Universidade Estadual de Campinas
Instituto de Computação

Hudson Martins Silva Bruno

LIFT-SLAM: a deep-learning feature-based visual SLAM method

LIFT-SLAM: um método de SLAM visual baseado em características com aprendizado profundo

CAMPINAS
2020

Hudson Martins Silva Bruno

LIFT-SLAM: a deep-learning feature-based visual SLAM method

LIFT-SLAM: um método de SLAM visual baseado em características com aprendizado profundo

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Dissertation presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Master in Computer Science.

Supervisor/Orientadora: Profa. Dra. Esther Luna Colombini

Este exemplar corresponde à versão final da Dissertação defendida por Hudson Martins Silva Bruno e orientada pela Profa. Dra. Esther Luna Colombini.

CAMPINAS
2020

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

B836L

Bruno, Hudson Martins Silva, 1995-
LIFT-SLAM : a deep-learning feature-based visual SLAM method / Hudson Martins Silva Bruno. – Campinas, SP : [s.n.], 2020.

Orientador: Esther Luna Colombini.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Visão por computador. 2. Aprendizado profundo. 3. Robótica. I. Colombini, Esther Luna, 1980-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: LIFT-SLAM : um método de SLAM visual baseado em características com aprendizado profundo
Palavras-chave em inglês:
Computer vision
Deep learning
Robotics
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Esther Luna Colombini [Orientador]
Paulo Lilles Jorge Drews Junior
Paula Dornhofer Paro Costa
Data de defesa: 04-05-2020
Programa de Pós-Graduação: Ciência da Computação

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0002-0963-0637
- Currículo Lattes do autor: http://lattes.cnpq.br/1474491202600835

Universidade Estadual de Campinas
Instituto de Computação

Hudson Martins Silva Bruno

LIFT-SLAM: a deep-learning feature-based visual SLAM method

LIFT-SLAM: um método de SLAM visual baseado em características com aprendizado profundo

Banca Examinadora:

- Profa. Dra. Esther Luna Colombini
  Instituto de Computação - Universidade Estadual de Campinas
- Prof. Dr. Paulo Lilles Jorge Drews Junior
  Centro de Ciências Computacionais - Universidade Federal do Rio Grande
- Profa. Dra. Paula Dornhofer Paro Costa
  Faculdade de Engenharia Elétrica e de Computação - Universidade Estadual de Campinas

A ata da defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Campinas, 04 de maio de 2020

Acknowledgments

We would like to thank CNPq and the company Quinto Andar for funding this project.

I would also like to thank my advisor Esther, who was not only instrumental in the development of this work, but also welcomed me to the University and was available to help me whenever necessary. I also thank all the professors and employees at Unicamp, especially the Institute of Computing, who provided us with a great work and research environment.

Many thanks to my girlfriend Jessica who gave me the strength to continue during this arduous journey. To my parents Angelo Jr. and Wanderleia to my sister Giovanna, who even being thousands of kilometers away, supported me in all my decisions. To all my family members, especially my grandparents Angelo and Alaiz, who accompanied me since graduation giving me love and affection.

I also thank my friends at the Laboratory of Robotics and Cognitive Systems (LaRoCS), especially Pedro, Patrick and Gabriel, for the important exchange of knowledge during the development of projects in robotics. Finally, I would also like to thank the friends I made outside Unicamp Bruno, Lorena, Marcos, Maykon, Nigel, Pedro, Renata and Rodrigo, for their invaluable friendship for many years.

Resumo

O problema de localização e mapeamento simultâneos (SLAM) aborda a possibilidade de um robô se localizar em um ambiente desconhecido e, simultaneamente, criar um mapa consistente desse ambiente. Um dos principais componentes do SLAM, chamado Odometria, é responsável por estimar a localização do agente e as mudanças de posição ao longo do tempo. Recentemente, as câmeras foram usadas com êxito para obter as características do ambiente para executar SLAM e Odometria, que são chamados SLAM visual (VSLAM) e Odometria visual (VO), respectivamente. No entanto, os algoritmos clássicos de VO e VSLAM podem ser facilmente induzidos a falhar quando o movimento do robô ou o ambiente são muito desafiadores. Portanto, novas abordagens baseadas em redes neurais profundas (DNNs) alcançaram resultados promissores em VO/VSLAM. Dessa forma, propomos combinar o potencial dos descritores de características baseados em aprendizado profundo com o VSLAM tradicional baseado em geometria, criando um novo sistema VSLAM para robôs móveis chamado LIFT-SLAM. As experiências realizadas nos conjuntos de dados KITTI e EuRoC mostram que o aprendizado profundo pode ser usado para melhorar a performance de sistemas de VSLAM tradicionais, uma vez que a abordagem proposta foi capaz de alcançar resultados comparáveis ao estado da arte sendo robusto à ruídos sensoriais.

Abstract

The Simultaneous Localization and Mapping (SLAM) problem addresses the possibility of a robot to localize itself in an unknown environment and simultaneously build a consistent map of this environment. One of the main components of SLAM is called Odometry, which is responsible for estimating the agent's location and changes in position over time. Recently, cameras have been successfully used to get the environment features to perform SLAM and Odometry, referred to as visual SLAM (VSLAM) and visual Odometry (VO), respectively. However, classical VO and VSLAM algorithms can be easily induced to fail when the robot's motion or environment is too challenging. Therefore, new approaches based on Deep Neural Networks (DNNs) have achieved promising results in VO/VSLAM. In this way, we propose to combine the potential of deep learning-based feature descriptors with the traditional geometry-based VSLAM, building a new VSLAM system for mobile robots called LIFT-SLAM. Experiments conducted on KITTI and EuRoC datasets show that deep learning can be used to improve traditional VSLAM systems' performance, as the proposed approach was able to achieve results comparable to the state-of-the-art while being robust to sensorial noise.

List of Figures

2.1 Comparison between SLAM and odometry. 18
2.2 Comparison between feature-based and direct VSLAM. 20
2.3 Comparison between sparse, semi-dense and dense VSLAM. 20
2.4 Typical VSLAM pipeline. 21
2.5 The epipolar geometry. 22
2.6 Pose graph representation. 24
2.7 A typical CNN architecture. 26
2.8 LIFT pipeline. 28
2.9 Siamese LIFT training architecture. 28

3.1 Overview of ORB-SLAM's pipeline. 32

4.1 Example of image from KITTI dataset 36
4.2 Example of image from EuRoC dataset 37

5.1 Comparison between features LIFT and ORB 42
5.2 LIFT-SLAM pipeline. 43
5.3 Keypoints and Outliers in feature tracking. 43
5.4 Keyframes and map points. 44
5.5 Example of trajectory when relocalization is performed. 45
5.6 Representation of the loop closure process. 46
5.7 Fine-tuning orientation estimator. 48
5.8 Matching thresholds problem. 50
5.9 Thresholds in Adaptive LIFT-SLAM. 51

6.1 2D trajectories comparison between LIFT-SLAM, ORB-SLAM and ground-truth in KITTI Dataset. 54
6.2 2D trajectories comparison between LIFT-SLAM, ORB-SLAM and ground-truth in EuRoC Dataset. 55
6.3 2D trajectories comparison between LIFT-SLAM fine-tuned with KITTI sequences and ground-truth in KITTI Dataset. 56
6.4 2D trajectories comparison between LIFT-SLAM fine-tuned with KITTI sequences and ground-truth in EuRoC Dataset. 57
6.5 2D trajectories comparison between LIFT-SLAM fine-tuned with EuRoC sequences and ground-truth in KITTI Dataset. 59
6.6 2D trajectories comparison between LIFT-SLAM fine-tuned with EuRoC sequences and ground-truth in EuRoC Dataset. 60
6.7 2D trajectories comparison between Adaptive LIFT-SLAM and ground-truth in KITTI Dataset. 61

6.8 2D trajectories comparison between Adaptive LIFT-SLAM and ground-truth in EuRoC Dataset. 62
6.9 2D trajectories comparison between Adaptive LIFT-SLAM fine-tuned with KITTI sequences and ground-truth in KITTI Dataset. 63
6.10 2D trajectories comparison between Adaptive LIFT-SLAM fine-tuned with KITTI sequences and ground-truth in EuRoC Dataset. 64
6.11 2D trajectories comparison between Adaptive LIFT-SLAM fine-tuned with EuRoC sequences and ground-truth in KITTI Dataset. 66
6.12 2D trajectories comparison between Adaptive LIFT-SLAM fine-tuned with EuRoC sequences and ground-truth in EuRoC Dataset. 67
6.13 Comparison of the 2D trajectories performed by the algorithms with and without distortion. 69

List of Tables

2.1 Matching score comparison. 30

4.1 Details of some of the datasets available for VSLAM. 35
4.2 Sequences from EuRoC MAV dataset. 36

5.1 Absolute Trajectory Error for different matching thresholds in KITTI and EuRoC datasets. 49

6.1 Comparison of results between ORB-SLAM and LIFT-SLAM in KITTI Dataset. 53
6.2 Comparison of results between ORB-SLAM and LIFT-SLAM in EuRoC Dataset. 55
6.3 Results of LIFT-SLAM fine-tuned with KITTI sequences in KITTI dataset. 56
6.4 Results of LIFT-SLAM fine-tuned with KITTI sequences in EuRoC dataset. 57
6.5 Results of LIFT-SLAM fine-tuned with EuRoC sequences in KITTI dataset. 58
6.6 Results of LIFT-SLAM fine-tuned with EuRoC dataset in EuRoC sequences. 58
6.7 Results of Adaptive LIFT-SLAM in KITTI dataset. 60
6.8 Results of Adaptive LIFT-SLAM in EuRoC dataset. 62
6.9 Results of Adaptive LIFT-SLAM fine-tuned with KITTI sequences in KITTI dataset. 63
6.10 Results of Adaptive LIFT-SLAM fine-tuned with KITTI sequences in EuRoC dataset. 64
6.11 Results of Adaptive LIFT-SLAM fine-tuned with EuRoC sequences in KITTI dataset. 65
6.12 Results of Adaptive LIFT-SLAM fine-tuned with EuRoC sequences in EuRoC dataset. 65
6.14 Comparison of LIFT-SLAM with results from literature in KITTI dataset. 70

List of Abbreviations and Acronyms

ATE	Absolute Trajectory Error
BA	Bundle Adjustment
BoW	Bag-of-Words
CNN	Convolutional Neural Network
DLT	Direct Linear Transformation
DNN	Deep Neural Network
DoF	Degrees of Freedom
IMU	Inertial Measurement Unit
LIFT	Learned Invariant Feature Transform
LSTM	Long-Short Term Memory
ORB	Oriented FAST and Rotated BRIEF
PTAM	Parallel Tracking and Mapping
RANSAC	Random Sample Consensus
RMSE	Root Mean Squared Error
RPE	Relative Pose Error
SfM	Structure from Motion
SIFT	Scale Invariant Feature Transform
SLAM	Simultaneous Localization and Mapping
VIO	Visual Inertial Odometry
VO	Visual Odometry
VSLAM	Visual Simultaneous Localization and Mapping

Contents

1 Introduction **14**
- 1.1 Objectives . . . 15
- 1.2 Contributions . . . 15
- 1.3 Text Organization . . . 16

2 Theoretical Background **17**
- 2.1 Simultaneous Localization and Mapping . . . 17
  - 2.1.1 Odometry . . . 17
  - 2.1.2 Visual Odometry and Visual SLAM . . . 19
  - 2.1.3 Drawbacks on geometry-based Visual SLAM . . . 25
- 2.2 Convolutional Neural Networks . . . 26
- 2.3 Deep-learning and Visual SLAM . . . 27
- 2.4 Learned Invariant Feature Transform . . . 27
- 2.5 Final Considerations . . . 30

3 Related Work **31**
- 3.1 Geometry-based Approaches . . . 31
  - 3.1.1 Direct methods . . . 31
  - 3.1.2 Feature-based methods . . . 32
- 3.2 Deep learning-based approaches . . . 33
  - 3.2.1 End-to-end methods . . . 33
  - 3.2.2 Hybrid methods . . . 34
- 3.3 Final Considerations . . . 34

4 Materials and Methods **35**
- 4.1 Materials . . . 35
  - 4.1.1 Datasets . . . 35
  - 4.1.2 Evaluation Metrics . . . 37
  - 4.1.3 Programming Languages and Tools . . . 38
  - 4.1.4 Hardware Specification . . . 38
- 4.2 Methodology . . . 38

5 LIFT-SLAM **40**
- 5.1 LIFT-SLAM Pipeline . . . 41
  - 5.1.1 Map Initialization . . . 41
  - 5.1.2 Tracking . . . 41
  - 5.1.3 Mapping . . . 44
  - 5.1.4 Loop closure and relocalization . . . 45
  - 5.1.5 Algorithm Parameters . . . 46

5.2 Versions of LIFT-SLAM 47
5.2.1 Fine-tuned LIFT-SLAM 47
5.2.2 Adaptive LIFT-SLAM 49
5.3 Final Considerations 51

6 Results **52**
6.1 LIFT-SLAM 53
6.2 LIFT-SLAM fine-tuned with KITTI sequences 55
6.3 LIFT-SLAM fine-tuned with EuRoC sequences 58
6.4 Adaptive LIFT-SLAM 60
6.5 Adaptive LIFT-SLAM fine-tuned with KITTI sequences 62
6.6 Adaptive LIFT-SLAM fine-tuned with EuRoC sequences 64
6.7 Robustness tests 67
6.8 Discussion and comparison with literature 70

7 Conclusion and Future Work **71**

Chapter 1

Introduction

The ability to know its localization in an environment is an essential task for mobile robots, and it has been a subject of research in robotics for decades. To correctly localize itself, the robot must know its pose (position and orientation) in the environment. The process that estimates this information is called **Odometry**. When the robot simultaneously localizes itself and constructs a map of the unknown environment, the algorithm is called **Simultaneous Localization and Mapping** (SLAM).

In the last decade, the advances in hardware technologies, such as embedded GPUs, allowed significant advances in mobile robots pose estimation. These advancements encouraged progress in camera-based methodologies of odometry and SLAM, called Visual Odometry (VO) and Visual SLAM (VSLAM). Much work has been done to develop accurate and robust VO and VSLAM systems, but traditional approaches depend on significant engineering effort on a classic pipeline: Initialization, feature detection, feature matching, outlier rejection, motion estimation, optimization, and relocalization. Furthermore, the traditional approaches tend to fail in challenging environments (inadequate illumination, featureless areas, etc.) or when the camera is at high speed. Moreover, if the camera is monocular, these systems suffer from scale uncertainty.

VSLAM and VO have many real-world applications. The most important is in Virtual and Augmented Reality (VR/AR) and robotics. In mobile robotics, VSLAM is used to estimate the robot's position, especially in GPS-denied environments. Examples include autonomous cars, Unmanned Aerial Vehicles (UAVs), underwater robots, and space robots. It is applied in UAVs to perform autonomous environment exploration and navigation with low-cost, light-weight, and low-power consuming hardware [86]. In underwater scenarios, VSLAM is a cost-effective solution for autonomous underwater vehicles and coral-reef inspection systems [25]. In space, VO is used by the Mars Exploration Rovers to aid the wheel odometry [9].

Advances in artificial intelligence, led by deep learning, are revolutionizing the computing scenario leading to a significant change in how applications are being created in several fields. The impact of deep learning in computer vision has been significant, and it is making substantial improvements to traditional robotics algorithms, including VO and VSLAM.

Recent applications of deep learning-based methods in VO/VSLAM have achieved promising results, bringing robustness to the situations mentioned above. Many works

have proposed using Deep Neural Networks (DNNs) to estimate camera motion with an end-to-end system. These systems can replace the entire traditional VSLAM pipeline that depends on significant engineering effort to develop and tune [47, 67, 87]. However, these methods are not able to outperform traditional methods yet. Thus, some works propose to replace only some modules of the VSLAM traditional pipeline with DNNs, creating hybrid methods [14, 39, 48]. This approach can leverage the robustness of deep learning to enhance traditional VSLAM systems. In this way, we aim at exploring the potential of deep neural networks to improve the performance of conventional VO/VSLAM systems.

1.1 Objectives

Our main objective in this work is to develop a visual SLAM system that takes advantage of deep neural network robustness to extract features from images and use them in a traditional VSLAM pipeline. We aim to develop this system for monocular camera applications. Therefore, we explore a DNN called LIFT [92] as our feature extractor and exploit the well-known ORB-SLAM [59] algorithm to compute the pose and the map of the environment.

More formally, our objectives are:

- To study the main techniques for monocular feature-based VO/VSLAM;
- To study deep learning-based feature extraction and description techniques;
- To develop a hybrid system that uses deep learning to perform feature extraction and description in a traditional VO/VSLAM pipeline;
- To evaluate our system and make comparisons with state-of-the-art results;
- To explore, apply, and evaluate other features that can improve the performance of our system.

Based on these objectives, we formulated some hypotheses to confirm if our objectives were achieved:

1. H_1: Deep Learning-based feature detection and description can be used to enhance VO/VSLAM systems;
2. H_2: The resulting system can successfully perform VO/VSLAM in different types of environments.

