16 as image gradient
3. Relative to the keypoint orientation, rotate the gradient locations AND directions.
4. Create an array of orientation histograms
5. Add into the local orientation histograms the rotated gradients with 8 orientation bins
6. A length 128 vector containing a 4x4 histogram array with 8 orientation bins per histogram makes up the resultant SIFT description.

FAST detector

Features from Accelerated Segment Test (FAST) was proposed by Rosten, E and Drummond, T. One of its advantages is that has a high efficiency in comparison with other methods. The FAST detector what does is first, takes a candidate point, which is identified as a corner if around it there are contiguous pixels on the circle template that must be brighter than the intensity of the candidate pixel $I_p + t$, where t is a threshold or darker than $I_p - t$. The radious of the circle can be of

any size and the algorithm check only the pixels in the location of 1,5,9 and 13. If 3 of the absolute differences between I_p and the pixels previously mentioned are more than $I_p + t$ or less than $I_p - t$, then the candidate is a corner. Otherwise is excluded.

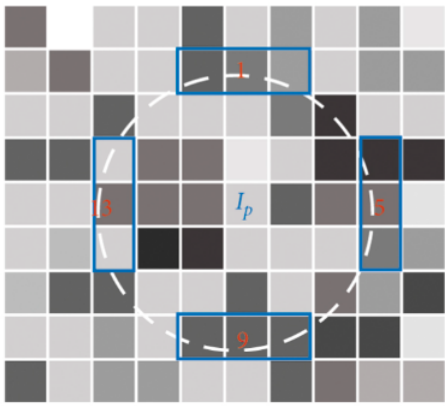

Figure 2.21: FAST detecting

SURF detector

The second-order Hessian matrix is calculated using Speeded-Up Robust Features (SURF) using integral pictures. Instead of iteratively down-sampling the filtered pictures, the scale space is created by scaling up the filter size. In the 3x3x3 neighborhood of the following scale space layer, the keypoint is established, and the maximum is kept for subsequent non maxima. Every 60 degrees, the cumulative operation of the wavelet response values is carried out in a region of a circle with a radius of $6 * s$ around the keypoint. The direction of a keypoint is determined by the subarea's maximum total of responses. The description vector must be built as the final step. To create the description, a local square region with the same orientation as the keypoint and a size of $20 * s$ where s is the scale of this keypoint is divided into four 4 x 4 sub-square regions. $\sum dx$, $\sum |dx|$, $\sum dy$, and $\sum |dy|$ filtered by Gaussian are summed separately in each sub-region.

KAZE features

Alcantarilla et al. have presented an enhanced version of SIFT called KAZE. Nonlinear scale space, which is formed by the nonlinear diffusion filter, aims to prevent blurring edges and details lost in linear scale space, which is created by the Gaussian filter. The non linear diffusion filter is described by the next equation:

$$\frac{\partial L}{\partial t} = div(c(x, y, t)\Delta L) \tag{2.15}$$

In the previous expression t is the scale factor and div is the conduction function corresponding to the SIFT-used Gaussian filter. In theory, KAZE can find more keypoints than SIFR. Fast explicit diffusion (FED), a sophisticated numerical algorithm integrated in a pyramidal framework, is used in KAZE's improved vision to significantly speed up feature recognition in the nonlinear scale spaces.

2.5 Epipolar geometry

Since when there is use a camera to capture images or videos, several information is lost due to the 3D-2D conversion, such as the depth of an image, or which is the distance between each point in the image. A solution would be use not only one camera and use more than one camera. If for instance there are used two cameras, the way in which the system works would be similar as how the human eyes work. If it is considered the next figure 2.22 In the figure the two points o and o' represent

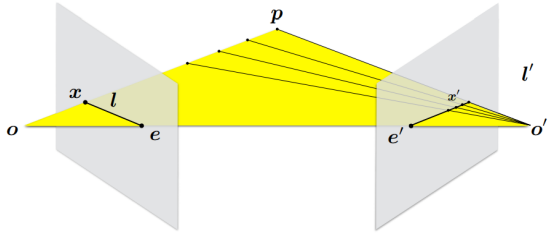

Figure 2.22: Stereo cameras representation

the two cameras and p is the point that the both cameras are seeing. As is possible to see if it is only considered one of the two cameras, it is not possible two find the 3D points corresponding to the p point in the image, this is due to the fact that each point along the line op projects the same point on the image plane. If there is considered that the different points on the line op projects to different points in the right plane. Having this information is possible to triangulate the correct 3D points.

The projection of the points in the line op form a line on the right plane, that is call the epiline corresponding to the point p. It implies that there is need to search along this epiline to locate point x on the right image. It should be along this line, it is important to keep in mind that it is not needed to look through the entire image to find the matching point; just look along the epiline. As a result, it offers improved performance and precision. The term for this is epipolar constraint. In the other image, all points will have their associated epilines. The name for plane poo' is Epipolar plane.

From the setup shown in the figure 2.22, it is possible to see that the projection of the right camera o' is seen on the left image in the point e. This point is call

epipole. The epipole is the point where the line connecting the picture planes and camera centers intersects. The epipole of the left camera is e' in a similar manner. In some instances, the epipole won't be seen in the image since it may be outside of it. Its epipole is the intersection of all epilines. We can locate numerous epilines and their points of contact in order to determine the epipole's location.
We therefore concentrate on locating epipolar lines and epipoles in this session. However, in order to locate them, we need two additional components: Fundamental Matrix (F) and Essential Matrix (E). The translation and rotation data, which specify where the second camera is in relation to the first in global coordinates, are contained in the Essential Matrix [42]. In essence the Fundamental Matrix

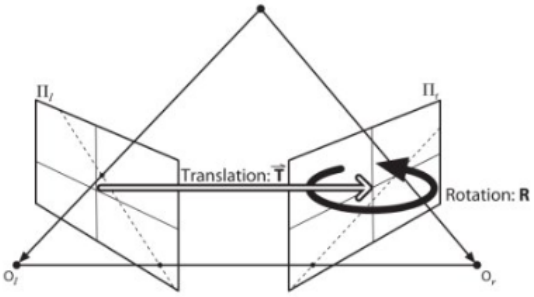

Figure 2.23: Translation and rotation representation

and the Essential Matrix contains the same information, but the Fundamental Matrix has in addition the information about the intrinsics of both cameras, which is an important information because it allows to relate he two cameras in pixel coordinates. Fundamental Matrix F, to put it simply, converts a point in one image to a line (epiline) in the other. This is calculated using points from both photos that match. Finding the basic matrix requires a minimum of 8 such points. To obtain a more reliable result, more points are preferred and RANSAC is used [43].

2.5.1 The Fundamental Matrix and The Essential Matrix

It must be computed the epipolar geometry in order to extract depth information from a pair of images. This geometric restriction is represented algebraically in the calibrated environment using a structure known as the Essential Matrix. It is recorded in the Fundamental Matrix in the uncalibrated environment. As in show in the figure 2.23, if there are considered the two views, the camera coordinate systems are related by translation T and rotation R, this can be reprensented in the next expression

$$x' = Rx + T \tag{2.16}$$

Then if there is take the product vector with T, and after that multiply by the scalar product with x', there is obtained:

$$x'.(T \wedge Rx) = 0 \tag{2.17}$$

The previous equations express the fact that the vectors ox, $o'x'$ and oo' are coplanar, which can also be written in the next way:

$$x'^{T} Ex = 0 \tag{2.18}$$

Where E is equal to:

$$E = \begin{bmatrix} 0 & -t_x & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} .R \tag{2.19}$$

Where E is the essential matrix and $T = (t_x, t_y, t_z)^T$, the equation 2.18 is the algebraic representation of epipolar geometry for known calibration and matching image points reported in the camera coordinate system are related by the fundamental matrix.
The equation 2.17 is homogeneous with respect to T. This represents the reality that size is uncertain, and without some additional information, such as knowing the separation in space between two places, we are unable to identify the exact scale of the picture. So for this, E is a 3x3 matrix that only depends on five parameters. In the case that $T = 0$, is a straightforward answer, but one that cannot be used to determine the depth of points in space, hence it is typically disregarded [44].
The essential and the fundamental matrices have the following properties:

- The essential matrix only includes the extrinsic parameters of the camera, while the fundamental matrix includes both the intrinsic and extrinsic characteristics.
- The essential matrix has only five degrees of freedom an its size is 3x3. In order to estimate it with corresponding image points, the intrinsic parameters of both camera must be known
- The fundamental matrix maps epipoles to the origin of the corresponding image plane
- The Fundamental matrix has seven degrees of freedom. Only their ratio out of the 9 matrix elements—which leaves 8 degrees of freedom—is significant. Additionally, the constraint that $detF = 0$ leaves seven degrees pf freedom.

2.6 Triangulation in computer vision

Triangulation in computer vision is the process of identifying a point in 3D space from its projections onto two or more images. The parameters of the camera

projection function from 3D to 2D for the involved cameras, in the simplest case represented by the camera matrices, must be known in order to solve this problem. Reconstruction or intersection are other names for triangulation.
Considering two given points (x_i, x_1') that are located in one image each, that are taken by the two cameras left and right in the system, as is shown in the following figure:

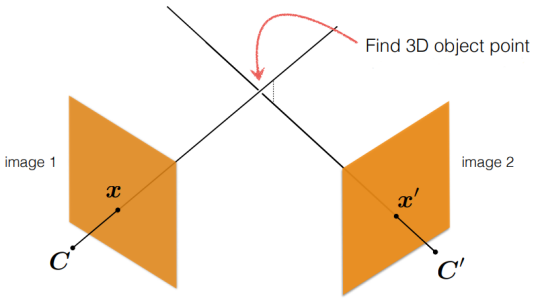

Figure 2.24: Graphic of the problem to solve by triangulation

In order to estimate the 3D point from a set of noisy matched points (x_i, x_i') and camera matrices (P, P'), it is needed to computed the this 3D point X from two correspondences (x, x'), as is shown in the following expression

$$x' = P'X \;\; x = PX \tag{2.20}$$

This process ca be developed with homogeneous or in-homogeneous coordinates. In the case of the in-homogeneous the expression will consider in this way:

$$x = \alpha PX \tag{2.21}$$

Then:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \alpha \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.22}$$