1.2 Contributions

The contributions of this work are the following:

- A review on feature-based VO/VSLAM algorithms and deep learning-based algorithms;

- The implementation of a new hybrid VO/VSLAM algorithm based on the LIFT network to perform feature detection and description in a traditional back-end based on ORB-SLAM's system;
- Two versions proposed to improve the system's performance in different environments.

1.3 Text Organization

This dissertation is organized as follows:

- Chapter 2 presents the necessary background knowledge relevant to this dissertation, such as algorithms that were used in the development of the project;
- Chapter 3 shows related works available in the literature about traditional VSLAM and deep learning-based VO/VSLAM;
- Chapter 4 presents the materials used to develop this work as long as the methodology we have employed to test our hypotheses;
- Chapter 5 describe the proposed VSLAM system and other methods we propose to improve the performance of the system;
- Chapter 6 presents the results of our experiments with comparisons between different works and a discussion about these results;
- Chapter 7 presents the final considerations of this work and the possibilities of future work.

Chapter 2

Theoretical Background

In this chapter, we will present fundamental concepts for the development of this work: the main concepts of SLAM, odometry, and details about the visual SLAM pipeline, a brief explanation of Convolutional Neural Networks, and how they are employed in VSLAM. Lastly, we give information on the network called Learned Invariant Feature Transform, used in this work as our feature extractor.

2.1 Simultaneous Localization and Mapping

The Simultaneous Localization and Mapping (SLAM) problem addresses the possibility of an agent locating itself in an unknown environment and simultaneously building a consistent map of this environment. SLAM is a crucial problem in real-world applications, such as autonomous robots, virtual reality, and augmented reality. The solution to the SLAM problem is one of the notable successes of the robotics community [17].

Several SLAM algorithms were proposed over history, with a typical structure that relies on two main components: Odometry and Loop-closure detection. Odometry is one of SLAM's central parts, which is responsible for estimating the agent's pose (position and orientation) in the environment through the estimate of changes in position over time. On the other hand, the loop closure detection is responsible for detecting when a position is being revisited, which is crucial to correct drift on the map.

SLAM's goal is to obtain a consistent global estimate of the robot path, which implies keeping track of a map of the environment. This aspect of SLAM differs from odometry that only cares about the local consistency of the trajectory. Figure 2.1 shows a comparison between SLAM and odometry. The choice between odometry and SLAM depends on the trade-off between performance and consistency and simplicity in implementation. However, the global consistency of the trajectory is sometimes desirable, odometry trades off consistency for real-time performance [17].

2.1.1 Odometry

Odometry is the most widely used method for determining the temporary position of a mobile robot [27]. Early odometry methods were performed by monitoring the robot's wheels rotations to compute the offset from a known starting position. Therefore, wheel

Figure 2.1: Comparison between SLAM and odometry on a self-driving car trajectory. The ground-truth positions are provided by a GPS are in the green line. The blue line shows the path estimated by odometry. The red line shows the global path estimated by SLAM, performing loop-closure to improve previous positions. Adapted from [45].

rotation measurements are incrementally used in conjunction with the robot's kinematic model to find its current location to a global reference coordinate system [71]. Nonetheless, since the wheel odometry produces incremental measurements, the position errors are accumulated over time and cause the estimated pose to drift from the ground-truth pose. The position errors are caused mainly by wheel slippage in uneven terrain or slippery floors [71]. Developing an error model to estimate the slippage is too difficult because the inaccuracies are highly dependent on surfaces. For example, the robot's wheel will slip differently on carpet than on a polished floor [60]. Furthermore, this method is limited to wheeled ground vehicles, which limits its applications.

An alternative to deal with wheel odometry problems is localization based on Inertial Measurement Units (IMUs). On inertial localization, the measurements are provided by motion sensors (accelerometers) and rotation sensors (gyroscopes) to track the position and orientation of an object [90]. However, this method is highly prone to drift accumulation because the calculation of the change in speed and position requires successive mathematical integrations of acceleration concerning time. Thus, the errors are cumulative and increase with time, making IMU-based localization unsuitable for positioning applications over an extended period. This method is used to aid other navigation systems to provide a higher accuracy rather than to replace them [2].

Many localization methods rely on a Global Positioning System (GPS) sensor, but it is only useful outdoors and might not work indoors and in confined spaces [2]. Additionally, it could have errors in order of meters, which is unacceptable for localizing human-scale mobile robots as well as miniature mobile robots [74]. Differential GPS and real-time kinematic GPS can provide the position with centimeter accuracy, but these techniques are too expensive and have significant latency.

Light detection and ranging sensors (LiDARs) are also possible sensors used on methods related to localization. LiDARs work by sending laser pulses and measuring the time until it returns. For odometry purposes, each point detected on consecutive scanning is used for pose estimation. Moreover, LiDARs possess a high sampling rate, high angular resolution, significant range detection, and high robustness against environmental variability. The main drawback of using these sensors is that they are costly. Furthermore, the

laser scanning can fail in a transparent material, such as glass, because of the reflections on these surfaces lead that to suspicious data [2].

Some works have used a set of on-board ultrasonic sensors as a low-cost alternative to LiDARs [18]; these sensors measure the time-of-flight of an ultrasonic pulse. However, the reflection of signal waves is highly dependent on the material and the orientation of the object surface, thus the ultrasonic sensor often produces inaccurate measurements of range. Moreover, this sensor has a slow response, which tends to worsen when a robot has a set of ultrasonic sensors. It needs to measure distance in sequence to minimize interference between the sensors [74].

Cameras have also been used for odometry in mobile robots. Vision-based odometry is an inexpensive alternative technique that is relatively more accurate than conventional ones and has been the subject of recent research. The process of estimating the pose of an agent by using only visual input (from single or multiple cameras) is called Visual Odometry [71].

2.1.2 Visual Odometry and Visual SLAM

Recently, cameras have been successfully used to get the environment features to perform odometry. If odometry is based only on visual information, the technique is called Visual Odometry (VO). When extended to SLAM, it is called Visual SLAM (VSLAM). When also using IMU data to estimate motion, VO is referred to as visual-inertial odometry (VIO). In general, the technical difficulty of VSLAM is higher than that of other sensor-based SLAMs because cameras can acquire less visual input from a limited field of view (compared to LiDARs, for example). Thus, camera poses need to be continuously estimated, and the 3D structure of the environment simultaneously reconstructed [80].

There are several algorithms for VSLAM. Most of the state-of-the-art algorithms are geometry-based methods [59, 19, 20], which means that they rely on geometric constraints extracted from images to estimate motion. Geometry-based VSLAM algorithms are either feature-based (indirect) or direct, dense, semi-dense, or sparse. These classifications are applied in the context of different sensor combinations and modalities, including monocular (single camera sensor), stereo (two camera sensors), visual-inertial (camera and IMU), and RGB-D (camera and depth estimation) [22].

Algorithms based on tracking and mapping feature points are called feature-based approaches. These algorithms extract and match keypoints and use them to estimate the camera motion. The keypoints can be geometric features (corners, edges, etc.), or more sophisticated feature descriptors (SIFT, ORB, etc.). However, they can fail in textureless and featureless environments. Therefore, direct approaches were proposed. These direct algorithms do not detect feature points, but directly use the whole image (its pixel intensities) for tracking and mapping [80]. Nevertheless, the latter are more affected by changing lighting conditions due to the change in pixel intensities. Figure 2.2 shows a comparison between the feature-based and direct approaches.

Visual SLAM systems might also be classified in sparse, dense, or semi-dense. Sparse methods use and reconstruct only a selected set of independent points, while dense methods attempt to use and rebuild all pixels in each received frame. There are also semi-dense

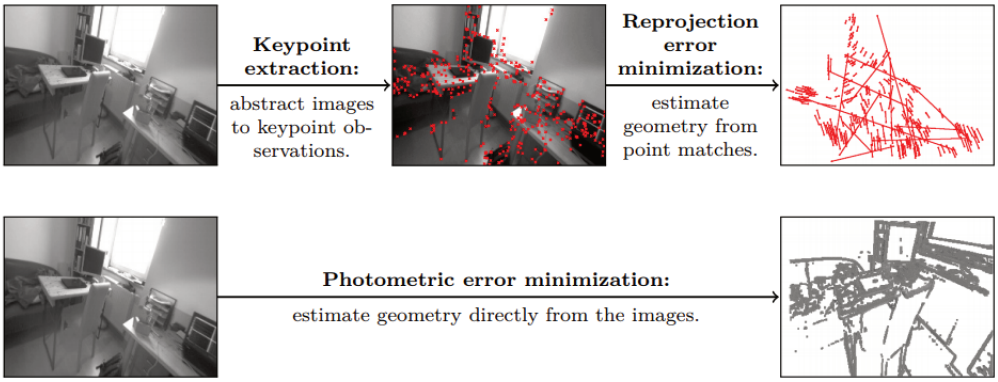


Figure 2.2: Comparison between feature-based (top) and direct approaches (bottom). The feature-based approach needs an extra step to extract the keypoints before estimating the geometry. Extracted from [22].

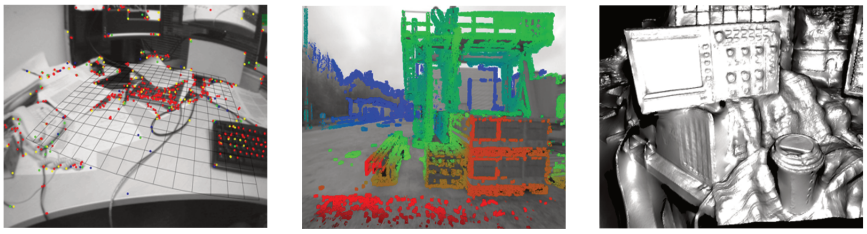


Figure 2.3: Comparison between sparse (left), semi-dense (middle) and dense (right) methods. The sparse map was created by the PTAM algorithm [40] (the colored points are the map points), the semi-dense map was constructed with the LSD-SLAM [19] (the colored points are map points) and the dense map was built with a DTAM algorithm [61] (all points are the map points). Adapted from [43].

methods that avoid reconstructing the entire surface but still aim to use and rebuild a subset of it. The maps generated from sparse methods are point clouds. Therefore, there is no notion of a neighborhood, and geometry parameters are conditionally independent, given the camera's poses and intrinsic model.

On the other hand, dense (or semi-dense) approaches to exploit the connectedness of the used image region to formulate a geometry prior. Still, because they use many more pixels than sparse methods, more powerful hardware is usually needed [22]. Figure 2.3 shows a comparison between the maps built with a sparse, semi-dense and dense method.

All of these algorithms usually follow a fundamental pipeline composed of initialization, tracking, mapping, global optimization, and relocalization (Figure 2.4). In initialization, a global coordinate system is defined for the agent. After this, the tracking step takes the image, performs a feature matching or tracking to get correspondences between the image and the map, and estimates the camera local pose. In the mapping step, a map is created from the images, expanding when unknown regions are found. In SLAM, a global optimization module is required to correct the global map as tracking introduces an accumulated drift error. Thus, after revisiting a map region, a loop closure is detected, and the reference information that represents the error from the beginning to the present can be computed. Finally, the relocalization module is required when the algorithm lost

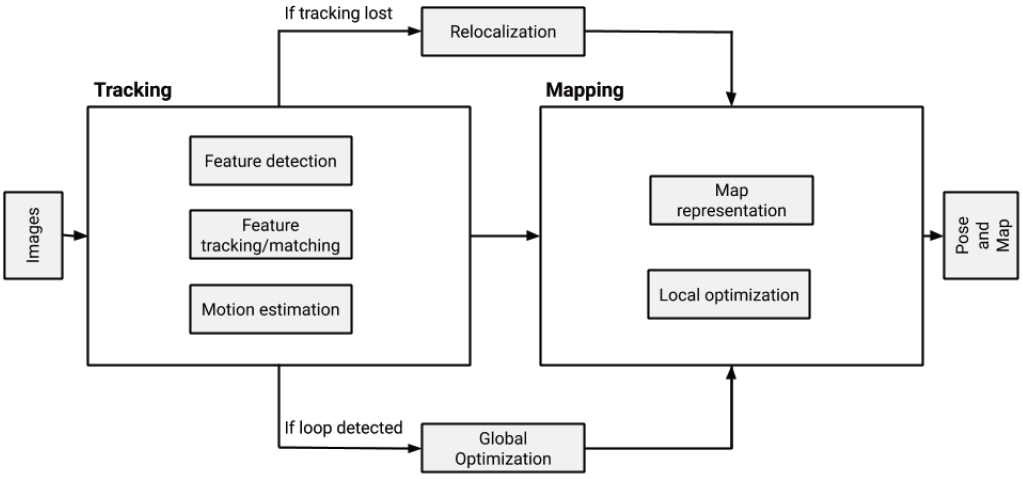


Figure 2.4: Typical VSLAM pipeline: Tracking gets the images and estimate the motion, mapping represents the camera poses and world features, relocalization is required when algorithm has lost camera tracking and global optimization is performed when a loop closure is detected.

track of the camera pose; therefore, the algorithm will try to estimate the camera pose concerning the map again. Some authors often condense this pipeline in two steps: front-end (initialization, tracking, and global optimization) and back-end (mapping) [7]. In the next sections, more details will be given about the main modules of the feature-based monocular Visual SLAM pipeline.

Tracking

Typically, one of the first steps in the tracking module is the local feature detection. In this way, feature detection algorithms are applied to look for keypoints (or points of interest). These points represent image locations that are potentially stable and repeatable from different lighting conditions and viewpoints. Hence, it is possible to find the same keypoints again in other images [83]. To improve feature matching or feature tracking, some algorithms perform a feature description process where each region around the detected feature is converted into a compact descriptor that can be matched against other descriptors [79]. Some of the most important traditional feature descriptors are Scale Invariant Feature Transform (SIFT) [50], Speeded Up Robust Features (SURF) [4] and ORB [68]. Recently, deep learning-based local features detectors and descriptors started to outperform the traditional approaches, such as Learned Invariant Feature Transform (LIFT) [92] and Super point [15].

After detecting and describing features in an image, the visual SLAM algorithm can either perform a feature tracking or a feature matching. The former approach is to detect features in one image and track them in the following images using local search techniques, such as correlation. However, this approach will work better if the sequence of images is taken from nearby viewpoints. The latter approach is to detect features in every image of the sequence and match them based on some similarity metric between their descriptors. Therefore, this approach is more suitable for sequences with large

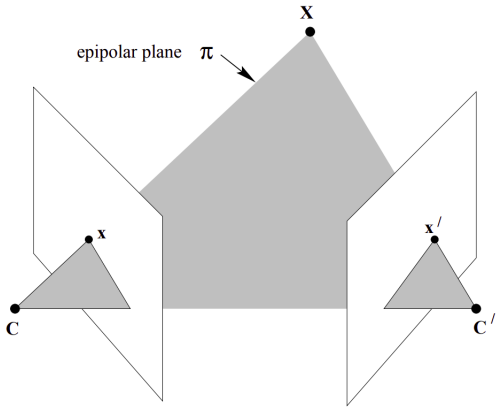


Figure 2.5: The epipolar geometry. $\mathbf{C}$ and $\mathbf{C}'$ are the cameras centers, $\mathbf{X}$ is the 3D point and $\mathbf{x}$ and $\mathbf{x}'$ are the reprojection of this point in the 2D images, that lie in a common plane π, named as epipolar plane. Extracted from [36].

viewpoint changes [26]. Matched keypoints usually are contaminated by the wrong data association. Consequently, it is necessary to remove the outliers with some outlier rejection algorithm, such as the Random Sample Consensus (RANSAC). To avoid such problem, some methods employ keypoint selection techniques, which also reduce the error in motion estimation [16, 11]. Then, the motion between the previous and the current frame is computed. Depending on whether the feature correspondences are specified in two or three dimensions, there are three different methods: 2D-to-2D, 3D-to-2D, and 3D-to-3D. The last one is not our focus since it is necessary to triangulate 3D points at each frame, which requires a stereo camera [71].

In 2D-to-2D methods, motion is estimated from the correspondences between two images. This approach is usually needed to compute the first two calibrated monocular frames, where map points have not been triangulated yet. Therefore, the epipolar geometry is used to calculate this transformation, as seen in figure 2.5. The epipolar geometry between two images is the geometry of the intersection of the image planes with the plane that connects both images with a point in space. Therefore, each camera reprojects in a 2D image a 3D point in the world. The perspective projection equation [36] gives the mapping from the 3D world to the 2D image. The epipolar geometry holds a constraint (equation 2.1) that determines that there is a line (epipolar line) on which the corresponding point x' from x lies in the other image, this constraint is useful to estimate the motion.

$$\mathbf{x'^T E x} = 0 \tag{2.1}$$

where $\mathbf{x}$ and $\mathbf{x}'$ are the corresponding points in image planes and E is the essential matrix, given by equation 2.2.

$$\mathbf{E} = [t]_\times R \tag{2.2}$$

where R is the rotation matrix, t is the translation matrix $t = [t_x, t_y, t_z]^T$, and $[t]_\times$ is the skew symmetric matrix given by equation 2.3.

$$[t]_\times = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & t_y \\ -t_y & t_x & 0 \end{bmatrix} \tag{2.3}$$

The solution of this motion estimation involves at least five correspondences of points; it is proposed in [62] an algorithm called 5-point algorithm to estimate motion efficiently in this case. Another alternative is to use the 8-point algorithm [49] or the normalized 8-point algorithm [35], for 8 or more correspondent points.