In order to find a solution for unknowns in a similarity relation, it is applied the Direct Linear Transform to remove the scaling factor and convert the system into a one linear system and solve with SVD. Since the cross product of two vectors of same direction is zero, the scaling factor can be removed

$$x \times PX = 0 \tag{2.23}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \times \begin{bmatrix} p_1^T X \\ p_2^T X \\ p_3^T X \end{bmatrix} = \begin{bmatrix} yp_3^T X - yp_2^T X \\ p_1^T X - xp_3^T X \\ xp_2^T X - yp_1^T X \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.24}$$

Since the third line is a linear combination of the first and second line, to obtain 2D to 3D point correspondence there are going to be need two equations:

$$\begin{bmatrix} yp_3^T X - yp_2^T X \\ p_1^T X - xp_3^T X \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{2.25}$$

Now the system is linear so it can be created a system of linear equations. Considering the points (x, x') the system of equations will be:

$$\begin{bmatrix} yp_3^T - p_2^T \\ p_1^T - xp_3^T \\ y'p_3'^T - p_2'^T \\ p_1'^T - x'p_3'^T \end{bmatrix} X = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.26}$$

Summarizing the system to solve could be considered as the next expression:

$$AX = 0 \tag{2.27}$$

And applying the Total Least Square the solution will be the eigenvector that corresponds to the smallest eigenvalue of $A^T A$

Chapter 3

Dataset

Since the aim of this work is to find the pose of a camera in order to find the pose of a robot in an indoor environment while it is doing a route, there were used two main datasets considering for the task: KITTI 2015 that is one of the most used datasets for odometry estimation and a dataset constructed by images taken in an indoor environment with an OAK-D camera doing different paths. These datasets are described in the following sections:

3.1 KITTI 2015

Is a real-world computer vision benchmark using the autonomous driving platform Annieway. The system is composed of two high-resolution color and gray-scale video cameras and the ground truth for the datasets is provided by a Velodyne laser scanner and a GPS localization system. The purpose was to create datasets for tasks such as optical flow, visual odometry, 3D object detection, and 3D tracking. The videos were recorded by driving in the city of Karlsruhe in rural areas or in highways and up to 15 cars and 30 pedestrians can be seen in the images. For this work, it was used the dataset for the optical flow was to evaluate the behavior of the CNN selected and how it influence the odometry estimation [45]

Figure 3.1: (a) Montage for the KITTI 2015 dataset (b) Example image in the KITTI 2015 dataset

3.2 Flying chairs

Flying Chairs is a synthetic dataset with optical flow ground truth. 22872 picture pairings and related flow fields make up this dataset. Images depict dynamic 3D chair models on arbitrary backgrounds taken from Flickr. Purely planar movements may be seen in both the background and the seats. This data set had been used in several works to train Convolutional Neural Networks to estimate the optical flow, for instance, was used in the FLOWNET and FLOWNET 2.0. [46]

Figure 3.2: Flying chairsl Image dataset example

3.3 Sintel

Sintel is dataset that has naturalistic video sequences. It is intended to promote study into non-rigid motion, long-range motion, motion blur, and multi-frame analysis. This dataset is contained by Flow fields, motion boundaries, mismatched

regions, and image sequences. The sequences many degrees of complexity in the rendering. The Sintel was a collaboration between Ton Roosendaal and the Blender Foundation that created the animated short film. This dataset had been used also several times to train new neural networks to estimate the optical flow such as the FlowNet 2.0 [47]

Figure 3.3: Sintel Image dataset example

3.4 Custom Dataset

3.4.1 Camera OAK-D

The OAK-D is a camera that implements stereo and RGB vision with the three cameras that have on board. The images can be piped directly into OAK SoM for depth and AI processing, which allows running advanced neural networks meanwhile provides depth information from the two stereo cameras. The camera is powered via USB3 5Gbps speeds for streaming video or data from the device, the whole system has a total power consumption that is around 900ma. For the purpose of this project, it was obtained from the camera the stereo images from the left and right camera and also the depth map. The following figure shows an image of the camera used: The maximum depth perception distance depends on the accuracy of the depth perception and can be calculated with the following equation:

$$Dm = (baseline/2) * tan((90 - HFOV/HPixels) * PI/180) \tag{3.1}$$

The specifications of the camera are shown in the next table:

3.4.2 DepthAI platform

In order to be able to acquire the information needed from the camera OAK-D it was used the DepthAI platform that is a Spatial AI platform, it is built around

Figure 3.4: OAK-D camera

Camera	Baseline [cm]	Focal Length [pixels]	Max. depth distance [m]	Lens size [inch]	Resolution
OAK-D	7.5	882.5	38.25	1/4	1MP (1280x800)

Table 3.1: Specifications of the camera OAK-D

Movidius VPU, which means in essence that it is used to allow robots and computers to identify and process what objects and features are and also where they are. This type of platform combines 6 features mainly: Artificial Intelligence, Computer Vision, Depth perception, high resolution, support of different sensor configurations, and Embedded and low-power solution. This platform was downloaded and used on a computer with Ubuntu 18.0.

3.4.3 Camera Calibration

Before use the camera is necessary to do and stereo calibration. The camera was placed in the baseline in which it was going to be for take the images for this work, and it was used the DepthAI platform and also the charuco board as calibration image that is shown as follows:

This image was placed onto a flat surface, been secured that it did not have wrinkles or waves. After that is was started the calibration script and the input parameters were:

- The measure of the square size in centimeters of the charuco board
- A flag that specifies that it was used the charuco board as calibration image.
- Device of board type, that in this case since the work was done with the OAK-D, was BW1098OBC

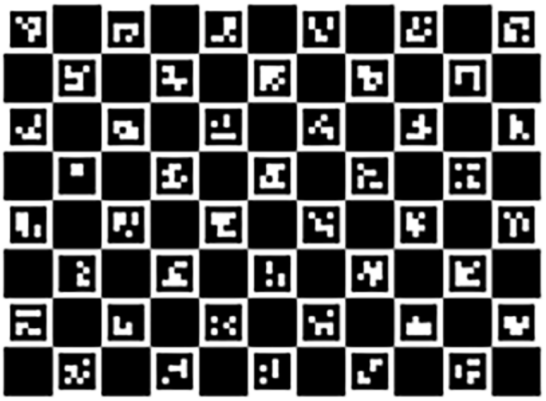

Figure 3.5: Calibration image

When the script was run it ask to put the calibration image in different positions and take pictures of it. Basically, in the calibration what happens is that to determinate the orientation and distance of the charuco board, it uses the intersection. For this reason, the accuracy will be get by a display of the provided board image on a flat plane. The results of the calibration were:

```
[[ 8.08707774e+02  0.00000000e+00  6.30945648e+02 -7.26598825e+03]
 [ 0.00000000e+00  8.08707735e+02  3.89927151e+02  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00  0.00000000e+00]]
```

Figure 3.6: Calibration matrix

3.5 Images acquisition

For the acquisition of the images, there were used the left and right cameras of the OAK-D to obtain the stereo images and the depth map from these. The camera was put on a robot developed in the Pic4Ser and there were three different paths to do. The paths were inside the installations of the Pic4Ser, the first one was in the corridor and the robot did a straight route, the second was also in the corridor but the robot did a turn to get into the parallel corridor, and finally the third was inside one of the thesis workrooms. The video from the two cameras and the depth map were taken using the DepthAI platform run in Ubuntu 18.0. After that, the videos were split into frames using Python 3.8.0 also in Ubuntu 18.0. The dataset was composed basically of three sub-datasets: one for the left images, another for the right images, and the last for the depth map, with the aim of having an easier handling of the data in the algorithm.

Figure 3.7: OAK-D camera on the robot

Chapter 4

RAFT

Recurrent All-Pairs Field transforms (RAFT) is a deep neural network architecture to calculate the Optical Flow. It is composed of three main components: a feature encoder that extracts a feature vector for each pixel, a correlation layer that produces a 4D correlation volume for all pairs of pixels with subsequent pooling to produce a lower resolution volume, and a current GRU-based update operator that retrieves values from the correlation volume and iterative updates a flow field initialized at zero.

4.0.1 Network Structure

In the network the function of the feature encoder is to extract features per pixel, the correlation layer looks for the visual similarity between the pixels of the image, and finally, the update operator behaves as an iterative optimization algorithm. The RAFT has three main features that differentiate it from traditional methods. The first is that the fixed flow is maintained and updated always in high resolution, the advantage is that working only in a single high-resolution flow field, avoids missing small fast-moving objects and recovers easily from the course the features and motion priors are learned by the feature encoder and update operator respectively. The second is that the update operator is lightweight and recurrent, it has 2.7M parameters and can be applied more than 100 times during inference without divergence. The third is the design of the update operator, which consists of a Gated Recurrent Unit (GRU) able to search on 4D multi-scale correlation volumes. The implementation of the RAFT is composed of these steps: Feature Extraction, Computing Visual Similarity, and Iterative Objects which are going to be described in the following sections.

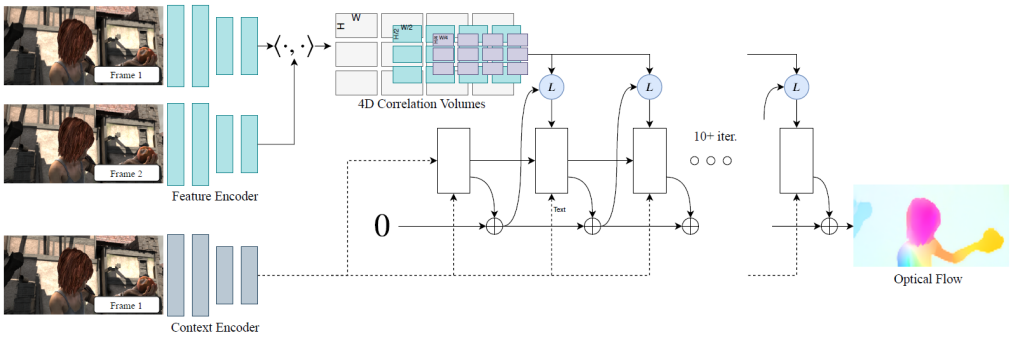

Figure 4.1: RAFT Architecture

4.0.2 Feature Extraction

The inputs of the neural network are two images I_1 and I_2. First, a convolutional network is used to extract the features, then a feature encoder is applied to these images to map them in dense feature maps at a lower resolution. The feature encoder is made with pairs of 1/2, 1/4, and 1/8 resolution for a total of 6 residual blocks. For this step, the authors also used a context network that extracts features only from I_1. The architecture of this network is the same as the one of the feature extraction networks. The implementation of the Feature and Context Network is the first stage and it is done only once.