Moreover, the 3D-to-2D motion estimation for monocular VO needs to observe at least three different frames, since it needs to triangulate the first two frames into 3D points and its 2D correspondence in the third frame. The transformation T_k is obtained by reducing the reprojection error between the previous and the current frame with the equation 2.4.

$$\arg\min_{T_k} \sum_i \left\| \mathbf{x}_k^i - \hat{\mathbf{x}}_{k-1}^i \right\|^2 \tag{2.4}$$

where, T_k is the transformation for frame k, i is the number of correspondent points, $\mathbf{x}_k^i$ is the observed point in the current frame, $\hat{\mathbf{x}}_{k-1}^i$ is the reprojection of the 3D point X_{k-1}^i into the current frame according to the transformation T_k. This problem is often called as perspective from n points (PnP), the minimal case uses 3 point correspondences [41], however there are some well-known approaches that solves the problem for 4 or more points such as EPnP [46] and for 6 or more points that is the Direct Linear Transformation (DLT) algorithm [36].

Mapping

Maps are a stored representation of the robot's knowledge of the world, and they can be represented as topological, metric, or topological-metric maps [93]. Topological maps hold information about object adjacency avoiding metric information. Usually, topological maps are represented by a graph in which the nodes represent possible places in the world, and edges represent the possible paths between these places. In metric maps, the environment is represented in terms of geometric relations between the objects and a fixed reference frame (distances, directions, etc.). On the other hand, topological-metric (topometric) maps enhance topological maps by including metric information on the map edges. Furthermore, the metric information within the topological place node can be stored as a sparse landmark map or a dense occupancy grid map [51].

Key concern on SLAM systems is scalability, which means that it will keep efficiency in large maps. Many new monocular VSLAM methods are keyframe-based approaches, which means that the map is estimated using only some selected frames (keyframes). This approach avoids waste of computation to process consecutive frames with little new information. Recent VSLAM systems also represent the computed camera poses in pose graphs where the nodes are keyframe poses, and the edges are the rigid-body transformations between the camera poses, as shown in figure 2.6. This representation allows the system to optimize the camera poses with pose graph optimization or even to jointly optimize the camera pose and the 3D structure parameters with bundle adjustment (BA).

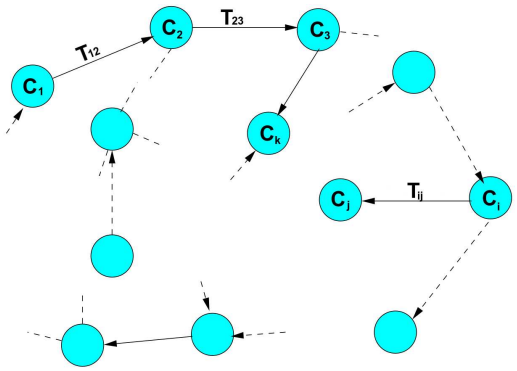


Figure 2.6: The pose graph representation of a VSLAM system. The camera poses C are represented as the graph nodes and the transformation between the camera poses T are the edges. Adapted from [33].

The pose graph optimization uses the constraints formed between non-adjacent poses by observing the same features from different locations [93]. These constraints defines the cost function presented in equation 2.5.

$$\sum_{i,j} \|C_i - T_{i,j}C_j\|^2 \tag{2.5}$$

where C is the camera center and T_{ij} is the rigid-body transformation between poses i and j. Pose graph optimization looks for the camera pose parameters that minimize this cost function. Generally, this is a non-linear optimization problem and to solve it a non-linear optimization algorithm is needed (e.g., Gauss-Newton, Levenberg-Marquardt) [26].

In BA, camera poses and 3D points are optimized simultaneously. It applies to the cases where image features are tracked over more than two frames, either globally or locally. If it is a global BA, all camera poses and 3D points are considered. On the other hand, in local BA (or windowed BA), the optimization is applied over several fixed frames (a window). The last approach is preferable because it limits the number of parameters for the optimization, making real-time bundle adjustment possible [26]. The cost function to be optimized in BA is presented in equation 2.6.

$$\sum_{X^i,C_k} \left\|x_k^i - g(X^i, C_k)\right\|^2 \tag{2.6}$$

where, x_k^i is the ith image point of the 3D point X^i measured in the kth image, and $g(X^i, C_k)$ is the reprojection function according to the camera pose C_k. The reprojection error is a non-linear function, and and a non-linear optimization algorithm is used to solve it.

Loop-closure detection and relocalization

Loop-closure detection and relocalization are similar tasks since both algorithms depend on place recognition to detect a revisited place in the environment. The difference is the moment each one is used, relocalization is done for recovering a camera pose, and loop-closure detection is used as a constraint in pose graph optimization. Thus it is useful for

obtaining a geometrically consistent map.

The loop can be detected based on three different approaches: map-to-map matching, image-to-image matching, and image-to-map matching [88]. Map-to-map approaches search for correspondent features between two sub-maps using similarity in visual appearance and the relative distance between features. Once it founds a consistent set of common features, the relative scale, rotation, and translation are estimated. On the other side, image-to-image approaches look for correspondences between the current image and the previously seen images. These methods do not depend on a metric map since it only compares the images. Lastly, in image-to-map algorithms, correspondences are searched between the current frame and the features on the map.

The image-to-image method is widely used because it can achieve a real-time performance using the bag of visual words model to represent the images. The bag of visual words model creates a quantization of the local features descriptors into a visual vocabulary. In the visual vocabulary, the original descriptor space is partitioned into non-overlapping cells by k-means clustering [26]. For each image, every feature is assigned to a particular visual word, ignoring any geometric structure. This approach allows images to be reduced to binary strings or histograms of length n, where n is the number of words in the vocabulary [51]. Vocabulary trees make this process efficient in large scale problems. They use a hierarchical model to define words, organizing them in an inverted file structure according to the Term Frequency Inverse Document Frequency (TF-IDF) scoring of the relevance of an image to the query [63]. The TF-IDF score is the product of two values: the term frequency, which measures how often the word appears in the image, and the inverse document frequency, which measures whether the word is common across all images.

2.1.3 Drawbacks on geometry-based Visual SLAM

Although VSLAM (and VO) is solved for well-defined environments and slowly moving robots, current SLAM algorithms can be easily induced to fail when either the motion of the robot (UAVs or bipedal robots) or the environment are too challenging [7]. Geometry-based approaches are sensitive to camera parameters and are not robust to challenging situations such as featureless areas, motion blur, large viewpoint changes, dynamic elements in the environment, and illumination changes [53].

Another issue, exclusive to monocular VSLAM algorithms, is the scale ambiguity problem since it is not possible to estimate depth from a single image. Therefore, these algorithms initially set the scale to a predefined value, and all map points are relative to this initial value. To obtain the real global scale, additional information about the world has to be available, such as the initial transformation or data from other sources (IMUs, LiDARs, etc.).

Moreover, these problems worsen when the robot uses a rolling shutter camera, the most commonly used camera model. In rolling shutter cameras, each row of a captured image is taken by different camera poses [80]. Therefore, the images that are acquired by a camera attached to moving robots may be distorted. When using cameras and inertial sensors, most approaches require exhausting manual synchronization of the IMU

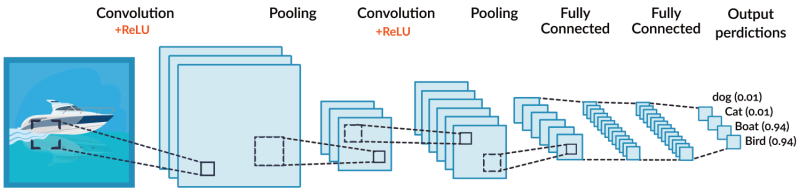


Figure 2.7: A typical CNN architecture with two convolutional layers, where ReLU are the non-linear activation functions. The fully connected layers after the convolutional layers are typically used to map the learned features to classification probabilities. Extracted from [54].

and camera as well as calibration of the IMU. Therefore, one of the main requirements for new VSLAM algorithms is performance robustness. In this way, several of the recent methods use Deep Neural Networks to perform VO and VSLAM.

2.2 Convolutional Neural Networks

A Deep Neural Network (DNN) is a representation learning technique that aims to learn high-level abstractions of a large amount of data by using multiple hierarchical layers of neural networks. Convolutional Neural Network (CNN) is one of the most popular DNN architectures. It is a specific kind of DNN for processing data that has a known grid-like topology [32], being widely applied in image applications.

The most important operation in CNN is the convolution, which is a weighted sum between two signals. The convolution output is a linear relationship between the neighboring values from each of the input values. The weight matrix used is called a kernel, and the convolution is applied by sliding the kernel over the input. For images, the convolution at an arbitrary location (i, j) on an input $\mathbf{I}$ with C channels is defined as equation 2.7.

$$H(i,j) = \sum_{c,n,m} \mathbf{I}(c, i \cdot s - n, j \cdot s - m)\mathbf{K}(c,n,m) \tag{2.7}$$

where, $\mathbf{K}$ is the kernel with size $N \times M$, s is a variable called stride, that defines how many pixels the kernel shifts in convolution.

Convolutional layers are the core component of the CNNs. These layers produce an output feature map using the convolution operation. Moreover, a convolutional layer is usually composed of three stages [32]. In the first stage, several convolutional operations produce some linear activations. In the second stage, a non-linear activation function is applied over the linear activations. The third stage involves a pooling function that will reduce the spatial size of the convolved features. Generally, CNNs are composed of multiple of these layers, as shown in figure 2.7.

A search through the parameters space is performed to look for the kernel weights that are more suitable for CNN to predict the desired output from any input. This process is called training, and it is driven by a loss function, which indicates how far the predictions are from the ground-truth data. The loss is a vital component of a DNN, as the objective of the network is to minimize the loss function in training. The resulting model is the

solution that best satisfies the loss function for all data.

A good property about DNNs, in general, is that after training the model in a specific dataset, it is still possible to adapt the weights of this model for a new data domain with a technique called transfer learning. Transfer learning is defined as the fine-tuning of the model for a new task. It occurs by the transfer of knowledge from a related task that was learned by the model [66].

2.3 Deep-learning and Visual SLAM

Recently, DNNs have gained a lot of popularity in the computer vision community. Recent developments on deep learning show that pose estimation can be treated as a learning problem [69]. DNNs can learn more robust and powerful features according to specific problems and have demonstrated its superiority in several computer vision tasks. Therefore, it seems unavoidable that Deep Learning-based visual SLAM algorithms will replace traditional VSLAM methods.

In this way, methods based on supervised learning, unsupervised learning, and reinforcement learning have been explored. Supervised learning methods usually train a DNN to learn geometric features by minimizing the difference between the predicted and the ground-truth poses. However, sometimes VO labeled data is costly to obtain, and it can limits generalization to new environments. On the other hand, unsupervised learning methods do not require ground-truth pose information to train the DNN but leverage some additional visual information that can assist the learning process. These include depth, stereo images, or optical flow [38]. Finally, reinforcement learning approaches do not use training data and rely on a reward signal to improve and construct policies that dynamically interact with the problem environment.

There are two main approaches proposed to use deep learning for VO/VSLAM systems: the end-to-end and hybrid strategies. In end-to-end methods, the DNN replaces the entire VO/VSLAM pipeline [87, 47, 67], these architectures generally use a Convolutional Neural Network (CNN) to extract features from the images and then estimate the motion. However, end-to-end approaches are still incapable of achieving the same performance and accuracy as state-of-the-art geometry-based approaches. To solve this problem, some authors propose to split the VSLAM pipeline and use DNNs to execute specific tasks, which we call hybrid strategies [10, 48, 31].

2.4 Learned Invariant Feature Transform

The Learned Invariant Feature Transform (LIFT) [92] is a DNN proposed by Yi et al. that implements local feature detection, orientation estimation, and description in a supervised end-to-end approach. The network architecture is composed of three main modules based on CNNs: Detector, Orientation Estimator, and Descriptor, as shown in figure 2.8.

The algorithm works with patches of images. After giving a patch of 128x128 pixels as input, the detector network provides a score map of this patch. A soft argmax operation [8] is performed over this score map to return the potential feature point location. After this,

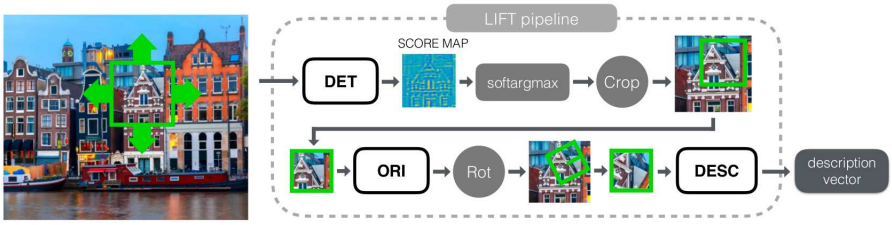


Figure 2.8: The LIFT Pipeline composed of three main modules: Detector, Orientation estimator and descriptor. Extracted from [92].

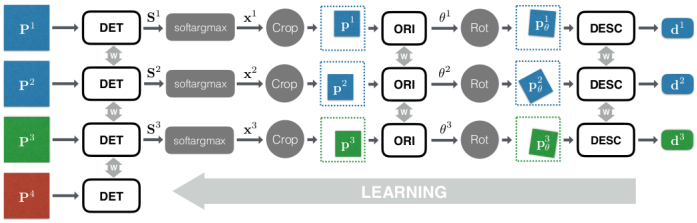


Figure 2.9: The Siamese LIFT training architecture, composed of four branches. Where $\mathbf{P^i}$ is a patch of a image inputed on the ith branch, S^i is a score map computed by the detector, $\mathbf{x^i}$ is a feature point location, and $\mathbf{p^i}$ is a smaller patch used as input to the orientation estimator. The orientation estimator computes a θ^i orientation that produces the rotated patch $\mathbf{p^i_\theta}$, this is processed by the descriptor network and produces a description vector $\mathbf{d^i}$. Extracted from [92].

a crop operation centered on the feature location is performed, returning a 64x64 patch. This patch is used as input to the orientation estimator. The orientation estimator module predicts an orientation to the patch. Thus, a rotation is applied in the patch according to the estimated orientation, and lastly, the descriptor network computes a feature vector.

LIFT was trained with photo-tourism image sets, as these images capture views of the same scene under different illumination conditions and seen from different perspectives. They have used a Structure from Motion (SfM) algorithm called VisualSFM [91] to reconstruct the scenes with SIFT features. They have adopted a problem-specific learning approach to train the network, which means that the descriptor network was trained alone; then, this network was used to train the orientation estimator. Finally, the detector is trained based on the already learned descriptor and orientation estimator. Moreover, the training architecture is a four-branch Siamese, as shown in figure 2.9. In training four patches of images are used as input, $\mathbf{P^1}$ and $\mathbf{P^2}$ correspond to different views of the same 3D point, $\mathbf{P_3}$ is a view from a different 3D point, and the last one $\mathbf{P^4}$ is a patch without any distinctive feature point. Each patch $\mathbf{P^i}$ corresponds to the ith branch of the network. It will be used as input. The last branch trains only the detector network, since it is used only to show the negative detector examples.

The descriptor network is trained with a loss to minimize the differences between the corresponding patches and maximizing the difference between different patches. The descriptor is formalized as $h_\rho(\mathbf{p}_\theta)$, where ρ are the descriptor parameters. The descriptor is trained to minimize the loss defined in equation 2.8.

$$L_{desc}(\mathbf{p}_\theta^k) = \begin{cases} \left\| h_\rho(\mathbf{p}_\theta^k) - h_\rho(\mathbf{p}_\theta^l) \right\|_2 & \text{for positive pairs} \\ \max(0, C - \left\| h_\rho(\mathbf{p}_\theta^k) - h_\rho(\mathbf{p}_\theta^l) \right\|_2) & \text{for negative pairs} \end{cases} \tag{2.8}$$

where C = 4, and positive pairs are patches that correspond to the same 3D point, and negative patches are the ones that do not correspond.

Moreover, the orientation estimator network is trained to provide the orientations that minimize the distances between description vectors for different views of the same 3D points. In training the orientation estimator, the description vectors are provided by the already trained descriptor, and the keypoints are taken from VisualSFM. The orientation estimator loss is defined in equation 2.9.

$$L_{ori}(\mathbf{P}^1, \mathbf{x}^1, \mathbf{P}^2, \mathbf{x}^2) = \left\| h_\rho(G(\mathbf{P}^1, \mathbf{x}^1) - h_\rho(G(\mathbf{P}^2, \mathbf{x}^2)) \right\|_2 \tag{2.9}$$

where $G(\mathbf{P}, \mathbf{x})$ is rotation applied to the patch P centered in location x.

Finally, the detector learns to minimize the distance between the descriptors vectors for correspondent patches (with the already learned descriptor and orientation estimator) and maximize the classification score for patches, not corresponding to the same physical points. Therefore, the loss in detector is the sum of two losses L_{class} and L_{pair}, as shown in equation 2.10. The detector output (score map) is defined as $f_\mu(\mathbf{P})$, where μ are the network parameters.

$$L_{det}(\mathbf{P}^1, \mathbf{P}^2, \mathbf{P}^3, \mathbf{P}^4) = \gamma L_{class}(\mathbf{P}^1, \mathbf{P}^2, \mathbf{P}^3, \mathbf{P}^4) + L_{pair}(\mathbf{P}^1, \mathbf{P}^2) \tag{2.10}$$

where γ is a hyper-parameter that defines a balancing between the two terms, with L_{class} increasing when detecting a keypoint in patch $\mathbf{P}^4$, as in equation 2.11. L_{pair} is defined in equation 2.12, it defines the distance between two correspondent description vectors.

$$L_{class}(\mathbf{P}^1, \mathbf{P}^2, \mathbf{P}^3, \mathbf{P}^4) = \sum_{i=1}^{4} \alpha_i \max(0, (1 - \text{softmax}(f_\mu(\mathbf{P}^i))y_i))^2 \tag{2.11}$$

where $y_i = -1$ and $\alpha_i = 3/6$ if $i = 4$ (for a non-keypoint patch), and $y_i = +1$ and $\alpha_i = 1/6$ otherwise. The softmax is a non-linear function.

$$L_{pair}(\mathbf{P}^1, \mathbf{P}^2) = \left\| h_\rho(G(\mathbf{P}^1, \text{softargmax}(f_\mu(\mathbf{P}^1)))) - h_\rho(G(\mathbf{P}^2, \text{softargmax}(f_\mu(\mathbf{P}^2)))) \right\|_2 \tag{2.12}$$

where the softargmax is a function that computes the center of mass of the score map, returning the feature location $\mathbf{x}$.

Yi et al. evaluated the algorithm in three datasets: Strecha dataset [78], DTU dataset [1] and Webcam dataset [85]. The algorithm was evaluated from three metrics: repeatability to captures the performance of the feature point detector, Nearest Neighbor mean Average Precision to capture how discriminating is the descriptor, and matching score that measures the overall performance of the pipeline. We present in table 2.1 a comparison between matches scores from LIFT and other feature descriptors. LIFT has presented good results on the matching score. Some LIFT's drawbacks involve the patches' extraction from the image (instead of using the whole image as input). The time taken to create

the feature descriptors is higher than the other approaches.

Dataset	SIFT	SURF	ORB	BRISK	LIFT-pic	LIFT-rf
Strecha	0.283	0.208	0.157	0.208	**0.374**	0.369
DTU	0.272	0.244	0.127	0.193	**0.317**	0.308
Webcam	0.128	0.117	0.120	0.118	0.196	**0.202**

Table 2.1: The average matching score for different algorithms. The best results are highlighted. LIFT-pic is the LIFT algorithm trained with Piccadilly Circus set of images and LIFT-rf is the algorithm trained with Roman Forum set of images [89]. Adapted from [92].

2.5 Final Considerations

This Chapter presented the theoretical background necessary to develop this work. Inspired by deep learning-based VSLAM systems shown in section 2.3, we created a hybrid VSLAM algorithm. To this end, we used the LIFT network presented in section 2.4 combined with a typical VSLAM pipeline described in section 2.1. Thus, with this approach, we expect to overcome some of the drawbacks from geometry-based VSLAM presented in section 2.1.3.

Chapter 3

Related Work

In this chapter, we show a literature review of the works regarding subjects explored in this dissertation. We divide this chapter into two parts: section 3.1 explores the most popular geometry-based VSLAM algorithms and their pipeline. Then, in section 3.2, we show some of the recent work with deep learning to enhance or replace the VSLAM traditional pipeline.

3.1 Geometry-based Approaches

The problem of estimating the pose of a vehicle from visual input alone started in the early 1980s [71], described in [55], where it was introduced the first motion-estimation pipeline. However, the term Visual Odometry was first presented by Nistér et al. in [64]. Since then, many methods have been proposed. We describe next some of the most relevant state-of-the-art geometry-based VO and VSLAM systems, either direct or feature-based.

3.1.1 Direct methods

Newcombe et al. proposed one of the first dense and direct SLAM methods for a monocular camera is Dense Tracking and Mapping (DTAM) [61]. They initialize the map using a stereo measurement, then estimate depth for each pixel; therefore, DTAM generates a dense 3D map for each frame. This algorithm performs tracking by comparing the input image with synthetic-view images generated from the reconstructed map.

Another relevant direct VSLAM algorithm called Large-scale Direct SLAM (LSD-SLAM) [19] was proposed by Engel et al.. This method is capable of running in large-scale environments with CPU. Textureless areas are ignored because it is difficult to represent their depth accurately. It represents maps with semi-dense inverse depth maps for selected keyframes, containing depth values for all pixels with a sufficient intensity gradient. They also provide a probabilistic solution to handle the noisy depth prediction during tracking. Moreover, pose-graph optimization is employed to obtain a geometrically consistent map.

Later, the authors of LSD-SLAM proposed a method called Direct Sparse Odometry (DSO) [20]. It combines photometric errors with geometric errors and performs a joint optimization of all parameters. The algorithm divides the input image into a grid, and

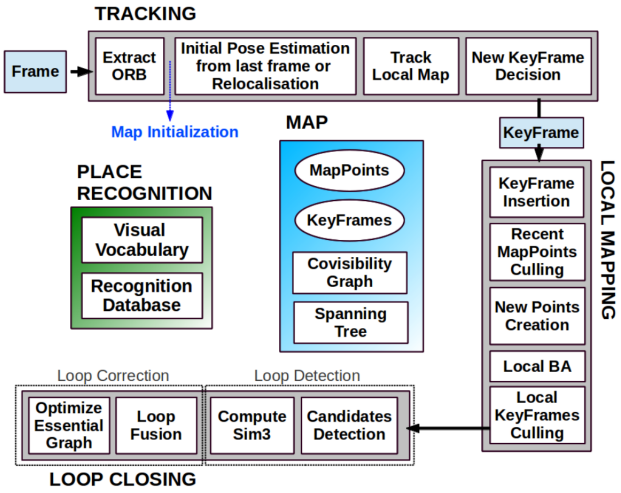


Figure 3.1: An overview of ORB-SLAM's pipeline with three threads: Tracking, Local Mapping and loop closing. The authors also propose to add a covisibility graph in the map and a place recognition module based on bag of visual words. Extracted from [59].

then, it selects high-intensity points as reconstruction candidates to its sparse map. By using this strategy, they can get points that are distributed within the entire image.

3.1.2 Feature-based methods

The first monocular VSLAM system proposed was MonoSLAM [12]. In this method, camera motion and 3D structure of an unknown environment are simultaneously estimated using an extended Kalman filter (EKF). In this method, there is no loop closure detection, and map initialization is done by using a known object. The main problem of this method is the computational cost, as it increases in proportion to the size of an environment.

To solve the problems of MonoSLAM, Klein and Murray proposed in [40], an algorithm called Parallel Tracking and Mapping (PTAM). To reduce computational costs, the authors propose split tracking and mapping into two separate tasks, processed in parallel threads. That way, the tracking estimates camera motion in real-time, and the mapping estimates accurate 3D positions of feature points with a computational cost [80]. It is the first real-time method that was able to incorporate BA. They have also created an automatic initialization with a 5-point algorithm. The main ideas of PTAM were used in ORB-SLAM.

ORB-SLAM [59] is a feature-based monocular VSLAM system proposed by Mur-Artal et al.. It works with three threads: Tracking, Local Mapping, and Loop Closing, as shown in figure 3.1. The system also adopts two graph representations: a covisibility graph and an essential graph. In the covisibility graph, each node is a keyframe, and an edge between two keyframes exists if they share observations of the same map points weighted by the number of map points in common. The essential graph is a lighter version of the covisibility graph. It contains the same nodes (keyframes) but fewer edges, which allows faster pose graph optimization.

The automatic map initialization system in ORB-SLAM is independent of the scene (planar or general) and do not require any human intervention. They compute two geo-

metrical models in parallel: a homography assuming a planar scene and a fundamental matrix considering a non-planar scene. So, they choose the model using a heuristic and try to recover the relative pose with a specific method for the selected model, avoiding low-parallax cases and planar ambiguity. In this way, the algorithm avoids initializing a corrupted map.

For tracking, they use a constant velocity model to predict the camera pose. This means that they assume a smooth camera movement and use the pose changes across the two previously tracked frames to estimate the initial pose for the current frame (assuming it will be optimized later). Then, the camera pose is optimized, and it is decided if the existing frame should be a keyframe. In local mapping, the keyframe is inserted in the covisibility graph, and new map points are created by triangulating ORB features from connected keyframes in the covisibility graph. A local bundle adjustment is then applied to all keyframes connected to the current keyframe in the covisibility graph and all map points seen by those keyframes.

ORB-SLAM's system has embedded a bags of words place recognition module for loop closure detection and relocalization. Visual vocabulary was created offline with ORB descriptors extracted from a large set of images. By using this vocabulary, the system builds incrementally a database, which stores, for each visual word in the vocabulary, in which keyframes it has been seen. So, querying the database can be done very efficiently. When relocalization is needed, the frame is converted into a BoW. Then, it queries the recognition database to find keyframe candidates for global relocalization. For loop detection, after converting the keyframes to BoW, a similarity between the current keyframe and its neighbors in the covisibility graph is computed, the lowest score is retained. After finding the loop candidates, it calculates the similarity transformation from the current keyframe to the loop keyframe, performing a pose graph optimization over the essential graph. Later, the authors of ORB-SLAM proposed an extension of ORB-SLAM applied to stereo and RGB-D cameras [58].

3.2 Deep learning-based approaches

Konda and Memisevic [42] presented one of the first end-to-end VO systems based on deep learning where a CNN was trained to relate local depth and motion representations to local changes in velocity and direction, using features from unsupervised pre-training. Since then, many approaches have been proposed to estimate VO from frames.

3.2.1 End-to-end methods

One of the most notable end-to-end methods is called DeepVO [87], proposed by Wang et al.. In DeepVO, a Recurrent Neural Network (RNN) is used to estimate the camera pose from features learned by a Convolutional Neural Network (CNN). The CNN architecture computes the optical flow from a sequence of images based on the Flownet [24]. Two stacked Long-Short Term Memory (LSTM) layers are applied to estimate temporal changes from the features predicted by the CNN.

Another end-to-end approach, based on unsupervised learning called UnDeepVO, is presented in [47]. The network is trained using stereo image pairs to recover the scale but tested using consecutive monocular images. Moreover, the loss function defined for teaching the networks is based on spatial and temporal dense information. The system successfully estimates the pose of a monocular camera and the depth of its view.

Parisotto et al. [67] proposed an end-to-end system that uses a similar architecture to DeepVO; however, instead of using LSTMs, they propose an attention phase, which is called Neural Graph Optimization. It considers that poses that are temporally adjacent should have similar outputs and should be visually similar. Still, temporally disparate poses should also have similar outputs, enabling a loop closure-like correction of drift.

3.2.2 Hybrid methods

Hybrid approaches were proposed to replace some modules of the traditional VSLAM pipeline. Li et al. proposed in [48], a monocular system called Neural Bundler, it is proposed an unsupervised DNN that estimates for motion. A conventional pose graph is constructed, enabling an efficient loop closing procedure based on the optimization of the pose graph estimated.

Another hybrid approach called SuperGlue [70] proposed a graph neural network with an attention mechanism to perform the matching between two sets of local features. They use the DNN between feature extraction and pose estimation, which they call a learnable "middle-end," as it lies between the front-end and back-end of a traditional VSLAM system. Furthermore, Tang et al. proposed in [81] a deep learning-based network for the generation of keypoints and descriptors called GCNv2. The network is designed with a binary descriptor vector, working as a replacement to ORB in ORB-SLAM2.

Recently, some papers proposed to use local learned features to replace the traditional local features (ORB, SIFT, etc.) of VSLAM systems. In DF-SLAM [39], the TFeat network [3] is used to create descriptors for features extracted from stereo images with the FAST corner detector. The feature descriptors are used in a traditional VSLAM pipeline, based on ORB-SLAM2 [58]. Another work has proposed a self-supervised approach called SuperPointVO [14], where they combine a DNN based in SuperPoint [15] feature extractor as a VSLAM front-end with a traditional back-end, using the stability of keypoints in the images to aid in learning.

3.3 Final Considerations

This chapter presented some of the methods proposed to perform VO and VSLAM in the literature. The most notable direct methods are LSD-SLAM [19] and DSO [20] and feature-based is ORB-SLAM [59]. The deep learning-based methods are very recent, and DeepVO [87] is one of the main end-to-end algorithms. From the hybrid methods, there are some approaches with similar algorithms pipeline as we proposed, such as GCNv2 [81], DF-SLAM [39], and SuperpointVO [14]. However, these works do not provide a set of experiments to evaluate the robustness of these algorithms.

Chapter 4

Materials and Methods

In this work, we develop a monocular VSLAM algorithm with deep learning-based features. To validate our algorithm, we use two datasets with different characteristics and a set of metrics for the evaluation of results. This chapter describes all material used to develop this project and the methodology we employed to obtain our results.

4.1 Materials

4.1.1 Datasets

There are a variety of datasets available that can be used for the development of VSLAM applications. Table 4.1 lists some of the most used datasets for VO/VSLAM with ground-truth information. In our work, we have used mainly the KITTI Vision Benchmark Suite [30] and EuRoC MAV Dataset [6].

Dataset	Focus	Environment	Cameras	Other Sensors
KITTI [30]	Self-driving cars	Outdoors	Global Shutter and Stereo	IMU, Laser and GPS
EuRoC MAV [6]	Micro Air Vehicles VIO	Indoors	Global Shutter and Stereo	IMU and Laser
TUM-Mono [21]	Tracking accuracy of monocular VO	Indoors and Outdoors	Global Shutter and Monocular	-
Event-Camera [56]	Micro Air Vehicles VIO	Indoors and Outdoors	Event camera, Global Shutter and Monocular	IMU
New College [75]	Long term SLAM	Indoors and Outdoors	Global Shutter, Stereo and Omnidirectional	IMU, Laser and GPS
Zurich Urban MAV [52]	Micro Air Vehicles VIO	Outdoors	Rolling Shutter and Monocular	IMU and GPS
TUM VI [72]	VIO	Indoors and Outdoors	Global Shutter and Stereo	IMU
UZH-FPV Drone Racing [13]	Autonomous Drone Racing	Indoors and Outdoors	Event Camera, Global Shutter and Stereo	IMU

Table 4.1: Details of some of the datasets available for VSLAM.

KITTI Dataset

KITTI vision benchmark suite (Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago) [30] is one of the most used benchmarks for evaluation in VO/VSLAM algorithms. They have developed benchmarks for stereo, optical flow, VO/VSLAM, and 3D object detection. The VO/VSLAM dataset consists of 22 stereo images sequences with a total length of 39.2 km recorded from a moving car. They also propose two metrics to evaluate VO/VSLAM systems: translation and rotation relative pose error, which will be described in section 4.1.2.

As our goal is to work with monocular images, we get only the left images in all sequences to run our algorithms. Moreover, we use only sequences from 00 to 10, as

Figure 4.1: Example of image extracted from KITTI dataset.

these are the only sequences with ground-truth information available. Figure 4.1 shows an example of an image we use from the KITTI dataset. This dataset will provide us large scale sequences in a real-life scenario. However, it contains limited motion patterns and environments since there is only a forward-motion (e.g., the car does not move backward), and all scenes are outdoors. Therefore, we also adopt the EuRoC MAV dataset in our experiments.

EuRoC MAV Dataset

The EuRoC MAV Dataset (Swiss Federal Institute of Technology/Autonomous Systems Lab) [6] is a dataset created to assess the visual-inertial SLAM and 3D reconstruction capabilities of contestants from the European Robotics Challenge (EuRoC) on Micro Aerial Vehicles (MAVs). Eleven sequences are provided in total, ranging from slow flights under good visual conditions to dynamic flights with motion blur and poor illumination. As opposed to KITTI, this dataset offers small scale indoor scenes with 6 degrees of freedom (DoF), figure 4.2 shows an example of an image from EuRoC dataset.

There are two types of sequences. The first type is from images taken in a realistic industrial scenario, recorded in a machine hall (sequences from MH_01 to MH_05). The second type is from images taken inside a Vicon motion capture system, with obstacles placed over the scene (sequences from V1_01 to V1_03 and V2_01 to V2_03). They also provide a complexity level for each sequence: easy, medium, or difficult. Details from all sequences are given in table 4.2.