4.0.3 Computing Visual Similarity

The visual similarity is computed with the construction of a full correlation volume between I_1 and I_2. The correlation volume is built by taking the dot product between the pairs of feature vectors. With this definition, the correlation volume can be defined as a single matrix multiplication, as is represented in the next equation:

$$C_{ijkl} = \sum g_\theta(I_1)_{ijh} \cdot g_\theta(I_2)_{klh} \tag{4.1}$$

Where $g_\theta(I_1)$ and $g_\theta(I_2)$ are the image features. The Correlation Pyramid is a four-layer pyramid constructed by pooling the last two dimensions of the correlation volume. The volumes have information about large and small displacements.
There is defined as a Lookup operator that generates a feature map by indexing from the correlation pyramid.

4.0.4 Iterative Updates

The update operator estimates the flow sequence from an initial starting point $f_0 = 0$. In each iteration, produce and update direction defined as Δf that is applied to the current estimate $f_{k+1} = \Delta f + f_{k+1}$. The update operator architecture was constructed to imitated the behaviour of an optimization algorithm so, it is trained to make its sequence converges to a fixed point.
The flow estimation starts with initialization of the flow field equal to 0 everywhere. After that, the current flow is estimate, which is used to retrieve correlation features from the correlation pyramid, these features are processed by 2 convolutional layers and also there are applied 2 more convolutional layers to generate the flow features. Then, the input feature map is taken as the concatenation of the correlation, flow, and context features. The flow update is predicted passing the output of the GRU through 2 convolutional layers, and it is 1/8 resolution of the input image. For this reason, there is need of an up-sampling stage, where are used 2 convolutional layer to obtain $H/8xW/8x(8x8x9)$ mask. The mask is used to perform softmax over the weights of 9 neighbors, finally, the high resolution flow field is found by taking a weighted combination over the neighborhood, after that it is done permutation a reshaping.

4.0.5 Implementation details

To implement the Convolutional Neural Network was used the open source integrated development environment (IDE) Jupyter with Python 3.8 and CUDA 10.4 in the Jetson Xavier AGX.

PTLFlow framework

PyTorch Lighting Optical Flow (PTLFlow) is a platform in development built on PyTorch for training and testing deep optical flow models. This platform allows to predict the Optical Flow with a pretrained model. Considering the approach of this work the datasets selected were the both described in the previous section: The Kitti 2015 and a custom dataset. Since KITTI 2015 is wide used in different works, there exist a pretrained model with it. In the case of the custom dataset, since it does not include the ground truth of the optical flow that corresponds to each image, it was used inference to calculate the optical flow in both cases. For the first dataset was used the checkpoint already uploaded in the platform for the CNN of interest. For the custom dataset were done several test to find the model with the best results. [33]

Data prepossessing

There were taken three videos with the OAK-D camera of different indoor paths. For each path, there were videos of the left and right monocameras. Following the methodology of the work, the optical flow was estimated only for the left images, to do this was necessary first separate the video in frames, which gives the results of several images in grayscale with a size of 1024x623. The optical flow was extracted for each frame to obtain information on how the camera was moving, for that reason the output was saved in a .flo file.

4.0.6 Experiments

The RAFT was used with the official checkpoints uploaded in the PLTflow platform. In the image 4.2 is shown the output of the optical flow estimated between two frames for the KITTI dataset. Since the KITTI dataset is one of the most used datasets for Visual Odometry purposes, it was used at the beginning to evaluate how well was the behavior of the inference with the RAFT and also to see the differences between the models available in the framework. However, it was noticed that the KITTI is a dataset built in an outdoor environment all the time and presents different conditions in comparison with the ones that can be found in an indoor environment. For this reason, the experiments were started in the dataset created for this work, made by stereo images from an indoor environment. There were trying the models available for inference in the PTLFlow framework, expecting that the model from the KITTI had a better behavior since it is used for Visual Odometry projects, but the better behavior was present by the model of the Sintel dataset, so this was the one used to develop this work.

Figure 4.2: RAFT Architecture

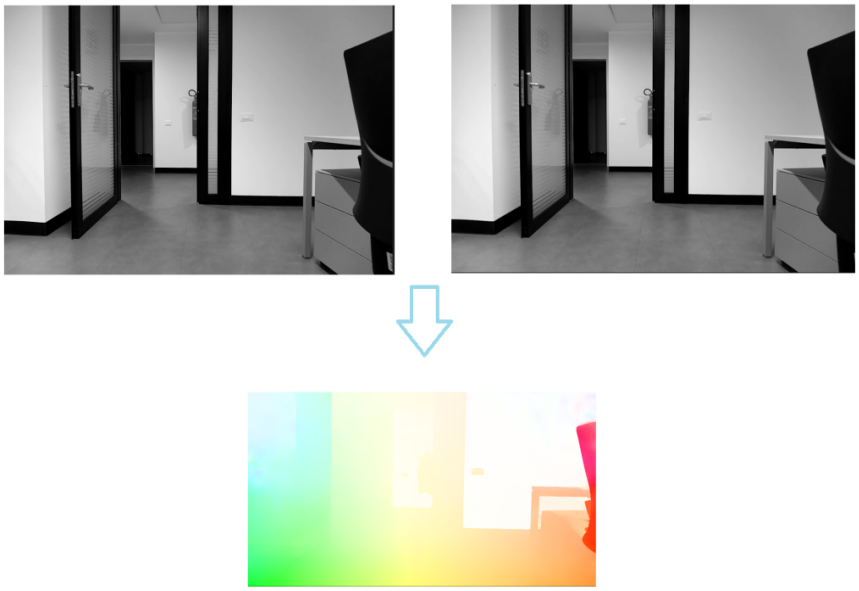

Figure 4.3: RAFT Architecture

Chapter 5

Key-points feature extraction

An important stage of this work to find the pose of the camera is to find the key points in the left and right images of each frame. Since the images are full of information that can be people, windows, or any object, in general, is necessary to find the characteristic points that can be matched along the images. It had to be taken into consideration that this work is limited to indoor environments, so is possible to not find as many objects to identify as in outdoor environments or that this task is more difficult because the illumination of the place is not the best. For these reasons was required to find a Key-point feature detector able to find the corners even for spaces with reduced luminosity and also faster since this is one of the stages of the whole process to calculate the pose of the camera. Considering this the point feature extractor [1], [29]:

5.0.1 FAST Algorithm for Corner Detection

Features from Accelerated Segment Test (FAST) came out as a solution for feature detection tasks in real time. What this algorithm does is:

1. Choose a pixel in the image p, that is to be identified as an interesting point or not. The intensity of this pixel is defined as I_p

2. Select a threshold value t

3. Consider a circle of 16 pixels around the pixel under test

4. The pixel p is a corner if in the image are around it, 12 contiguous pixels in the circle. These pixels should be brighter than $I_p + t$ or darker than $I_p - t$

5. A test check the pixels 1, 9, 5, and 13. If at least three of these are brighter than $I_p + t$ or darker than $I_p - t$, then p is a corner. If not, it is not considered as a corner

How the pixel looks for the algorithm can be seen in the next figure:

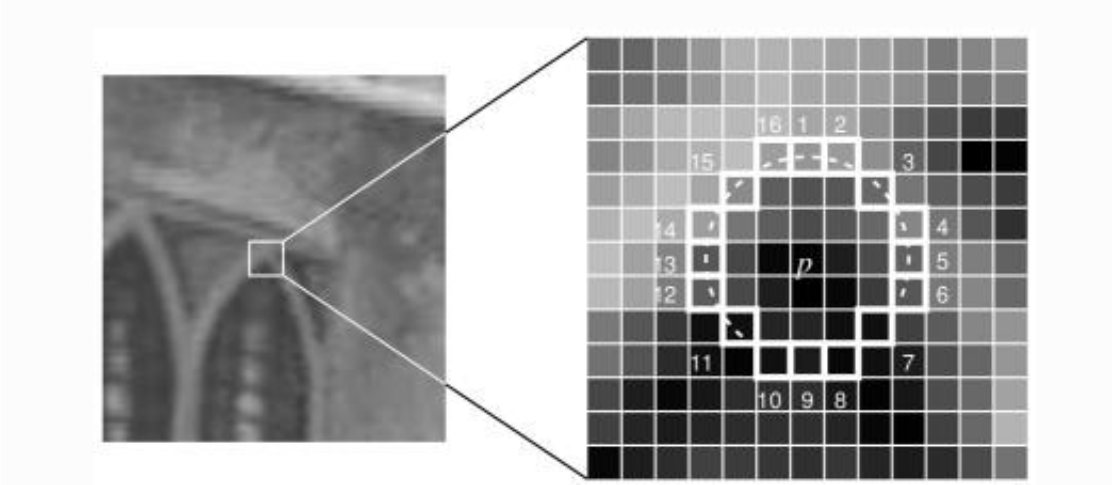

Figure 5.1: FAST Algorithm for corner detection

5.1 Implementation

For the implementation of this algorithm was used OpenCV 4.6.0 and Python 3.8. OpenCV is a free library of computer vision and machine learning. This library has a number bigger than 2500 optimized algorithms and supports C++, Python, Java, and Matlab.
The inputs of the algorithm were the left images of two consecutive frames I_i and I_{i+1}. The Key feature points were calculated for the image of the frame i. The steps followed in the algorithm were:

1. Initiate FAST object with default values.
2. Find the points in the images
3. Drawn the points in the image

In the following image can be shown an example of the results obtained from the FAST algorithm in one of the left images of the recording:

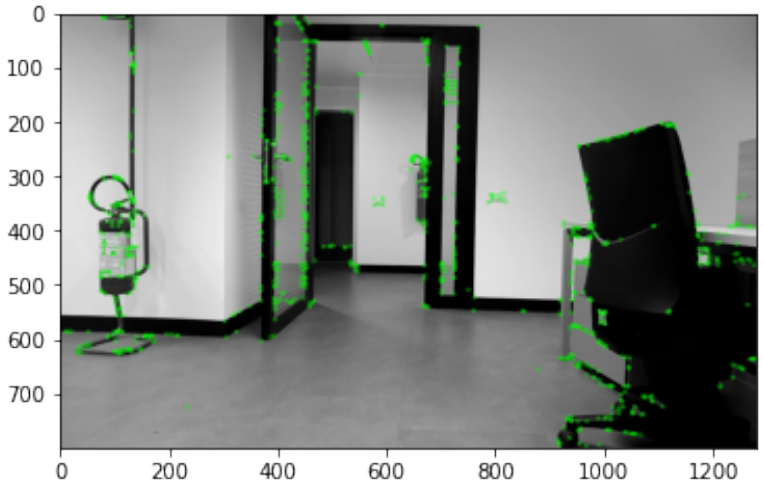

Figure 5.2: Key feature points in the left image I_i

In the second step to finding the points in the image, first, the images were divided into "Tiles" which are windows with a more detailed look of the part of an image without reducing the resolution of it. This is done with the intention of missing as less items as possible and also an emphasis on the areas near the border of the image. Considering this, if the tile is smaller more points are going to detect and if the tile is bigger key feature points are going to detect, an example is shown in the next image:

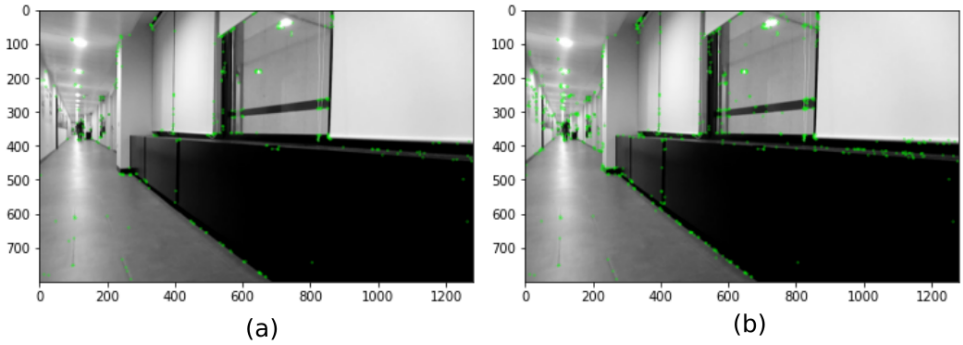

Figure 5.3: (a) Left Image with Tiles of (100,120) (b) Left Image with Tiles of (40,60)

5.2 Key feature points tracking

To calculate the position of the key feature points found in the Left images of the frame i in the frame $i+1$ was used the optical flow obtained from the Convolutional Neural Network. As is described in the previous section the optical flow contains information about how objects are moving over time, so it contains information on the displacement of each pixel between two frames. In this work was extracted a .flow file that contains specifically the vectors (u,v) that are the directions in which the pixels are moving. To extract this information was used the python library flowpy.

5.2.1 Flowpy library

Flowpy is a python package to work with optical flows, and it was used to read and extract the information from the .flo files. An advantage of this library is that is possible to plot arrows on the image for easier visualization of the flow and also gives the values of the flow for each pixel. The next image show the optical flow obtained from two images of the custom dataset with the arrows and the color wheel:

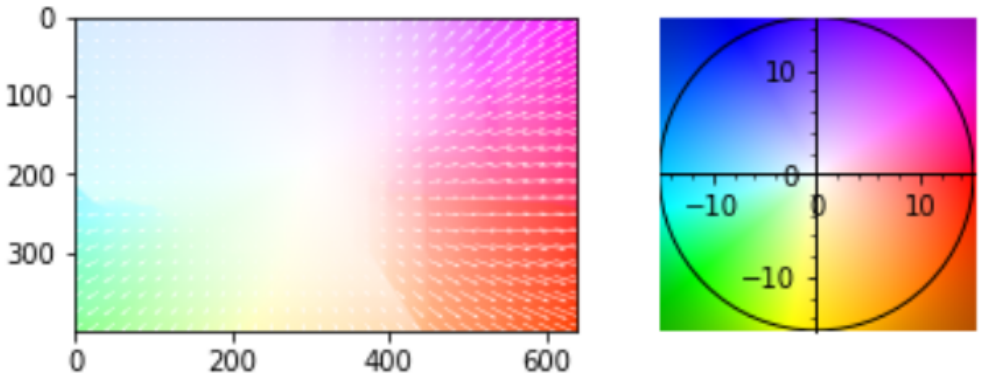

Figure 5.4: Image obtained using Flowpy

5.2.2 Algorithm Implementation

In the implementation of the algorithm was used Google Colab with OpenCV 4.6.0 and Python 3.8. Since the computational cost and energy of this process is not too high it was not necessary to use a GPU. For this purpose was used the python package flowpy, described in the previous section, to extract the information that was in the .flo file obtained from the CNN. Once the information on the

displacement of the pixels was available, it was looked at in this map the pixels of interest in the image, which in this case are the ones identified by the FAST algorithm in the left image of the frame i.
Each feature point has its own coordinates on each image. Since the image is a 3D matrix with size (w,h,3), the position of the point should be in the values (w,h) that were assumed as the coordinates (x,y) of the points.
Using the approach exposed by Bouguet J, Y in the work "Pyramidal Implementation of the Lucas Kanade Feature Tracker Description of the algorithm" the position of the points in the next image $i + 1$ is going to be the sum of the optical flow vector plus the position of each point. Consider these equations:

$$Final\ optical\ flow\ vector : d = g^0 + d^0 \tag{5.1}$$

$$Position\ of\ points\ on\ image\ I : v = u + d \tag{5.2}$$

Being u the point in the image, v is the corresponding location in the image of the frame $i + 1$, and d is the pixel displacement vector.

For the tracking point process, it is important to guarantee that the points are inside both images because as the camera is moving some objects or points seen in the previous image are not going to be anymore in the image of the following frame. The size of the images for this case was (375, 1242, 3), so in the code implementation, the tracking points were the ones that were inside these values. To do this, after the FAST algorithm calculate the key feature points in the image i and the points were tracking in the image of the frame $i + 1$ using the optical flow, the points selected in both images were those that have a position in x less than 1242 and in y less than 375.

5.2.3 Points tracking in right stereo images

Once the key feature points were obtained in the left images, these same points have to be found in the right stereo images in the same two continuous frames analyzed. This task was done using the disparity map.
From the OAK-D camera besides the stereo videos was also captured a video of the depth map from which was obtained the disparity, for this case was also used the library DepthAI.
The disparity map can be obtained from the distance between two corresponding points in the left and right images of a stereo pair. As is shown in the following image: x gets projected to XL in the left image and XR in the right image

Since the points XL and XR are known, the disparity map to the point X is equal to the magnitude of the vector between the coordinates of XL and XR. Each

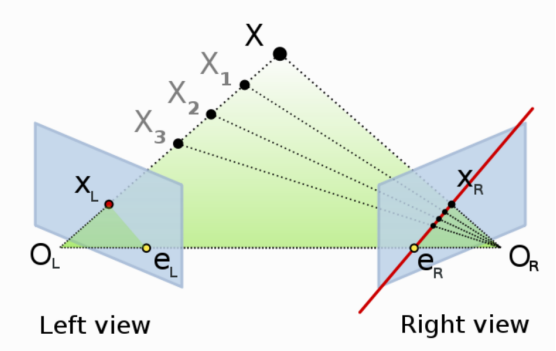

Figure 5.5: Disparity map calculation

pixel in the disparity map has a confidence value between 0 and 255. Where 0 is the maximum confidence that this pixel holds a valid value and 255 minimum confidence. The level of confidence means the chance of the value to be correct or incorrect.

For this work, an option was to apply a filter based on the confidence value of each pixel. For this, first, a threshold should be set, and the pixels that had a confidence value higher than this threshold get invalidated, which means that their values go to zero. For this case instead of used a threshold it was used a Median Filter with a Kernel 7x7

5.2.4 Depth Calculation

The depth map can be calculated from the disparity map. These two terms are inversely related, if the disparity increases the depth decrease exponentially and vice-versa. This concept means that if the value of the disparity is close to zero, then small changes in disparity generate the largest changes in the depth. To calculate the depth it can be considered this formula:

$$depth = \frac{focal\ length\ in\ pixels * baseline}{disparity\ in\ pixels} \tag{5.3}$$

The baseline is the distance between the two monocameras and the focal length can be obtained from the camera calibration. As can be seen in the following image the baseline for the OAK-d is 75mm

The focal length can be obtained theoretically with the next equation. Where HOFV is the horizontal dimensions of the measurement field in the measurement object plane, which can be determined with the horizontal visual angles of the lens

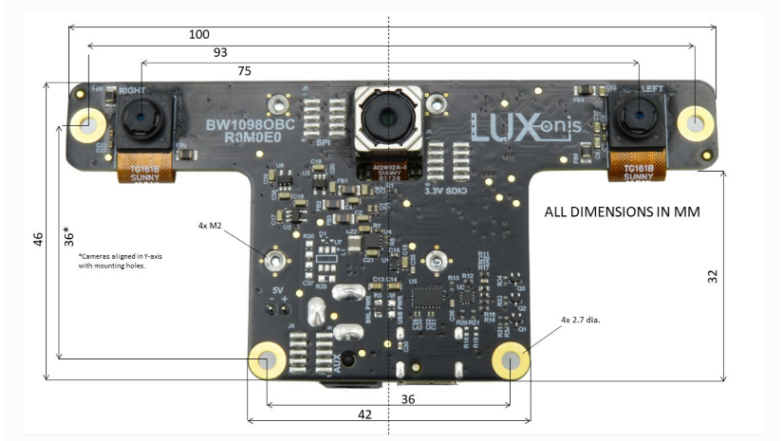

Figure 5.6: Baseline of the OAK-D

$$focal\ length\ in\ pixels = \frac{image\ width\ in\ pixels * 0.5}{tan(HFOV * 0.5 * \frac{PI}{180})} \tag{5.4}$$

In the documentation of the camera was found that the value of the focal length is 882.5 $pixels$. Taken as HFOV: 71.9 degrees and considering that the resolution of the monocameras is 800 pixels so, the image width in pixels is 1280.

For the purpose of this work also had to be taken into account the minimum depth perception distance of the device, because it is possible that in some cases the depth map looks weird for too close objects. To calculate this, it can use the previous equation for the depth using a baseline of 75mm, a focal length in pixels of 882.5 pixels, and the default maximum value for the disparity indicated in the documentation is 95, obtaining a minimum depth distance of 69.67cm

$$min\ distance = \frac{882.5 * 7.5cm}{95} = 69.67cm \tag{5.5}$$

To obtain the maximum depth distance it was used the next equation:

$$max\ distance = \frac{baseline}{2} * tan(\frac{90 - HFOV}{Hpixels}) * \frac{\Pi}{180} \tag{5.6}$$

For the OAK-D the values are:

$$max\ distance = \frac{7.5cm}{2} * tan(\frac{90 - 71.9}{1780}) * \frac{\Pi}{180} = 38.25m \tag{5.7}$$

This value is an approximation because the maximum distance possible depends mainly on the accuracy of the depth perception.

Depth perception accuracy

Since the depth map is calculated based in matching the features of two continuous images, the accuracy depends on theses parameters

- Texture objects and backgrounds
- Lighting
- Baseline and distance to the objects

The first parameter is taking into account specially because the backgrounds are objects too, so it could affect the accuracy of the depth calculation. This could not limit a lot in outdoors environments, but in indoors it does, so since this work is limited to an indoor environment the parameter was considered. To evaluate this, there were done three paths: the first two were in a more clean space (in the corridor), but the third was done in a space with more artifacts in it (Thesis room of the Pic4Ser). The Light in the space was also important, because if the illumination is low, the confidence of the disparity map is low. Finally, the baseline or the distance to objects were considered, because if the baseline is low it allows to detect the depth at a closer distance when the object is visible in both frames, the problem is that this can reduce the accuracy for large distances due to the fact that less pixels are going to represent the object and the disparity is going to decrease faster to 0. For this reason, was important to adjust the baseline based on how far or close there had to be detected the objects. For this work, it was considered a fixed based baseline of 7.5 cm that was the one that had the camera from fabrication, and the minimum and maximum depth were considered to analyze the results.

5.2.5 Right points estimation

The estimation of the points in the right images was done taking into account the correspondence problem, which talks about determinate the pair of pixels in the stereo images that are projection of the same physical point in the space. To find the corresponding pixels of the left image in the right image it was used the Block Matching Algorithm, which is based in compare a window around the point in the image of reference with numerous blocks along the same horizontal line in the second image. In this case, the point in the right image is the one with the best match with the point of the left image, which means that has the minimum loss value. Considering all this the disparity is equal to:

$$d(x, y) = x - \hat{x} \tag{5.8}$$

Where $\hat{x}$ is the point in the right image and x is the point in the left image, with this equation we can deduce that subtracting the disparity to the x value of

the pixel, it is possible to estimate the corresponding position of the pixel in the right image, as is shown:

$$\hat{x} = x - d(x, y) \tag{5.9}$$

The process is shown in the following image:

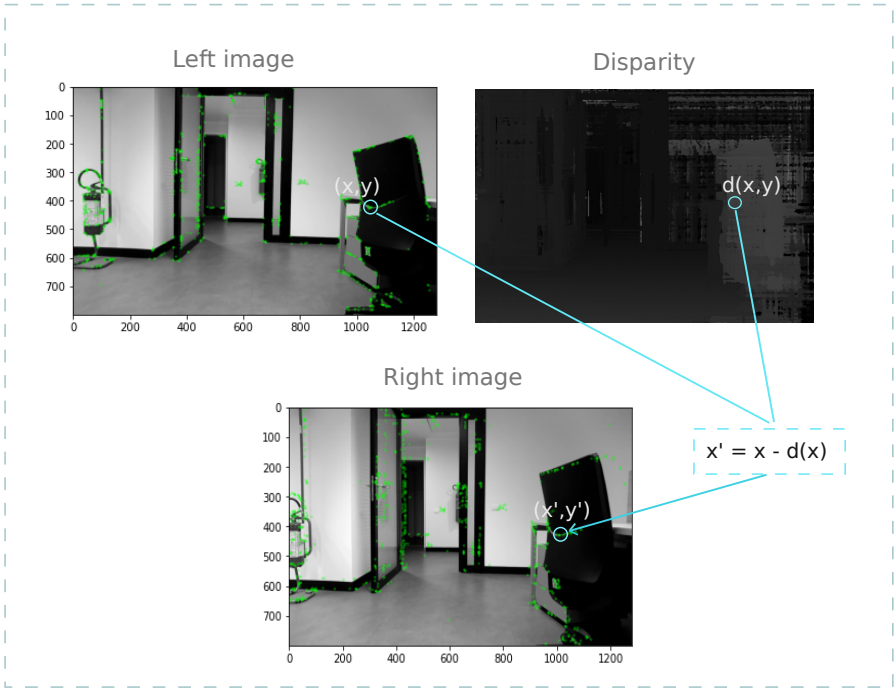

Figure 5.7: Right points calculation process

Chapter 6

Pose estimation

Once gotten the key feature points of the left and right images of the two consecutive frames, the next step was to obtain the 3D points in these two frames. After that, these values are going to be the inputs to calculate the pose of the camera. This process is composed by the following stages

6.1 Triangulation

To start the process to get the position of the camera it was important first to consider that the camera that was used was a Stereo Camera so, there were two cameras capturing the same scene. The concept used was the of the Triangulation in computer vision, which means that: First, it must be assumed that both cameras were calibrated, that was done and it is explained The Section 3.4.3. From this calibration was obtained the matrix P and P' for the two cameras that observed one point each.
P and P' contains the information about the position of the camera in the space, so if the information about the points that the camera is looking is available, a virtual line could be drawn from the center of the first camera $\widetilde{O}$ through the point $\widetilde{x}$ that is considered as the 3D point on the lens of the first camera. The same procedure was followed for the second camera, so a line was drawn for the second camera also from the center of the camera $\widetilde{O}'$ through the 3D point $\widetilde{x}$ and the intersection of these two lines should be the 3D point in the real space that was denominated as $\widetilde{X}$.
For this purpose was used OpenCV 4.6.0 and Python 3.8 in the platform of Google Colab. In OpenCV was found a function withe the name of **cv.triangulatePoints()** that allows to reconstruct a group of points with triangulation. This function receives as parameters:

- The 2D points of the image as an array

- The projection matrices of the camera.

For the algorithm, the inputs were the projection matrices of the left and right cameras and the points estimated in right and left images as it was explained in the previous section. This process was done two times, the first time for the images in the first frame and the other for the images in the second frame. The output of the array is the 3D points in that frame, so in this step, there were obtained the points in the real space of the frames i AND $i+1$. After the 3D points were found could be started the process to calculate the position of the camera. For the purpose of this work, the position of the camera was found, using the transformation matrix.

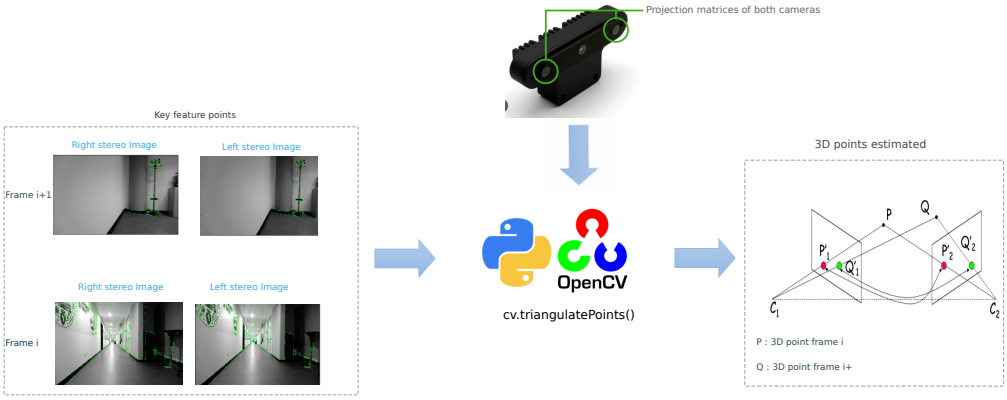

Figure 6.1: Right points calculation process

6.1.1 Transformation matrix estimation

For a rigid body basically, there are need six parameters to define the position and orientation. This means that three of these parameters are going to describe the position and three of these parameters are going to describe the rotation of the rigid body. To start to estimate the homogeneous transformation matrix and then the pose of the robot, there was important to consider the concept that both the body frame and the space frame are fixed frames that are fixed somewhere in space. The body frame is a fixed frame that is instantly attached to the moving body. So, the configuration of the body can be represented by the pair (R,p), where R is the rotation matrix with a size of (3x3) that represents the orientation of the frame that is in the robot in relation to another frame of interest. If both the translation vector and rotation matrix are put in the same matrix, there is obtained a single 4x4 matrix, that is the homogeneous matrix T as is shown in the next equation:

$$T = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_1 \\ r_{21} & r_{22} & r_{23} & p_2 \\ r_{31} & r_{32} & r_{33} & p_3 \end{bmatrix} \tag{6.1}$$

In the equation 6.1 R is the rotation matrix and p is a column vector. Another useful concept is the Special Euclidean group, which is the group that contains rigid body motions. The advantage of taking this approach is that the robot configuration is condensed in a 4x4 matrix.

In the algorithm what was done was the following: First, there were selected 5 random points in the image of the first frame. Then, these same 5 points were tracked in the image of the next frame and since these points also led to specifics 3D points these points were also tracked in that array and were selected in the same position, for this reason, if for instance there is taken one of this selected points with all the values previously mentioned it should be possible to find the corresponding point in the right image of the right and left frame. So, for summarizing these steps from 5 points in the image i randomly selected were also selected these same 5 points in the image $i + 1$ and in the 3D points of the frame i and $i + 1$. To understand which points work better to estimate with fewer errors in the position of the robot there was implemented an optimization step with the least square method.