Sequence	Complexity	Length [m]	Average Vel. [m/s]/Ang. Vel. [rad/s]	Characteristics
MH_01	easy	80.6	0.44/0.22	good texture, bright scene
MH_02	easy	73.5	0.49/0.21	good texture, bright scene
MH_03	medium	130.9	0.99/0.29	fast motion, bright scene
MH_04	difficult	91.7	0.93/0.24	fast motion, dark scene
MH_05	difficult	97.6	0.88/0.21	fast motion, dark scene
V1_01	easy	58.6	0.41/0.28	slow motion, bright scene
V1_02	medium	75.9	0.91/0.56	fast motion, bright scene
V1_03	difficult	79.0	0.75/0.62	fast motion, motion blur
V2_01	easy	36.5	0.33/0.28	slow motion, bright scene
V2_02	medium	83.2	0.72/0.59	fast motion, bright scene
V2_03	difficult	86.1	0.75/0.66	fast motion, motion blur

Table 4.2: Sequences complexity, details and characteristics in EuRoC MAV dataset. Adapted from [6].

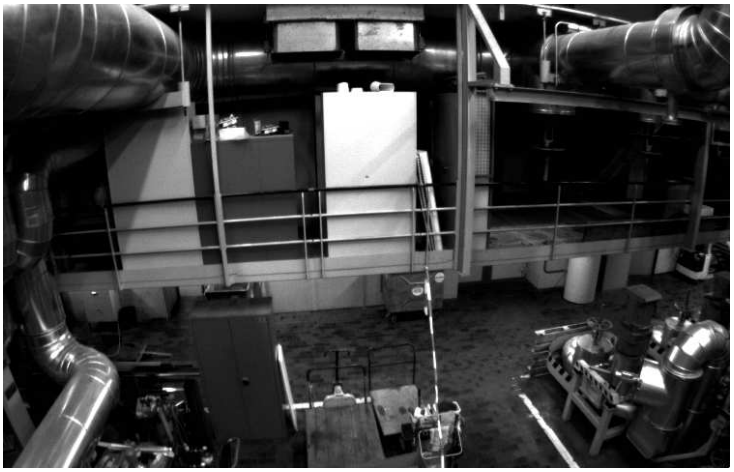


Figure 4.2: Example of image extracted from EuRoC dataset.

4.1.2 Evaluation Metrics

We have evaluated the performance of our approaches with a set of metrics. The accuracy of the VO/VSLAM estimates was evaluated with the Absolute Trajectory Error (ATE). In this metric, the corresponding poses from ground-truth trajectories and estimated trajectories are compared. This is useful to test the global consistency of the algorithm. However, both trajectories can be specified in different coordinates and scale. Therefore, we need to align them by using the Umeyama alignment method [84]. This method finds rigid-body transformations that map the estimated trajectory onto the ground-truth trajectory. Then, the error is computed, as shown in equation 4.1.

$$ATE = \frac{1}{N}\sum_{i=1}^{N} \|\hat{\mathbf{p}}_i - \mathbf{p}_i\|_2 \tag{4.1}$$

where, N is the number of frames, $\hat{\mathbf{p}}_\mathbf{i}$ is the estimated pose for frame i and $\mathbf{p}_i$ is the ground-truth pose for the same frame.

For sequences from the KITTI dataset, we used the relative pose error for translation and rotation, as described in [29]. Instead of a direct comparison of absolute poses, the relative pose error compares motions. This is possible due to the large scale nature of the dataset. Thus, these metrics capture different sources of error by evaluating error statistics over all sub-sequences of a given trajectory length or driving speed. Therefore, they evaluate errors as a function of the trajectory length and velocity. In KITTI benchmark page [1], they suggest using the subsequences of length from 100 to 800 meters, increasing by a factor of 100 (100, 200, ... 800). Moreover, these metrics use the inverse of the standard motion compositional operator described in [44]. Equation 4.2 is the relative pose error and equation 4.3 is the relative rotation error:

$$RPE_{trans} = \frac{1}{|\mathcal{F}|}\sum_{(i,j)\in\mathcal{F}} \|(\hat{\mathbf{p}}_j \ominus \hat{\mathbf{p}}_i) \ominus (\mathbf{p}_j \ominus \mathbf{p}_i)\|_2 \tag{4.2}$$

[1] `http://www.cvlibs.net/datasets/kitti/eval_odometry.php`

$$RPE_{rot} = \frac{1}{|\mathcal{F}|} \sum_{(i,j)\in\mathcal{F}} \angle[(\hat{\mathbf{p}}_j \ominus \hat{\mathbf{p}}_i) \ominus (\mathbf{p}_j \ominus \mathbf{p}_i)] \tag{4.3}$$

where $\mathcal{F}$ is a set of frames (i, j), $\hat{\mathbf{p}}$ and $\mathbf{p}$ are estimated and true camera poses respectively, $\ominus$ denotes the inverse of the standard motion compositional operator and $\angle[\cdot]$ is the rotation angle.

4.1.3 Programming Languages and Tools

The main programming language used to develop our approaches was C++. Furthermore, we used the Robotics Operating System (ROS) [76]. This framework has some tools that helped us in data synchronization and visualization, such as rosbag and rviz. Another important library used to manipulate images, and features were OpenCV [5].

Another programming language used in this project was Python 3. The main reason we used it was to evaluate our results and generate some visualizations, since there are many helpful libraries available in this language, such as NumPy [65] and Matplotlib [37]. To calculate all metrics, align trajectories, adjust the system scale, and plot the trajectories, we have used a python package to evaluate odometry and SLAM called Evo [34].

Moreover, we use Tensorflow [23], which is an open-source software library for high-performance numerical computation. It is mainly used for machine learning, especially deep neural networks, and it was fundamental to train our DNNs.

4.1.4 Hardware Specification

We execute our algorithms in a laptop with the following specifications:

- CPU: Intel® Core™ i7-850H CPU @ 2.20GHz;
- GPU: NVIDIA GeForce™ GTX 1050 Ti - 4 Gb
- RAM: 8 Gb

4.2 Methodology

To test our hypothesis, we use two datasets with completely distinct characteristics: KITTI and EuRoC, described in 4.1.1. This allowed us to test the robustness of the proposed algorithms for different camera motion (e.g., acceleration, velocities, DoF, etc.) and environments (e.g., outdoors/indoors, size, illumination, etc.). Moreover, in LIFT-SLAM fine-tuned approaches (section 5.2.1), using different datasets has allowed us to validate the improvement of the network for VO problems in general, instead of biasing the network for a single dataset.

To test our hypothesis our experiments are divided as following:

1. Evaluation of LIFT-SLAM:
    - Execution and evaluation in KITTI dataset;

- Execution and evaluation in EuRoC dataset;
- Comparison with ORB-SLAM results.

2. Evaluation of Fine-tuned LIFT-SLAM:
    - Execution and evaluation in KITTI dataset;
    - Execution and evaluation in EuRoC dataset;
    - Comparison with LIFT-SLAM results.
3. Evaluation of Adaptive LIFT-SLAM:
    - Execution and evaluation in KITTI dataset;
    - Execution and evaluation in EuRoC dataset;
    - Comparison with LIFT-SLAM results.
4. Comparison with state-of-the-art results.

Chapter 5

LIFT-SLAM

Based on the recent advances in deep learning methods applied in VSLAM approaches, we have explored the use of a Deep Neural Network to perform feature detection and description in the front-end of a typical VSLAM pipeline. In this chapter, we present our hybrid proposed approach addressing the VSLAM problem with features extracted based on a deep learning approach.

The chosen DNN was LIFT [92], described in section 2.4, as they provide the full feature point handling pipeline with state-of-the-art results. Moreover, we use a VSLAM pipeline based on ORB-SLAM's [59] pipeline, as they provide one of the complete feature-based monocular VSLAM systems. Therefore, we have implemented a deep-learning feature-based monocular VSLAM system called LIFT-SLAM. The method reconstructs sparse maps that are graph-based and keyframe-based, which allow us to perform bundle adjustment to optimize camera estimated poses.

As LIFT is based on SIFT, we expect it to be more accurate than ORB for scale, rotation, and affine variations. As discussed in [82], SIFT is one of the most precise feature descriptors in these situations. Therefore, before constructing the pipeline of LIFT-SLAM, we tested the robustness of LIFT descriptors under different scenarios. To this end, we performed a qualitative analysis of the LIFT feature matching over sequential images from the KITTI dataset. We compared them with ORB feature matching in the same conditions. First, we extract the features from a pair of images. Then we find the pair of descriptors with a smaller distance between them (similarity) to create a match. LIFT descriptors are vectors of float numbers. Therefore, we compute the distance between the two descriptors employing the Euclidean distance. On the other hand, ORB descriptors are binary vectors. Thus, we calculate the similarity between two ORB descriptors with Hamming distance.

We have created three different scenarios:

- Frame skipping: To evaluate the performance of the feature matching, we emulate different camera frequencies by skipping frames in sequences of the KITTI dataset. Therefore, given an image in time t, we look for feature matches with an image in time $t + 5$.

- Gamma power transformation with $\gamma > 1$: We can emulate under and overexposed images with gamma power transformation, as shown in [77]. This transformation

creates a new image I' from image I by applying: $I' = I^{\gamma}$. For $\gamma > 1$, we emulate an underexposed image.

- Gamma power transformation with $\gamma < 1$: We apply the same operation as before, but using $\gamma < 1$ to emulate an overexposed image.

From the qualitative results shown in figure 5.1, we concluded that LIFT is robust to all of the proposed scenarios. In frame skipping (figure 5.1a), we can notice that LIFT is still able to find correct correspondences between images, whereas ORB creates several wrong correspondences (figure 5.1b). In gamma power transformations, the feature matching with both descriptors can create correct correspondences, however, the ORB keypoints are grouped in only a few regions of the images (figures 5.1d and 5.1f). On the other side, the LIFT keypoints matched are spread within the whole image (figures 5.1c and 5.1e), this aspect can improve accuracy of VO systems, as discussed in [26].

On the other hand, the main drawback of using LIFT is its computational cost, once we did not optimize the network performance. Although the network does not have many parameters (approximately 290,000), the code to generate the descriptors is not optimized. Therefore, the time spent to generate the LIFT descriptors with the hardware described in section 4.1.4 is, on average, 36 seconds for images from the KITTI dataset and 35 seconds for images from EuRoC dataset.

5.1 LIFT-SLAM Pipeline

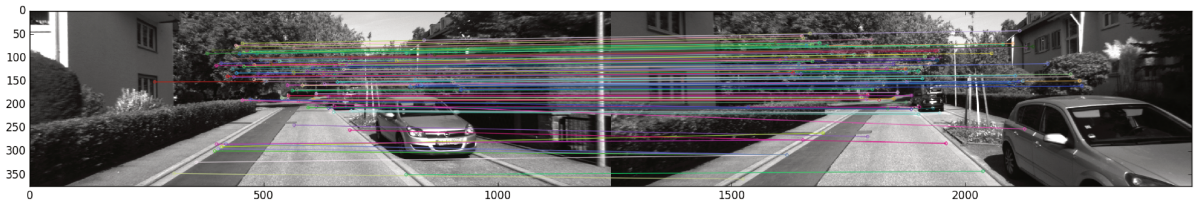
As aforementioned, our pipeline is similar to the pipeline of ORB-SLAM [59]. However, as we are not aiming, at this point, in an online version of the method, the mapping step runs sequentially after tracking and not in parallel, as in ORB-SLAM. Thus the only task we run in parallel is loop closure detection. Figure 5.2 shows an overview of our pipeline that will be described in the next sections.

5.1.1 Map Initialization

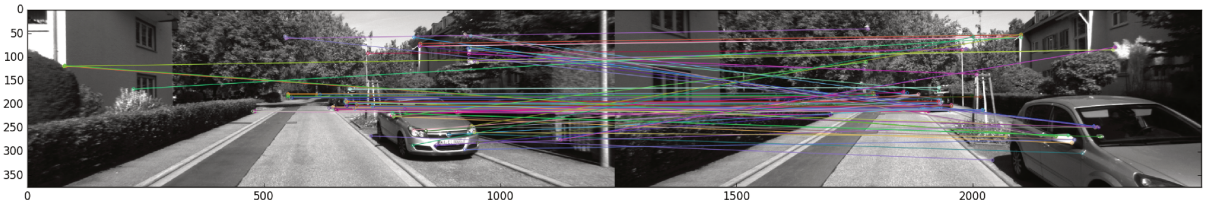
We follow the automatic map initialization steps defined in ORB-SLAM, computing in parallel two geometrical models: a homography, a relation between two images in the same plane, and a fundamental matrix, which relates to corresponding 3D points between the two images. Therefore, if the scene is planar (or there is low parallax), we should keep the homography model, as the fundamental matrix would generate the wrong results. On the other hand, if the scene is non-planar with enough parallax, the fundamental matrix should be selected. The homography model should be refused to create the correct map initialization. The model is selected based on a heuristic created in ORB-SLAM. After selecting the correct model, the motion hypotheses are retrieved, and a BA is performed.

5.1.2 Tracking

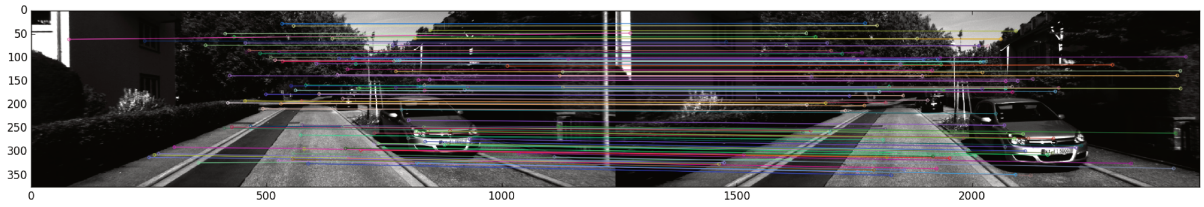
In tracking, for each frame, we extract LIFT keypoints and descriptors. These features are used in all feature matching operations that might be needed in initialization, tracking,



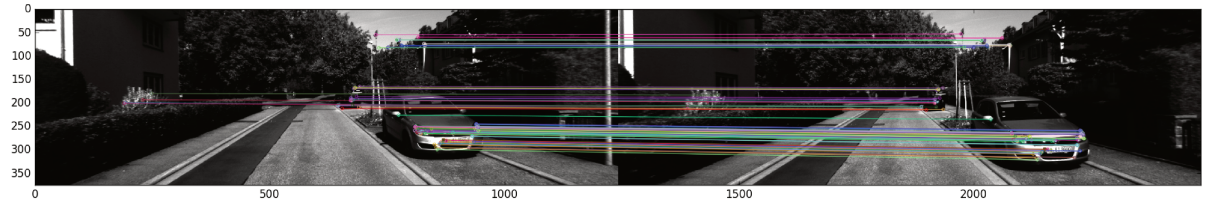
(a) LIFT feature matching in KITTI dataset after skipping 5 frames.



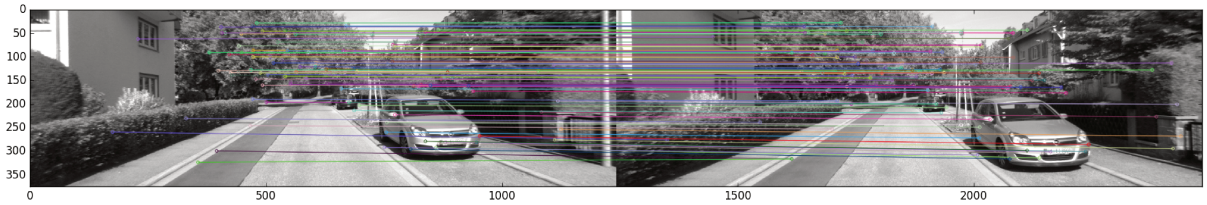
(b) ORB feature matching in KITTI dataset after skipping 5 frames.



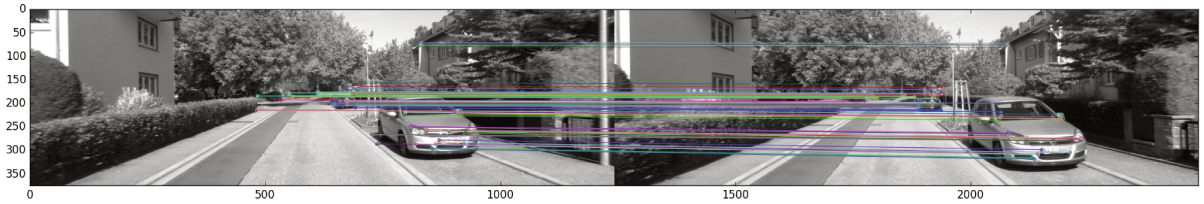
(c) LIFT feature matching in KITTI dataset after gamma power transformation with $\gamma = 2$.



(d) ORB feature matching in KITTI dataset after gamma power transformation with $\gamma = 2$.



(e) LIFT feature matching in KITTI dataset after gamma power transformation with $\gamma = \frac{1}{2}$.



(f) ORB feature matching in KITTI dataset after gamma power transformation of $\gamma = \frac{1}{2}$.

Figure 5.1: Comparison between features LIFT and ORB under different conditions. Lines are connecting correspondent keypoints computed by feature matching. We show only the best 100 matches in each figure.

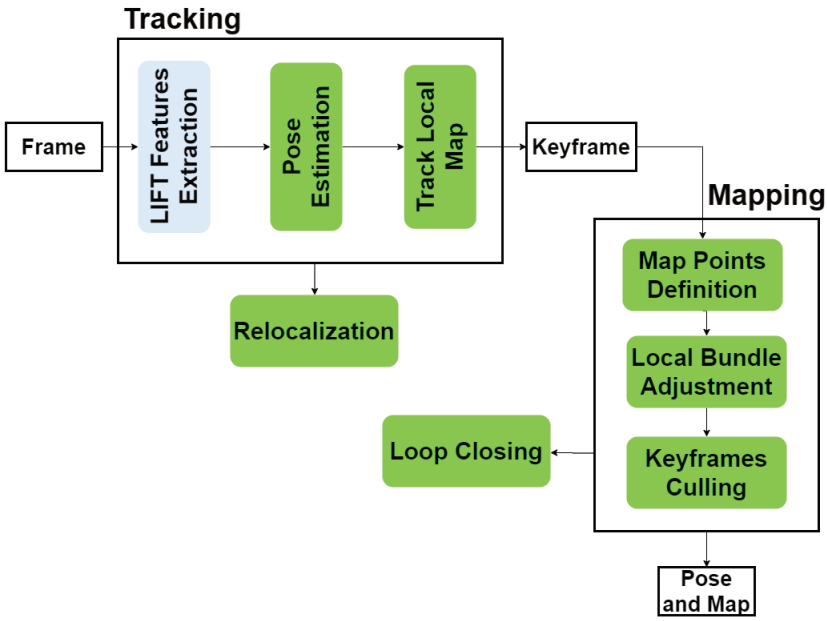


Figure 5.2: An overview of LIFT-SLAM pipeline, where tracking and mapping are sequential tasks, relocalization is called when the tracking of the camera pose is lost, and loop closing is a task that runs in parallel over the keyframes processed by mapping.

Figure 5.3: Keypoints and Outliers in feature tracking performed in KITTI dataset, green squares are the projected LIFT keypoints that were considered inliers and the red squares are the outliers.

mapping, and place recognition. Figure 5.3 shows an example of features extracted by LIFT and the outliers found after performing feature matching in the tracking step. Then, as in ORB-SLAM, the camera pose is predicted with a constant velocity model.

After estimating the pose, we track the local map. The local map is defined by the set of keyframes that share map points with the current frame and with its neighbors in the covisibility graph. The covisibility graph is a pose graph where each node is a keyframe, and the edges represent the number of observations of the same map points between two nodes (at least 15). To optimize the camera pose, we search for more map point correspondences in the current frame by projecting the local map 3D points into the image. We discard the point if it is not visible in the image. We also compare the LIFT descriptor of the map point with the still unmatched points near the projection and associate the map point with the best match (smaller Euclidean distance). After this, we perform a pose graph optimization with the found correspondences.

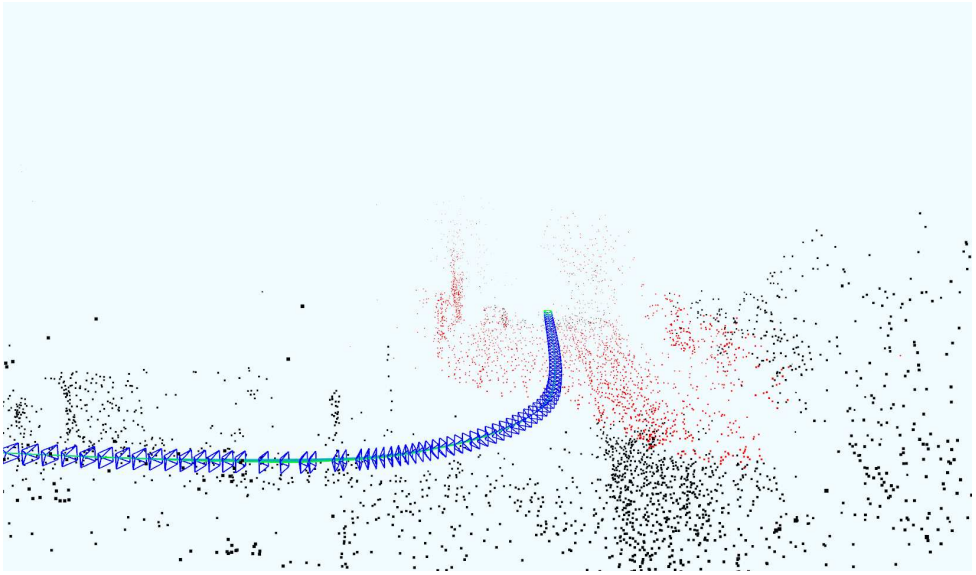


Figure 5.4: Representation of the keyframes and map points in our method, camera poses are represented in blue, current local map points in red and other map points in black.

Furthermore, the tracking step decides if the current frame should be a keyframe respecting some conditions: at least 20 frames have passed from the last relocalization, there are at least 15 points tracked in the current frame, and the current frame tracks less than 90% points than the last keyframe.

5.1.3 Mapping

For each new keyframe, we call the mapping step. First, we insert the keyframe into the covisibility graph as a new node, and its edges are computed based on the shared map points with other keyframes. Moreover, in this step, the BoW representation of this keyframe is computed.

Furthermore, new map points are created by triangulating LIFT features from keyframes connected in the covisibility graph. For each one of the unmatched features in the current keyframe, a match is searched with the other unmatched keypoints in the connected keyframes. Figure 5.4 shows a representation of our map. To speed up the triangulation process, we search for features that belong to the same node in the vocabulary tree. To be valid, the matches must fulfill the epipolar constraint (equation 2.1).

New map points are accepted after checking some constraints: positive depth in both frames, parallax, reprojection error, and scale consistency. During three keyframes after being created, map points must pass a restrictive test to be kept in the map, this is useful to avoid noisy estimates. The recent map points must fulfill two conditions: They need to be found in more than 25% of the frames in which it is predicted to be visible and, if more than one frame has passed since its creation, it must be seen from at least three keyframes.

In order to optimize the covisibility graph, a local bundle adjustment is applied to all keyframes connected to the current keyframe in the covisibility graph (including the current keyframe) and all map points seen by those keyframes. The keyframes that can

Figure 5.5: Example of trajectory when relocalization is performed, after losing the track the camera is relocalized after finding a revisited place.

see the same map points but are not connected in the covisibility graph are not optimized. However, they are used in the optimization process.

Finally, in keyframes culling, we discard keyframes that are redundant to improve the size of the covisibility graph. This is useful since BA is a costly operation that grows in complexity as the number of keyframes increases. Therefore, keyframes from which three other keyframes can see more than 90% of the map points in the same scale-level are discarded.

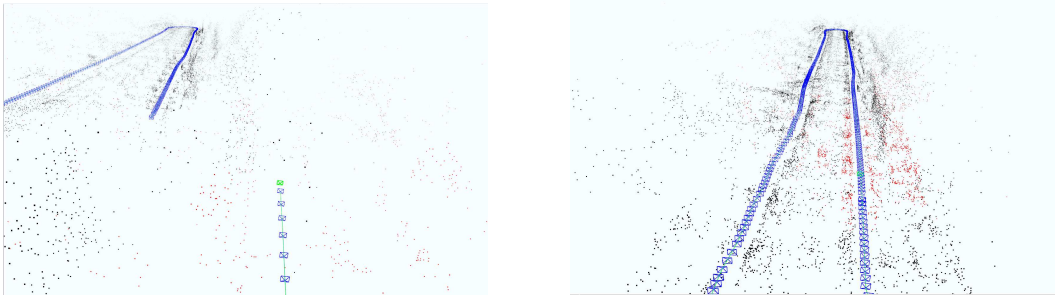
5.1.4 Loop closure and relocalization

In order to perform place recognition, we have created a visual vocabulary in an offline step with the DBoW2 library[1] [28]. The dictionary was created with LIFT descriptors of approximately 12,000 images collected from outdoors and indoors sequences from TUM-mono VO dataset [21], in such a way that we can generate a vocabulary that provides good results in both environments. The built vocabulary has six levels and 10 clusters per level. Thus we get 10^6 visual words, as suggested in [57]. Moreover, for each version of LIFT (original and fine-tuned described in section 5.2.1) we created a correspondent vocabulary.

After creating the keyframe's BoW in the VSLAM execution, a TF-IDF score is computed for each word presented in the image. The BoW of the current keyframe is then compared with the BoW vectors of the images in the database using an inverted index, which stores for each visual word, in which images it has been seen. So, if the tracking is lost, we query the BoW of the current frame into the database to find keyframe candidates for global relocalization. After relocalization, the last tracked pose is connected to the current pose with a straight line, as shown in figure 5.5

The loop closing task runs in a separate thread. It gets the last keyframe processed by

[1] `https://github.com/dorian3d/DBoW2`



(a) Pose graph before closing the loop. (b) Pose graph after closing the loop.

Figure 5.6: Representation of the loop closure process: after detecting a revisited place with place recognition the keyframes are adjusted based on the similarity transformation.

the local mapping and tries to detect if it closes a loop. After converting the keyframes to BoW, a similarity score between the current keyframe and its neighbors in the covisibility graph is computed, the lowest score s_{min} is retained. The similarity between two BoW is given by the L2-score, as defined in [63]. Then, all keyframes with a lower score than s_{min} in the recognition database are discarded. The loop candidates are accepted if there are at least three candidates detected in the same covisibility graph.

After finding the loop candidates, a rigid-body transformation from the candidate keyframe to the loop keyframe is computed, this transformation is called similarity transformation. The similarity transformation informs about the drift accumulated in the trajectory, and it also works as a geometrical validation of the loop. If a similarity transformation is successfully found, we proceed to correct the loop.

Before correcting the loop, the duplicated map points are fused, and the new edges are inserted in the covisibility graph. The loop adjusts the current keyframe with the similarity transformation. This correction is propagated to all of its neighbors, as shown in figure 5.6. Lastly, a pose graph optimization is performed over a reduced version of the covisibility graph, called an essential graph. After optimization, the map points are transformed according to the correction of one of the keyframes that observed it.

5.1.5 Algorithm Parameters

There are some fundamental parameters that need to be previously informed before executing LIFT-SLAM:

- Intrinsic and extrinsic camera calibration parameters [36], these parameters are different for each camera and, consequently, each dataset;
- Camera frame rate: the number of frames per second the camera can acquire;
- LIFT parameters:
  - Scale factor between levels in the scale pyramid [79] used to extract the features, in this project we set this value equals to 2, as it is the default in LIFT;
  - Number of levels in the scale pyramid, we also use the default value from LIFT, in this case it is 3;

– Matching thresholds: In this project we set these values to $TH_{LOW} = 1$ and $TH_{HIGH} = 2$ for EuRoC sequences and $TH_{LOW} = 2$ and $TH_{HIGH} = 3$ for KITTI sequences.

5.2 Versions of LIFT-SLAM

To explore the potential of our approach and to find changes that might lead to improved general results, we developed some different versions of LIFT-SLAM. The next sections describe the decision process to create these versions and how we developed them.

5.2.1 Fine-tuned LIFT-SLAM

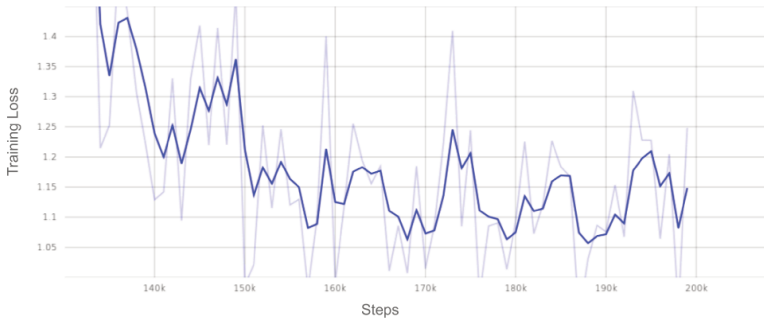
LIFT network was trained with photo-tourism image sets; this kind of data contains different geometrical aspects compared to a typical VO/VSLAM dataset. Usually, in VO/VSLAM datasets, the images are sequential, captured with the same camera that progressively changes its position and orientation. On the other hand, the photo-tourism images capture views of the same scene from different perspectives. Therefore, to address this aspect, we perform a transfer learning in LIFT network to generate a version of the LIFT that is fine-tuned with the geometrical nature of VO/VSLAM datasets.

To fine-tune the network, we had to collect the ground-truth data. As proposed in LIFT's paper [92], we generate the ground-truth with SIFT keypoints obtained with a structure from motion algorithm called VisualSFM [91]. This algorithm creates a 3D reconstruction of an environment given a set of images. Then, we created two sets of ground-truth data. We collected the first from sequences 00, 06, 09, and 10 (8434 images) of the KITTI dataset. The second dataset comes from the sequences MH_04, V1_03, and V2_03 (6104 images) of the EuRoC dataset.

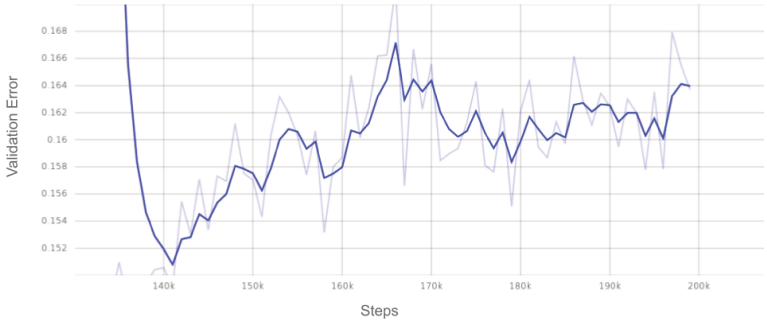
After collecting the datasets, we trained the network in two versions, one for each dataset. We performed it using the TensorFlow version of LIFT provided by the authors in their *github*[2]. The three LIFT modules were fine-tuned separately, following the same training procedure described in [92]: training first the descriptor, then, the orientation estimator, and lastly, the detector.

For both datasets, the only module that was improved in the validation set after training for a few iterations was the orientation estimator. We train the networks until they overfit, as shown in figure 5.7, which means that the training loss keeps improving, but the validation error is increasing. However, we only save the models that get the smaller validation error. The validation error is defined as $1-\text{AUC}$, where AUC is the Area Under the Curve of the validation loss defined in equation 2.9. Therefore, in this version of LIFT-SLAM, we use these fine-tuned models to perform feature detection and description.

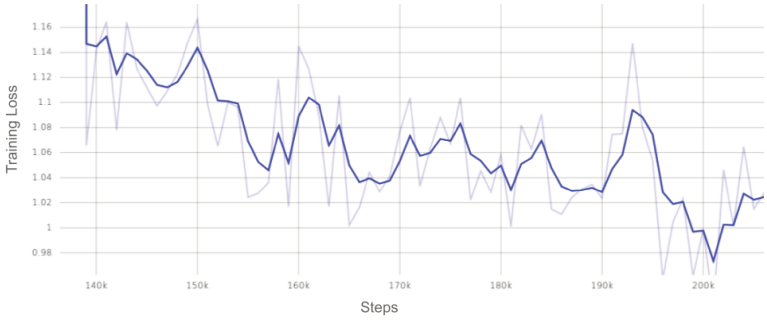
[2] `https://github.com/cvlab-epfl/tf-lift`



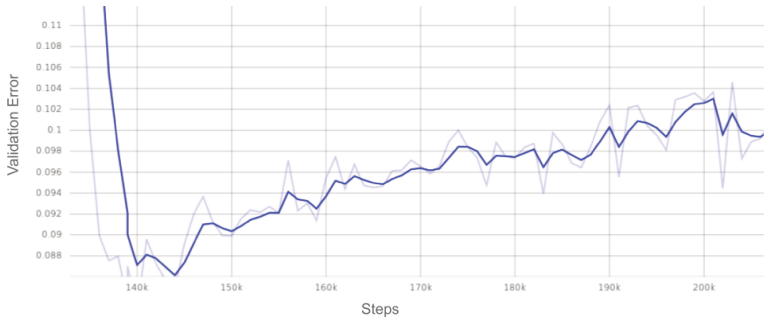
(a) Training loss in orientation estimator fine-tuning with KITTI dataset.



(b) Validation error in orientation estimator fine-tuning with KITTI dataset.



(c) Training loss in orientation estimator fine-tuning with EuRoC dataset.



(d) Validation error in orientation estimator fine-tuning with EuRoC dataset.

Figure 5.7: Fine-tuning orientation estimator. It is important to notice that the network was trained from scratch for approximately 130k steps. Therefore our fine-tuning starts from where it was stopped.