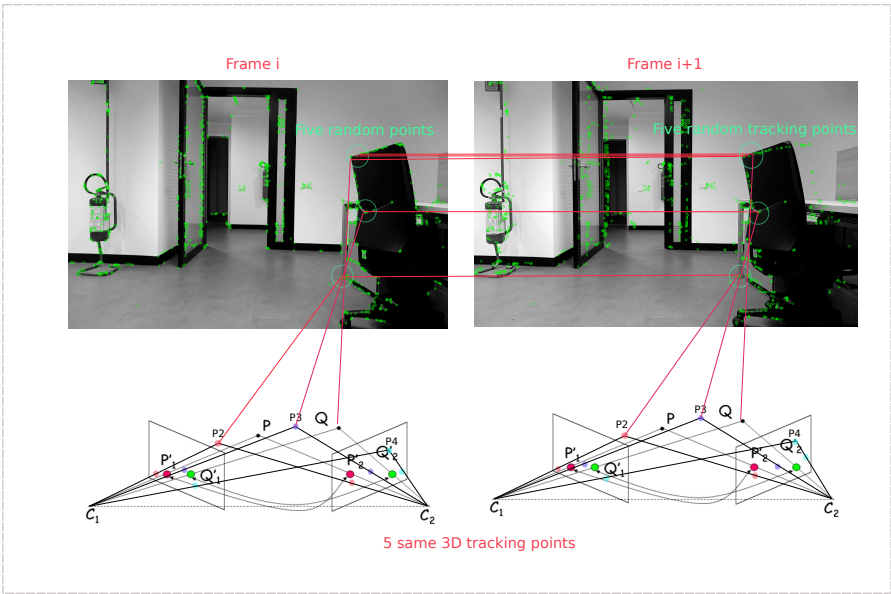

Figure 6.2: Five random points selection and trancking

Least-Square method

This optimization step was done with the version of Scipy 1.7.3 in Python 3.8 on the platform google colab. Scipy is built on the NumPy module of Python, SciPy is a collection of mathematical algorithms and useful functions. By giving the user advanced commands and classes for data manipulation and visualization, it significantly increases the power of an interactive Python session. SciPy transforms an interactive Python session into a system prototyping and data processing environment that competes with programs like MATLAB, IDL, Octave, R-Lab, and SciLab.
The least-square optimization was implemented with the function of Scipy least-squares() which basically finds the local minimum of the cost function nominated by the variable $F(x)$ and represented by the next equation:

$$F(x) = 0.5 * sum(\rho(f_i(x) * *2), i = 0, ..., m - 1 \tag{6.2}$$

In the equation 6.2 it is used the loss function $\rho(s)$ is to minimize the impact of the outliers in the function. The parameters that were the input of this function were:

1. Vector of residuals.
2. Initial vector of guess on independent variables
3. The algorithm to perform the minimization that in this case was Levenberg-Marquardt algorithm which does not support sparse and bound Jacobians and is one of the methods that is typically most effective for minor, unrestricted issues.
4. Number of maximum function evaluations before the termination. For Levenberg-Marquardt is 100*n, and n in this case was chosen as 200.
5. The additional arguments passed to the function that computed the vector residuals.

For the residual vectors in this work was implemented a function that uses as parameters: the transformation matrix between two frames so, there were needed the first three elements to describe the rotation, and then the other three elements to describe the translation, so it was a NumPy array with the shape of 6. The other elements are the feature points of the left and right images and the 3D points seen from the images of the two consecutive frames. The output of this function was a NumPy array with size $2 * n_{points} * 2$ where the n_{points} were the number of points selected.
The first thing to do was extracted the rotation vector, with the Rodrigues method.

For this was used the function of OpenCV **cv.Rodrigues()** that converts a rotation matrix into a rotation vector or a rotation vector into a rotation matrix. In this case, the input was the rotation vector, and the output was the rotation matrix, then the translation vector was extracted. This function converts a rotation matrix using these equations:

$$\theta < norm(r) \tag{6.3}$$

$$r < -r/\theta \tag{6.4}$$

$$R = cos(\theta) * I + (1 - cos(\theta)) * r * r^T + sin(\theta) * A \tag{6.5}$$

$$A = \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix} \tag{6.6}$$

The next step was to construct the transformation matrix, having the translation vector and the rotation matrix.
The next step was estimating the projection matrix for the images of the frames i and $i + 1$.

Projection Matrix

The projection matrix P is a square matrix with size nxn that gives the vector space projection from R^n to a subspace V. The columns of P are the standard basis vector projection and V is the image of P.

In this step, there were estimated twice the projection signs. The first one was to project the 3D points from frame $i + 1$ to frame i, and then the same procedure was repeated from frame $i - 1$ to i. Once the projections were obtained the residuals were calculated. For this task the function of **np.vstack()** from NumPY was employed. The function basically stack arrays vertically in order. This is comparable to reshaping 1-D arrays of shape (N,) to concatenation along the first axis (1,N). Creates arrays that have been split by vsplit. The most useful arrays for this method are those with up to three dimensions. For instance, consider pixel data having r/g/b channels, width (second axis), and height (first axis) (third axis). More versatile stacking and concatenation operations are provided by the functions concatenate, stack, and block.
The residuals vector contains the difference between the points in both frames predicted and extracted from the image.
Once the re-projection residuals are obtained, the next step is to calculate the error for the optimized transformation. Previously was selected a number of iterations for this process, but since the objective of this process is to minimize the error as much a is possible if there is a point in which the results are not going to improve, the algorithm must interrupt even if it is first that the number of iteration selected

is accomplished.

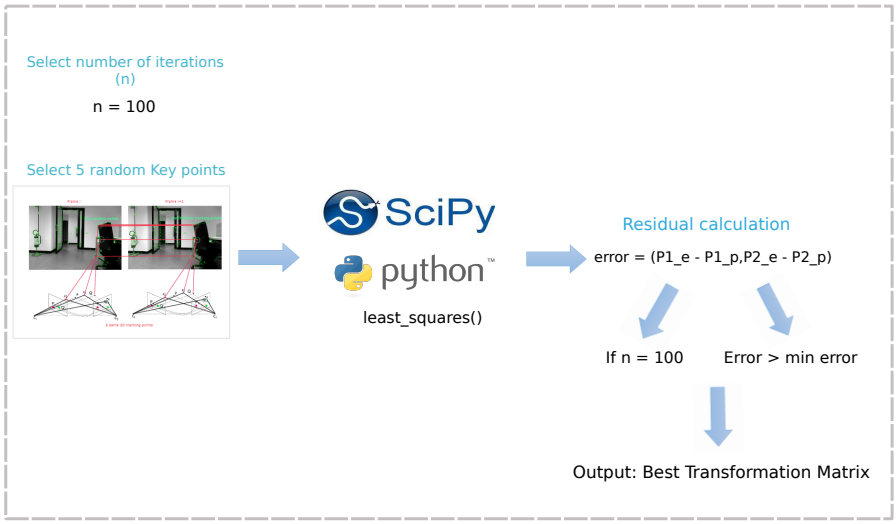

Figure 6.3: Transformation matrix calculation process

Once the best possible results are obtained there was extracted the rotation vector and the rotation matrix was, and with these two, the transformation matrix was constructed.
To finally obtain the position of the camera the algorithm was developed taking the following approach as reference: First, if the frame was the first one the current pose was going to be an identity matrix of size $4x4$, then when the frame is from the second and so on:

1. The current pose is the multiplication of the past current pose by the transformation matrix

2. It is created a NumPy array with size $nx2$ that contains the position (x,y) of the robot which is extracted from the matrix of the current pose

After these steps, the values were saved in a vector to plot them and compare the results with the ground truth that in this case are the values acquired with ROS.

Chapter 7

Results and analysis

Once the results were obtained for the three paths, these were plotted and compared with the ones obtained from the odometry calculated with ROS2 that for this work was considered the ground truth.
Before comparing both plots was needed to put them on the same reference axis. To do this was assumed as the reference axis the one used for the odometry calculated via ROS2, so each array that contains the information of the pose of the camera along the time was multiplied by a rotation matrix in z. The angles used for each path are shown in the table 7.1

	Path 1	Path 2	Path 3
Angle of rotation (°)	80	80	10

Table 7.1: Angles used for the rotation matrix in Z for each path

The process followed to put the two plots of the paths in the same reference axis is shown in the image 7.1. As is possible to see, the multiplication of the positions by the rotation in the z-axis, had to be done in order to compare the two results and exploit the reasons for the difference between both.

As was explained in section 3.5 how the dataset was composed, there were done three routes, the first one was a straight route, the second was also in the corridor but the robot did a turn to get into the parallel corridor, and finally the third was inside one of the thesis workrooms. The expectations for the first route was kind of a straight line, for the second was something similar to a part of a rectangle and for the third, since the thesis room has mainly three big tables and the robot passed around them, it was expected to see something similar to three rectangles.

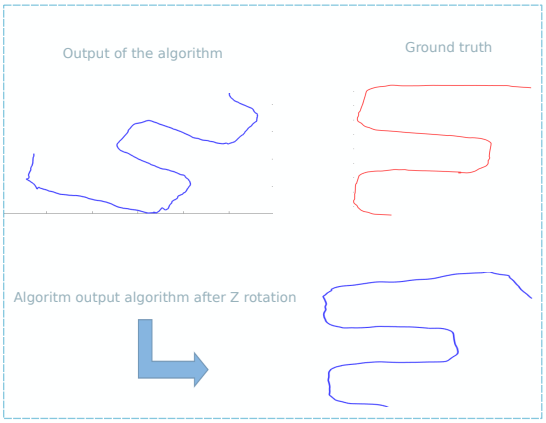

Figure 7.1: Plots of the path first of the rotation and then in the same axis

In the figures 7.2, 7.3, and 7.5 are shown the results obtained from the algorithm of the approach proposed in this work (blue line) and the plots of the position output measure by ROS (red line). As is possible to see in the figure 7.2 the output was similar to the one expected because it sort of represents a straight route. In general, any of the two plots is completely clean and straight, which could be due to the way that was moving the robot, because on the first hand it was conducted by a human, and during the route, some abrupt movements were done to avoid an obstacle in the floor, or even by error. An important feature to notice is that both lines start at the same point and cover almost the same points, which could be considered a sign that the algorithm is working correctly.

In the figure 7.3 are shown the plots for the second path. The shape of the plot agrees in general terms with the hypothesis previously exposed. In this case, can be seen also that the two plots start at the same point, and both cover almost the same points. The biggest difference is presented at the end of the trajectory, and this part represents the moment in which the robot turned to the right to get into the corridor that was parallel to the corridor where was first doing the route. This error in the measurement could be due to different factors, first is necessary to take into account that one of the steps more sensible to disturbances is the estimation of the key feature points in the right images which is also a step very important in the pose estimation. Since the Key feature points in the right images are estimated with the depth map, if the acquisition of it is altered could cause errors in the behavior of the algorithm. In this case, can be noticed from the images that the number of features is different from one corridor to another, and a really special point to consider is that when the robot turned to the right it found a wall made with a transparent material, that could considerably affect the estimation of the depth because since is transparent in some parts, is not possible

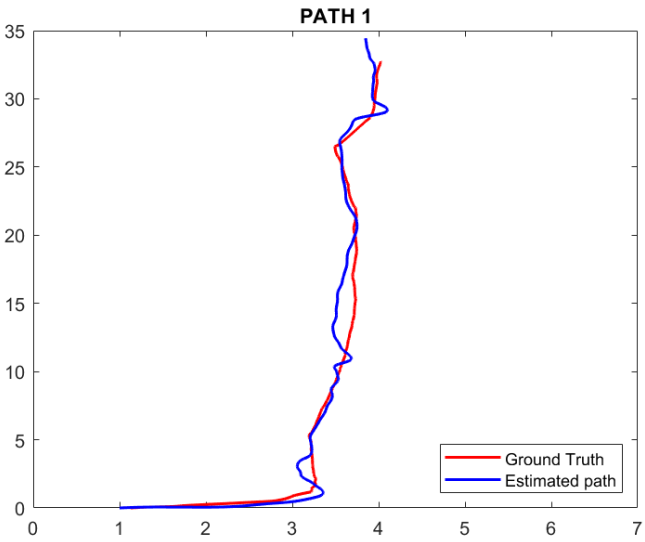

Figure 7.2: Plot of the results from the algorithm vs the ground truth for path 1

to recognize precisely the distance to the camera, so this could lead errors in the 3D points estimation, that propagates along the algorithm.