5.2.2 Adaptive LIFT-SLAM

A wrong data association in feature matching might affect the quality of the motion estimation. Therefore, to select the best features, a threshold is applied right after feature matching. In this way, the matches with greater distance than this threshold are discarded. On the other hand, if the threshold value is too small, we might discard good matches and loose track of the camera pose in challenging environments. To mitigate this problem, two thresholds are defined: the higher threshold (TH_{HIGH}) and the lower threshold (TH_{LOW}). We use TH_{LOW} when we need to be more restrictive about the quality of the matches, as in relocalization or map point triangulation.

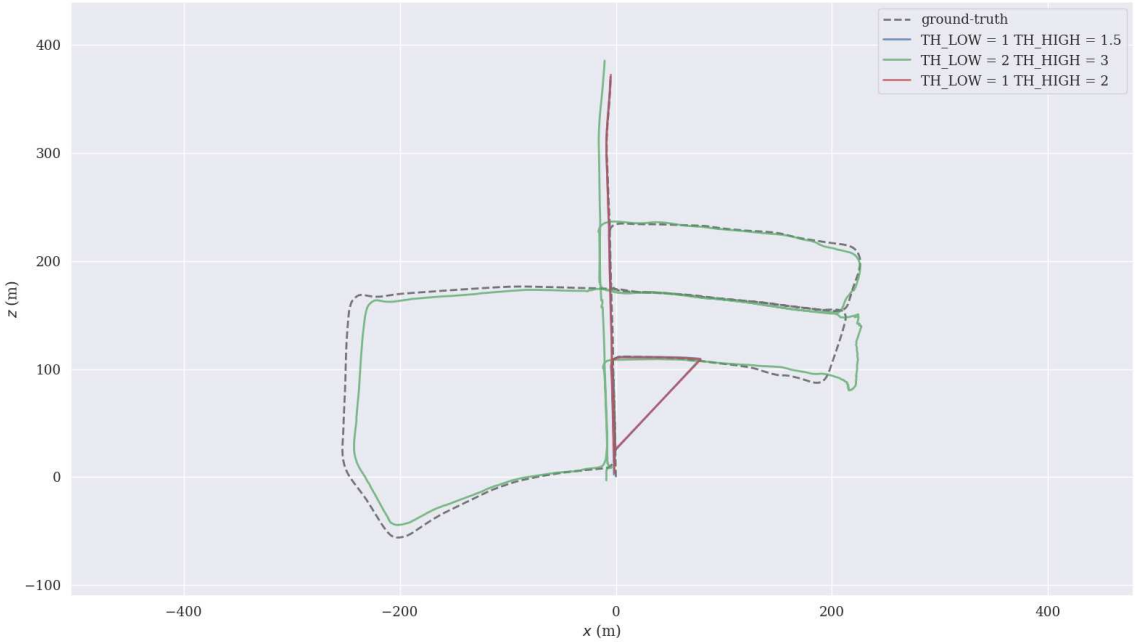
However, while performing our experiments, we found out that for different datasets, the best values for these thresholds could change, as shown in table 5.1 and figure 5.8. Therefore, we had to change the thresholds every time we needed to change the dataset. This is not desirable since, in real-world applications, it is not possible to deduce the values of these thresholds. Hence, we have developed an adaptive method that decides the thresholds values online, based on the number of outliers of the current frame and the number of map points on the last frame.

Threshold		**ATE (m)**	
TH_{LOW}	TH_{HIGH}	MH_01	KITTI 05
1.0	2.0	0.0522	-
1.0	1.5	0.0488	-
2.0	3.0	0.6288	12.6084

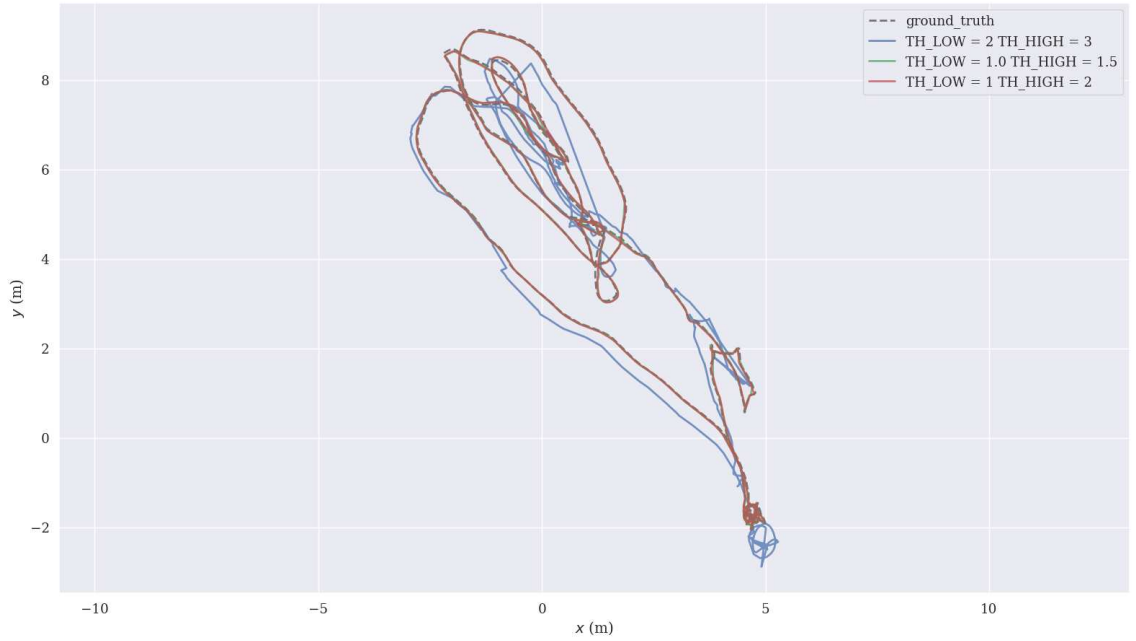
Table 5.1: Absolute Trajectory Error for different matching thresholds in KITTI and EuRoC datasets. In EuRoC MH_01 sequence, the error is completely different for different thresholds, where the best thresholds are $TH_{LOW} = 1.0$ and $TH_{HIGH} = 1.5$. Furthermore, in KITTI 05 sequence, the algorithm could track the camera pose only with $TH_{LOW} = 2.0$ and $TH_{HIGH} = 3.0$. The 2D trajectories for these results are illustrated in figure 5.8.

After estimating the pose with the constant velocity model, we search map point correspondences by projecting the map points from the last frame into the current frame. If the number of outliers approaches the number of map points, the number of matches gets too small and, consequently, the tracking is lost. We use this fact to create our adaptive method. It changes the thresholds values based on the distance from the number of map points and the number of outliers. The adaptive method is performed with algorithm 1. Figure 5.9 depicts the variation of the threshold based on the number of map points used to match and the number of outliers after performing the matching.

In algorithm 1, we define the following parameters empirically. $reference$ is defined as 70% of the number of map points. Ideally, if we keep the number of outliers equal to this value, the thresholds are not updated. The α constant controls the thresholds curves' smoothness, which means that the rate at which the threshold's value changes depends on this parameter. The $minThreshold$ and $maxThreshold$ parameters create a boundary for the threshold values and are set to 1 and 10, respectively. Therefore, in this version, we use this adaptive method to update the matching thresholds while performing VO, we name this version as Adaptive LIFT-SLAM.



(a) Comparison of LIFT-SLAM performance with different matching threholds in KITTI 05 sequence.



(b) Comparison of LIFT-SLAM performance with different matching threholds in EuRoC MH_01 sequence.

Figure 5.8: The problem in matching thresholds, for different thresholds the performance of LIFT-SLAM is completely different on the same sequences. Moreover, the most suitable thresholds for the KITTI sequence (figure 5.8a) are $TH_{LOW} = 2.0$ and $TH_{HIGH} = 3.0$. However, for the EuRoC sequence 5.8b, the best thresholds are $TH_{LOW} = 1.0$ and $TH_{HIGH} = 1.5$.

Algorithm 1: Update the thresholds

Input: Number of map points ($nMapPoints$) and number of outliers ($nOutliers$)
Output: Updated Thresholds
$reference \leftarrow 0.7 * nMapPoints$;
$distance \leftarrow nOutliers - reference$;
$\alpha \leftarrow 0.005$;
if ($distance > 0$ ***and*** $TH_{HIGH} < maxThreshold$)
or ($distance < 0$ ***and*** $TH_{LOW} > minThreshold$) **then**
  $TH_{HIGH} \leftarrow TH_{HIGH} + \alpha * distance$;
  $TH_{LOW} \leftarrow TH_{LOW} + \alpha * distance$;
end

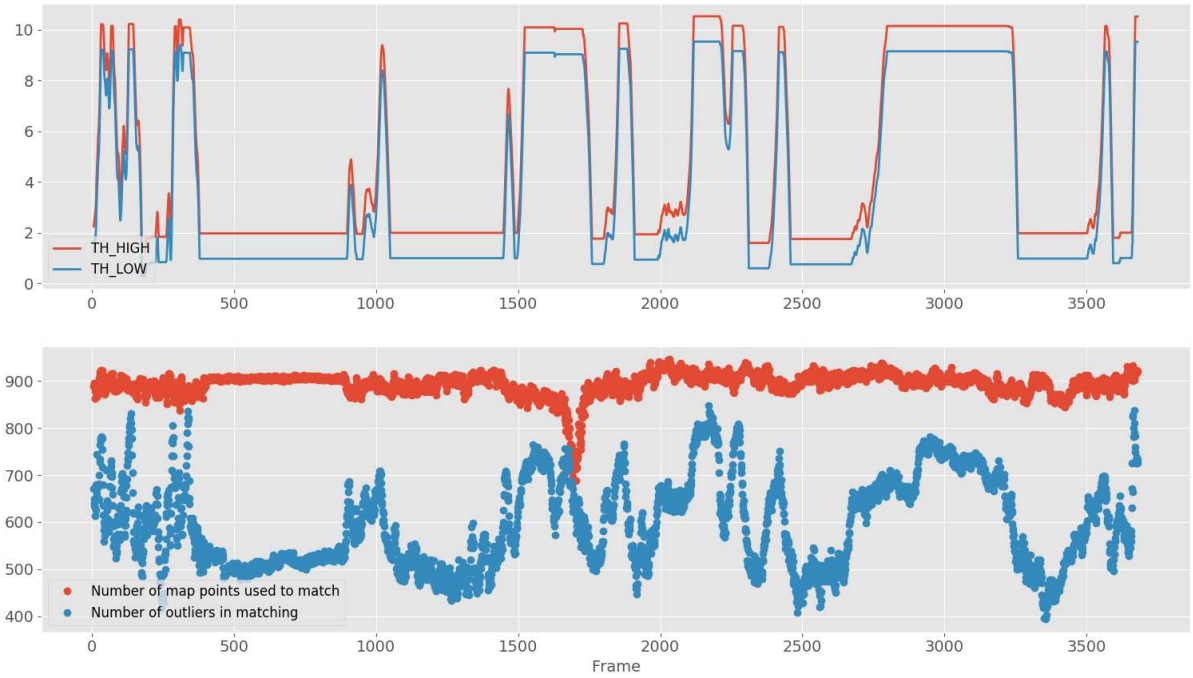


Figure 5.9: Thresholds variation based on the number of map points used to match and the number of outliers after performing the matching. The thresholds values decrease as the number of outliers reduce.

5.3 Final Considerations

This chapter presented a brief qualitative analysis we made to chose our feature descriptor. This analysis showed that LIFT is more robust than ORB under different scenarios. Later, we describe in section 5.1 the pipeline of LIFT-SLAM. Then, in section 5.2, we proposed some versions of LIFT-SLAM to improve the algorithm's performance. The fine-tuned LIFT-SLAM (section 5.2.1) uses a version of the LIFT network fine-tuned with VO/VSLAM datasets. The adaptive LIFT-SLAM (section 5.2.2), in its turn, changes the matching thresholds during the algorithm's execution based on the number of outliers and the number of map points.

Chapter 6

Results

In this chapter, we show how we performed our experiments and collected the results of the algorithms. We also present discussions about these results.

We evaluate the robustness of our algorithm in different environments and situations using KITTI and EuRoC datasets, as described in chapter 4. For each sequence, we generate a quantitative and qualitative comparison between the trajectories estimated with VSLAM algorithms and the ground-truth data. The quantitative evaluation is based on the metrics presented in chapter 4. Results in KITTI sequences are evaluated based on Relative Pose Error (RPE) and Absolute Trajectory Error (ATE), while ATE only assesses the estimates on EuRoC sequences. The ORB-SLAM's results shown here were computed by our executions since, in ORB-SLAM's paper, an evaluation with RPE is not presented. Moreover, they do not provide results in the EuRoC dataset. Furthermore, we present qualitative comparisons showing a 2-D plot of the trajectories. None of the algorithms presented here could estimate odometry in sequence KITTI 01. Therefore, we do not show the results for this case.

Each of the following sections presents a version of LIFT-SLAM, with all possible combinations: LIFT-SLAM, LIFT-SLAM fine-tuned with KITTI sequences, LIFT-SLAM fine-tuned with EuRoC sequences, Adaptive LIFT-SLAM, Adaptive LIFT-SLAM fine-tuned with KITTI sequences, and Adaptive LIFT-SLAM fine-tuned with EuRoC sequences. Moreover, in sequences where the algorithm is not capable of tracking at least 50% of the camera poses we fill the table with a "-." Furthermore, we avoid comparing results in which the sequence was used to fine-tune the network: KITTI sequences 00, 06, 09, and 10, or EuRoC sequences MH_04, V1_03, and V2_03. For the algorithms that are not adaptive, we set the values of $TH_{LOW} = 2$ and $TH_{HIGH} = 3$ for KITTI sequences, and $TH_{LOW} = 1$ and $TH_{HIGH} = 2$ for EuRoC sequences. We also generate a visual vocabulary for each version of LIFT.

Due to the algorithms' stochastic nature, all of the quantitative metrics are an average of 5 executions. Moreover, we run a statistical paired t-test to evaluate with 95% confidence if the quantitative differences between the two algorithms are significant. We fill the *paired t-test* tables as following:

- "+": if the proposed algorithm has improved the performance in the given metric with statistical significance;

- "-": if the proposed algorithm has decreased the performance in the given metric with statistical significance;
- "=": if the proposed algorithm has the same performance in the given metric.

6.1 LIFT-SLAM

First, we compare the results of LIFT-SLAM (without any fine-tuning in the dataset) with ORB-SLAM. Table 6.1 shows the quantitative results of these approaches for the KITTI dataset. Both algorithms could not complete the sequence 02. Our approach has reduced the average error in at least one metric in all of the sequences. Paired t-test also shows that we have improved ORB-SLAM's results with statistical significance in most of the sequences.

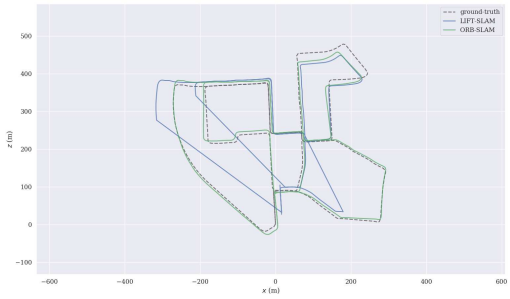
Qualitative results also shows that LIFT-SLAM performed better, specially in smaller sequences, such as 03 (figure 6.1b), 04 (figure 6.1c) and 06 (figure 6.1e). On the other side, in sequence 00, LIFT-SLAM lost track of the pose multiple times whereas ORB-SLAM did not lose track of the pose (figure 6.1a), and although it was capable to perform relocalization, the pose was not accurately estimated.

In some sequences, LIFT-SLAM was capable of closing loops where ORB-SLAM did not, as shown in figures 6.1d (sequence 05), 6.1e (sequence 06), and 6.1f (sequence 07). However, in sequence 08, both algorithms could not close the loop. This sequence is very large, thus, as the drift accumulates over time, if the loop is not found the drift can not be not corrected.

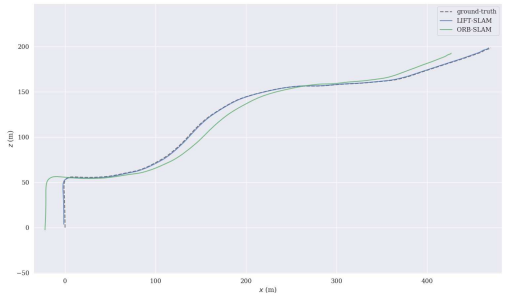
	ORB-SLAM			**LIFT-SLAM**			*paired t-test*		
Seq.	**RPE_{trans} (%)**	**RPE_{rot} (deg/m)**	**ATE (m)**	**RPE_{trans} (%)**	**RPE_{rot} (deg/m)**	**ATE (m)**	**RPE_{trans} (%)**	**RPE_{rot} (deg/m)**	**ATE (m)**
00	**4.4649**	3.2783	**11.5437**	6.7059	**2.1979**	18.767	-	+	-
02	-	-	-	-	-	-	-	-	-
03	9.7559	2.7803	15.1319	**0.8748**	**0.3419**	**1.1046**	+	+	+
04	3.7082	2.1542	4.2917	**2.1023**	**0.6475**	**0.3969**	+	+	+
05	**3.3486**	3.569	**7.7451**	4.4642	**2.5784**	8.0865	=	=	=
06	8.1155	2.8829	20.2568	**7.7577**	**2.4952**	**18.4675**	=	=	=
07	7.4283	**3.5759**	13.4714	**2.5092**	3.5973	**4.0343**	+	=	+
08	**12.1617**	3.0528	**39.5121**	27.6288	**2.1047**	80.9727	-	+	-
09	26.5071	11.1348	**49.6687**	**20.6534**	**2.1185**	59.8848	=	=	=
10	**8.6547**	3.6241	**19.9418**	10.0807	**2.2551**	31.8443	=	+	-

Table 6.1: Comparison of results between ORB-SLAM (our execution) and LIFT-SLAM in KITTI Dataset, the smaller average errors are highlighted. The *paired t-test* shows if LIFT-SLAM has improved the performance of ORB-SLAM with statistical significance.