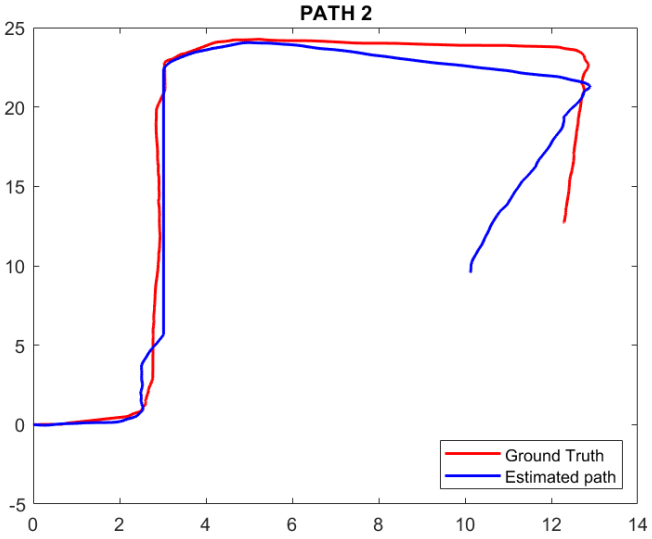

Figure 7.3: Plot of the results from the algorithm vs the ground truth for path 2

In the image 7.4 are shown an example of the image acquired by the stereo

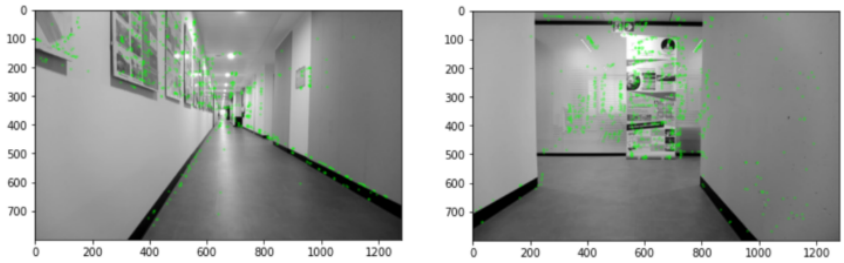

Figure 7.4: left image: First corridor - Right Image: View when the robot turned

camera when was in the first part of the trajectory (corridor) and when turned to the right. In these two views can be notices the different features of these two and how this influence the estimation of the position of the Key feature points. For instance, in the left image the points present a better estimation in comparison with the image on the right, which can be a reason of why in this part the algorithm shows better behavior.

In the figure 7.5 is shown the plot for the third path, as it was described before there were expected a similar shape of three rectangles in the image, which can be seen, also in this case both plots cover almost the same points but the biggest difference is in the second part of the trajectory. The reasons of that difference could be similar as the ones exposed for the second path. In this case, is important to consider that all the trajectory was done inside the same room, but something important is the first and the second part of the trajectory have different features.

In the images below 7.6 are shown the different spaces present in the third estimated path. In the image on the left, the objects in the images are closer to the camera, so the depth map estimated tended to give more correct information and for that reason, the Key Feature points were better estimated in these images. In the image on the right, there were objects really far from the camera that are still in the range in which the OAK-D captures depth information, but at this point, the depth information obtained has not the same quality. For this reason, the key feature points estimated in this part of the trajectory could be less precise and this could lead to errors in the 3D points estimation and finally in the pose estimation, which could explain the bigger variation in the last part of the trajectory.

To do a quantitative measurement of the method proposed behavior it was used the Mean-Square Error (MMSE) and the Root Mean Square Error (RMSE). The Mean-Square Error is the average squared difference between the values predicted

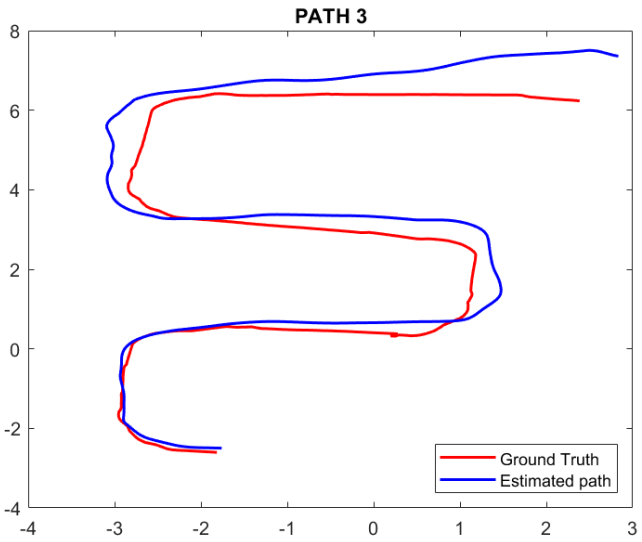

Figure 7.5: Plot of the results from the algorithm vs the ground truth for path 3

Figure 7.6: left image: First part - Right Image: Last part of the trajectory

and observed, it evaluates the average squared difference. The RMSE is the square root of the MMSE and is used to have the values of error in the same order of the data. The MMSE is equal to 0 when a model is error-free and when the model error rises its value rises. The mathematical formulation is:

$$MMSE = \frac{\sum(y_i - \hat{y}_i)^2}{n} \tag{7.1}$$

$$RMSE = \sqrt{\frac{\sum(y_i - \hat{y}_i)^2}{n}} \tag{7.2}$$

In the table 7.2 are shown the results of MMSE and RMSE for x and y from each path estimated. The results showed that it was a bit more accurate the estimation of the robot pose on the x-axis in comparison with the results obtained for the y-axis that differs more from the ground truth. Is important to take into account that the system is around many external perturbations that could cause these errors, for instance along the movement of the robot and because of its structure, there was presented a lot of vibration that can be easily seen in the videos taken by the camera, which affected the measurements taken by the camera including the right and left stereo images and the depth map. Another thing that could affect is that the camera was put on the top of the robot which makes it more sensible to the perturbances. The light is a factor that could influence the measurements because depending if there was or not enough illumination, this could avoid that the objects seen by the camera had been easily captured or not.

	MMSE - x	**MMSE - y**	**RMSE - x**	**RMSE - y**
Path 1	0.131	0.8891	0.3619	0.9429
Path 2	0.7745	1.8658	0.8801	1.3638
Path 3	0.2156	0.2562	0.4643	0.5062

Table 7.2: Results of the Mean Square Error

Chapter 8

Conclusion and Future Work

8.1 Conclusions

From this work is possible to conclude that the methodology proposed is valid to calculate the position of a robot. It can be concluded that Visual Odometry can be implemented using Deep learning to find the Optical Flow of images between two consecutive frames and that good results can be reached using the inference procedure. So, the miss of ground truth for the Optical flow dataset to train a neural network does not avoid to obtain acceptable results, because as is shown in this work models trained with other datasets give good results in the Optical Flow estimation task. Another important conclusion that can be done from this job is that for approaches similar to the one taken for this thesis work, which calculates the position of a camera/robot using images and specifically the depth map, this measurement has a high influence on the final results, so it is important to obtain the depth map estimation as precise as possible in order to avoid errors in the steps that depends on this values.
Regarding the hardware, It can be concluded that for methodologies that include the implementation of deep learning techniques there is a need for hardware with specifications that give it the computational capability enough to run the codes needed. For instance, the Jetson AGX Xavier proves that can be used for this kind of task with good performance, also that due to its size is a portable device so, if there is needed, can be part of the montage in the robot. For the camera, could be concluded that the OAK-D showed a good performance in this work, and also has numerous tools to calculate important information from the images, such as the depth map and other computer vision tasks. Thanks to its library DepthAI that can be installed in Windows or Ubuntu, and facilities in many ways the interaction with the camera and open many possibilities to use it in several types of works such as Visual Odometry, object recognition, etc.

For future work, there is a need to improve the estimation of the key feature points for the right images and to improve the depth estimated from the stereo images of each frame. Another task for future work could be to improve this methodology in scenarios in which the objects are far from the camera and also in environments where the walls are made of transparent material. Also, the montage of the camera can be improved in future work, to avoid as much as possible vibrations in the image acquisition.

Appendix A

Appendix

Bibliography

[1] Jean-Yves Bouguet et al. «Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm». In: *Intel corporation* 5.1-10 (2001), p. 4 (cit. on pp. 1, 54).

[2] Sherif AS Mohamed, Mohammad-Hashem Haghbayan, Mohammed Rabah, Jukka Heikkonen, Hannu Tenhunen, and Juha Plosila. «Towards dynamic monocular visual odometry based on an event camera and IMU sensor». In: *International Conference on Intelligent Transport Systems*. Springer. 2019, pp. 249–263 (cit. on pp. 1, 18).

[3] Huangying Zhan, Chamara Saroj Weerasekera, Jia-Wang Bian, Ravi Garg, and Ian Reid. «DF-VO: What Should Be Learnt for Visual Odometry?» In: *arXiv preprint arXiv:2103.00933* (2021) (cit. on p. 2).

[4] Xicheng Ban, Hongjian Wang, Tao Chen, Ying Wang, and Yao Xiao. «Monocular visual odometry based on depth and optical flow using deep learning». In: *IEEE Transactions on Instrumentation and Measurement* 70 (2020), pp. 1–19 (cit. on pp. 2, 16).

[5] Alec Graves, Steffen Lim, Thomas Fagan, et al. «Visual odometry using convolutional neural networks». In: *The Kennesaw Journal of Undergraduate Research* 5.3 (2017), p. 5 (cit. on p. 2).

[6] Peter Muller and Andreas Savakis. «Flowdometry: An optical flow and deep learning based approach to visual odometry». In: *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2017, pp. 624–631 (cit. on pp. 4, 18).

[7] David Van Hamme, Werner Goeman, Peter Veelaert, and Wilfried Philips. «Robust monocular visual odometry for road vehicles using uncertain perspective projection». In: *EURASIP Journal on Image and Video Processing* 2015.1 (2015), pp. 1–21 (cit. on p. 5).

[8] Mohammad OA Aqel, Mohammad H Marhaban, M Iqbal Saripan, and Napsiah Bt Ismail. «Review of visual odometry: types, approaches, challenges, and applications». In: *SpringerPlus* 5.1 (2016), pp. 1–26 (cit. on pp. 6, 7, 12).

[9] Jean-Philippe Tardif, Yanis Pavlidis, and Kostas Daniilidis. «Monocular visual odometry in urban environments using an omnidirectional camera». In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2008, pp. 2531–2538 (cit. on p. 8).

[10] Alberto Sanchez, Angel de Castro, Santiago Elvira, Guillermo Glez-de-Rivera, and Javier Garrido. «Autonomous indoor ultrasonic positioning system based on a low-cost conditioning circuit». In: *Measurement* 45.3 (2012), pp. 276–283 (cit. on p. 9).

[11] Johann Borenstein, Hobart R Everett, Liqiang Feng, and David Wehe. «Mobile robot positioning: Sensors and techniques». In: *Journal of robotic systems* 14.4 (1997), pp. 231–249 (cit. on p. 10).

[12] Lasitha Piyathilaka and Rohan Munasinghe. «An experimental study on using visual odometry for short-run self localization of field robot». In: *2010 Fifth International Conference on Information and Automation for Sustainability*. IEEE. 2010, pp. 150–155 (cit. on p. 10).

[13] Aboelmagd Noureldin, Tashfeen B Karamat, and Jacques Georgy. «Fundamentals of inertial navigation, satellite-based positioning and their integration». In: (2013) (cit. on p. 10).

[14] Thomas Whelan, Hordur Johannsson, Michael Kaess, John J Leonard, and John McDonald. «Robust real-time visual odometry for dense RGB-D mapping». In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 5724–5731 (cit. on p. 11).

[15] Michael Dille, Ben Grocholsky, and Sanjiv Singh. «Outdoor downward-facing optical flow odometry with commodity sensors». In: *Field and Service Robotics*. Springer. 2010, pp. 183–193 (cit. on p. 11).

[16] Ignacio Parra Alonso, David Fernández Fernández Llorca, Miguel Gavilan, Sergio Álvarez Álvarez Pardo, Miguel Ángel Garcıa-Garrido, Ljubo Vlacic, and M Ángel Sotelo. «Accurate global localization using visual odometry and digital maps on urban environments». In: *IEEE Transactions on Intelligent Transportation Systems* 13.4 (2012), pp. 1535–1545 (cit. on p. 11).

[17] Geovanni Martinez. «Monocular visual odometry from frame to frame intensity differences for planetary exploration mobile robots». In: *2013 IEEE Workshop on Robot Vision (WORV)*. IEEE. 2013, pp. 54–59 (cit. on p. 11).

[18] Takeshi Takahashi. «2D localization of outdoor mobile robots using 3D laser range data». In: *Robotics Institute Carnegie Mellon University Pittsburgh, Pennsylvania* 15213 (2007) (cit. on p. 12).

[19] Ramon Gonzalez, Francisco Rodriguez, Jose Luis Guzman, Cedric Pradalier, and Roland Siegwart. «Control of off-road mobile robots using visual odometry and slip compensation». In: *Advanced Robotics* 27.11 (2013), pp. 893–906 (cit. on pp. 12, 13).

[20] Friedrich Fraundorfer and Davide Scaramuzza. «Visual odometry: Part i: The first 30 years and fundamentals». In: *IEEE Robotics and Automation Magazine* 18.4 (2011), pp. 80–92 (cit. on p. 12).

[21] Davide Scaramuzza, Friedrich Fraundorfer, and Marc Pollefeys. «Closing the loop in appearance-guided omnidirectional visual odometry by using vocabulary trees». In: *Robotics and Autonomous Systems* 58.6 (2010), pp. 820–827 (cit. on p. 13).

[22] David Nistér, Oleg Naroditsky, and James Bergen. «Visual odometry for ground vehicle applications». In: *Journal of Field Robotics* 23.1 (2006), pp. 3–20 (cit. on p. 13).

[23] Daobin Wang, Huawei Liang, Hui Zhu, and Shuai Zhang. «A bionic camera-based polarization navigation sensor». In: *Sensors* 14.7 (2014), pp. 13006–13023 (cit. on p. 13).

[24] Bernd Manfred Kitt, Joern Rehder, Andrew D Chambers, Miriam Schonbein, Henning Lategahn, and Sanjiv Singh. «Monocular visual odometry using a planar road model to solve scale ambiguity». In: (2011) (cit. on pp. 14, 15).

[25] Linhui Li, Jing Lian, Lie Guo, and Rongben Wang. «Visual odometry for planetary exploration rovers in sandy terrains». In: *International Journal of Advanced Robotic Systems* 10.5 (2013), p. 234 (cit. on p. 14).

[26] Hochang Seok and Jongwoo Lim. «ROVO: Robust Omnidirectional Visual Odometryfor Wide-baseline Wide-FOV Camera Systems». In: *CoRR* abs/1902.11154 (2019). arXiv: 1902.11154. URL: http://arxiv.org/abs/1902.11154 (cit. on p. 14).

[27] Christopher J. Holder and Toby P. Breckon. «Learning to Drive: Using Visual Odometry to Bootstrap Deep Learning for Off-Road Path Prediction». In: *2018 IEEE Intelligent Vehicles Symposium (IV)*. 2018, pp. 2104–2110. DOI: 10.1109/IVS.2018.8500526 (cit. on p. 17).

[28] Ruihao Li, Sen Wang, Zhiqiang Long, and Dongbing Gu. *UnDeepVO: Monocular Visual Odometry through Unsupervised Deep Learning*. 2017. DOI: 10.48550/ARXIV.1709.06841. URL: https://arxiv.org/abs/1709.06841 (cit. on p. 17).

[29] Ming He, Chaozheng Zhu, Qian Huang, Baosen Ren, and Jintao Liu. «A review of monocular visual odometry». In: *The Visual Computer* 36.5 (2020), pp. 1053–1065 (cit. on pp. 18, 54).

[30] Gunnar Farnebäck. «Two-frame motion estimation based on polynomial expansion». In: *Scandinavian conference on Image analysis*. Springer. 2003, pp. 363–370 (cit. on p. 21).

[31] Bruce D Lucas, Takeo Kanade, et al. *An iterative image registration technique with an application to stereo vision*. Vol. 81. Vancouver, 1981 (cit. on p. 21).

[32] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. «Flownet 2.0: Evolution of optical flow estimation with deep networks». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2462–2470 (cit. on p. 25).

[33] Zachary Teed and Jia Deng. «Raft: Recurrent all-pairs field transforms for optical flow». In: *European conference on computer vision*. Springer. 2020, pp. 402–419 (cit. on pp. 25, 51).

[34] Pierre Godet, Alexandre Boulch, Aurélien Plyer, and Guy Le Besnerais. *STaRFlow: A SpatioTemporal Recurrent Cell for Lightweight Multi-Frame Optical Flow Estimation*. 2020. DOI: 10.48550/ARXIV.2007.05481. URL: https://arxiv.org/abs/2007.05481 (cit. on p. 26).

[35] Xiaohui Huang, Chengliang Yang, Sanjay Ranka, and Anand Rangarajan. «Supervoxel-based segmentation of 3D imagery with optical flow integration for spatiotemporal processing». In: *IPSJ transactions on computer vision and applications* 10.1 (2018), pp. 1–16 (cit. on p. 28).

[36] Stephen M Smith and John M Brady. «ASSET-2: Real-time motion segmentation and shape tracking». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17.8 (1995), pp. 814–820 (cit. on p. 28).

[37] Peter O'Donovan. «Optical flow: Techniques and applications». In: *International Journal of Computer Vision* 1 (2005), p. 26 (cit. on p. 29).

[38] Shuai Hua, Manika Kapoor, and David C Anastasiu. «Vehicle tracking and speed estimation from traffic videos». In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 153–160 (cit. on p. 30).

[39] «https://www.synopsys.com/glossary/what-is-lidar.htm». In: *LIDAR*. 2022 (cit. on p. 32).

[40] «https://www.framos.com/en/products-solutions/3d-depth-sensing/depth-sensing-technologies». In: *DEPTH SENSING TECHNOLOGIES OVERVIEW*. 2022 (cit. on p. 33).

[41] «https://www.analyticsvidhya.com/blog/2019/10/detailed-guide-powerful-sift-technique-image-matching-python/». In: *A Detailed Guide to the Powerful SIFT Technique for Image Matching*. 2022 (cit. on p. 36).

[42] «https://docs.opencv.org/4.x/da/de9/tutorial$_p y_e pipolar_g eometry.html$». In: *Epipolar Geometry*. 2022 (cit. on p. 39).

[43] Carnegie Mellon University. «http://www.cs.cmu.edu/ 16385/s15/lectures/Lecture18.pdf». In: *Epipolar Geometry*. 2022 (cit. on p. 39).

[44] Robyn Owens. «https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL$_C OPIES/OWENS$/ In: *Epipolar Geometry*. 1997 (cit. on p. 40).

[45] Moritz Menze and Andreas Geiger. «Object Scene Flow for Autonomous Vehicles». In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 (cit. on p. 43).

[46] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. v.d. Smagt, D. Cremers, and T. Brox. «FlowNet: Learning Optical Flow with Convolutional Networks». In: *IEEE International Conference on Computer Vision (ICCV)*. 2015. URL: `http://lmb.informatik.uni-freiburg.de/Publications/2015/DFIB15` (cit. on p. 44).

[47] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. «A naturalistic open source movie for optical flow evaluation». In: *European Conf. on Computer Vision (ECCV)*. Ed. by A. Fitzgibbon et al. (Eds.) Part IV, LNCS 7577. Springer-Verlag, Oct. 2012, pp. 611–625 (cit. on p. 45).