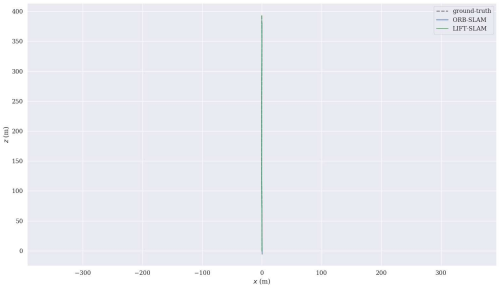
In EuRoC sequences, ORB-SLAM has a smaller ATE average in all sequences, as shown in table 6.2. In a qualitative analysis, LIFT-SLAM has performed well in the easiest sequences, as illustrated in figures 6.2a and 6.2b. However, it lost track easily and could not estimate motion in sequence MH_03 (figure 6.2c). Moreover, it could not even track the pose in MH_04, and in $V1$ and $V2$ sequences. Hence, we tried to improve the performance of the algorithm by fine-tuning the LIFT network with specific VO datasets, as explained in chapter 5.



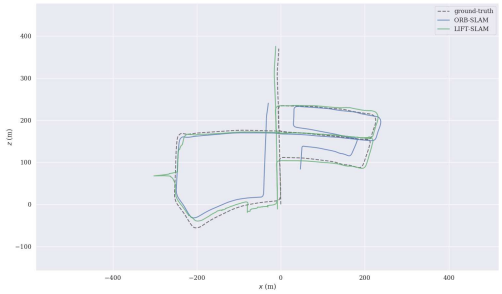
(a) KITTI 00



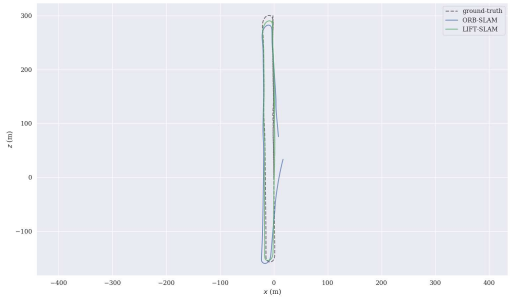
(b) KITTI 03



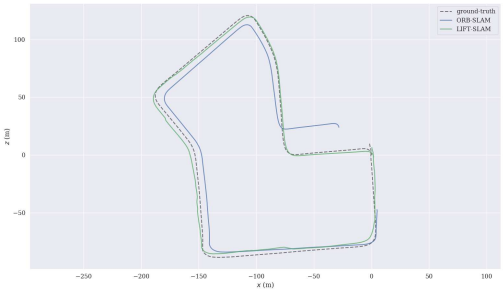
(c) KITTI 04



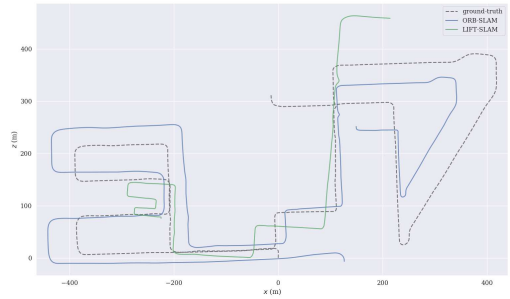
(d) KITTI 05



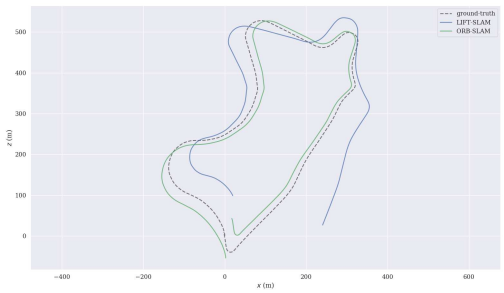
(e) KITTI 06



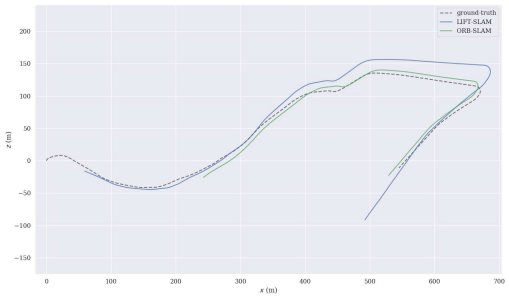
(f) KITTI 07



(g) KITTI 08



(h) KITTI 09

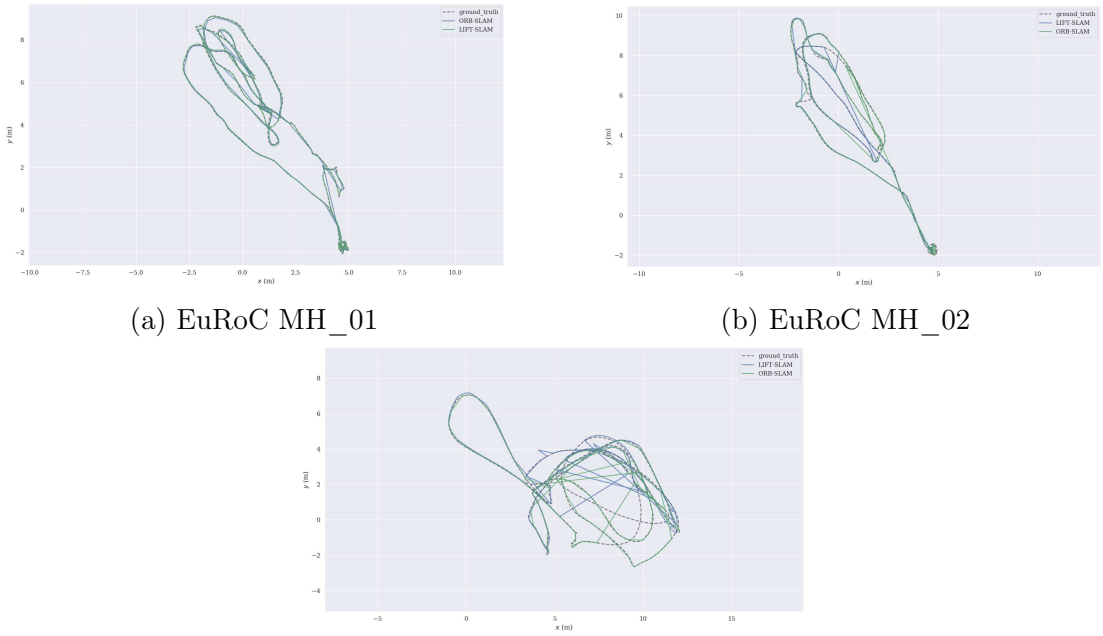


(i) KITTI 10

Figure 6.1: 2D trajectories comparison between LIFT-SLAM, ORB-SLAM and ground-truth in KITTI Dataset.

	ORB-SLAM	**LIFT-SLAM**	*paired t-test*
Seq.	**ATE (m)**	**ATE (m)**	**ATE (m)**
MH_01	**0.0481**	0.0620	=
MH_02	**0.0369**	0.2270	=
MH_03	**0.0404**	0.1445	=
MH_04	**0.4318**	-	-
V1_01	**0.0998**	-	-
V1_03	**0.3703**	-	-
V2_03	**0.2499**	-	-

Table 6.2: Comparison of results between ORB-SLAM and LIFT-SLAM in EuRoC Dataset.



(a) EuRoC MH_01

(b) EuRoC MH_02

(c) EuRoC MH_03

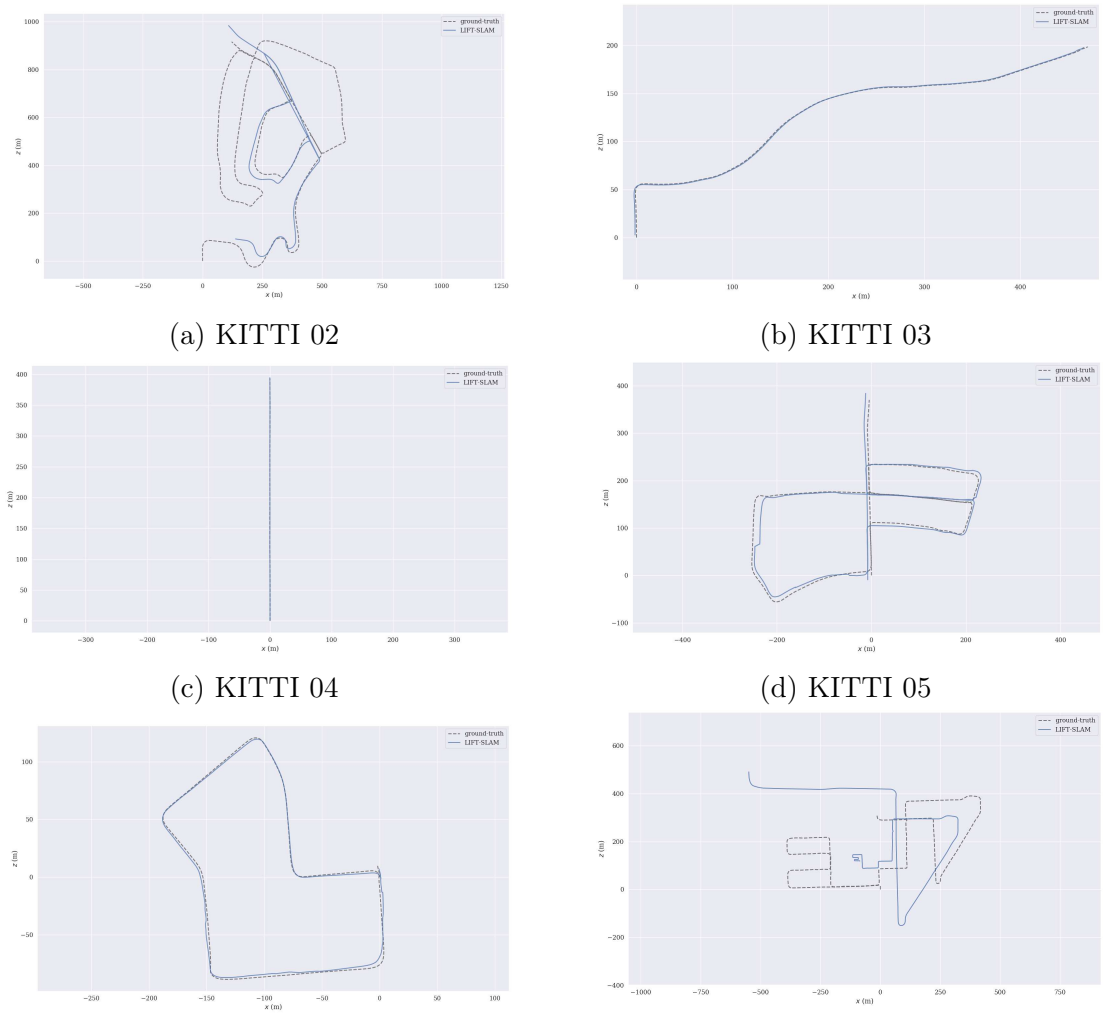
Figure 6.2: 2D trajectories comparison between LIFT-SLAM, ORB-SLAM and ground-truth in EuRoC Dataset.

6.2 LIFT-SLAM fine-tuned with KITTI sequences

The first fine-tuning we performed was using KITTI sequences. The quantitative results of LIFT-SLAM fine-tuned with KITTI sequences are presented in tables 6.3 (KITTI dataset) and 6.4 (EuRoC dataset). As we are looking for improvements in LIFT-SLAM, we compare the results of the fine-tuned networks with it. To avoid showing biased results, we do not evaluate the results of this algorithm in the sequences it was fine-tuned. The algorithm was now capable of tracking pose in a considerable part of the sequence 02. However, it still could not complete the entire sequence, as shown in figure 6.3a.

	LIFT-SLAM Fine-tuned with KITTI			*paired t-test*		
Seq.	**RPE_{trans} (%)**	**RPE_{rot} (%)**	**ATE (m)**	**RPE_{trans} (%)**	**RPE_{rot} (deg/m)**	**ATE (m)**
02	**8.7997**	**2.1088**	**29.8347**	-	-	-
03	1.3165	0.3420	1.9082	-	=	-
04	2.1633	**0.5167**	**0.3622**	=	+	=
05	5.0190	**2.4299**	12.4703	=	=	-
07	**1.7974**	**2.6684**	**2.5431**	=	=	=
08	48.8964	2.1098	188.5068	-	=	-

Table 6.3: Results of LIFT-SLAM fine-tuned with KITTI sequences in KITTI dataset. We highlight results in which this algorithm has a smaller average error than LIFT-SLAM. The *paired t-test* shows if LIFT-SLAM fine-tuned with KITTI sequences has improved the performance of LIFT-SLAM with statistical significance.



(a) KITTI 02

(b) KITTI 03

(c) KITTI 04

(d) KITTI 05

(e) KITTI 07

(f) KITTI 08

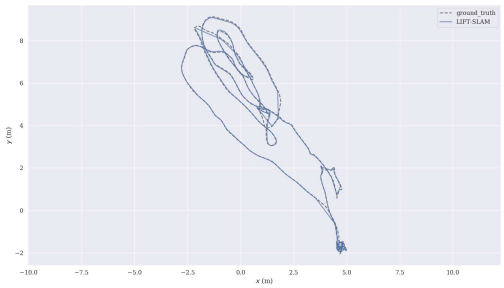
Figure 6.3: 2D trajectories comparison between LIFT-SLAM fine-tuned with KITTI sequences and ground-truth in KITTI Dataset.

The performance of the algorithm remains the same in most cases in the KITTI dataset. However, in the EuRoC dataset, the average ATE has decreased, as shown in table 6.4. In a qualitative analysis, it is possible to notice in figure 6.4 that the algorithm is now more robust to the challenging movements performed in EuRoC dataset. Therefore, even that we use KITTI sequences to fine-tune the network, we could improve

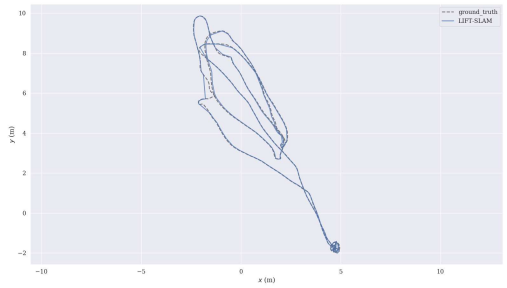
the algorithm's performance in a completely different dataset. This result shows that with transfer learning, LIFT could learn features that are important in VO tasks. Despite this improvement, the algorithm is still not able to perform VO in sequences $V1\_03$ and $V2\_03$.

	LIFT-SLAM Fine-tuned with KITTI	*paired t-test*
Seq.	**ATE (m)**	**ATE (m)**
MH_01	0.1146	=
MH_02	**0.0421**	=
MH_03	**0.0552**	=
MH_04	**0.1170**	-
V1_01	**0.1169**	-
V1_03	-	-
V2_03	-	-

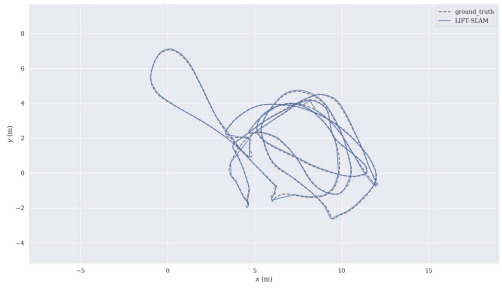
Table 6.4: Results of LIFT-SLAM fine-tuned with KITTI sequences in EuRoC dataset.



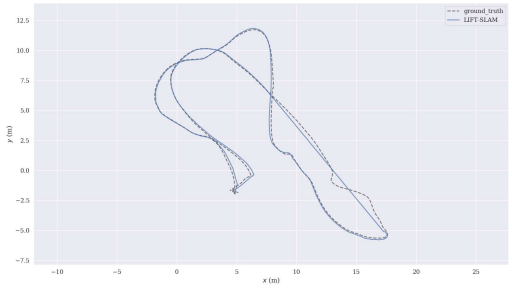
(a) EuRoC MH_01



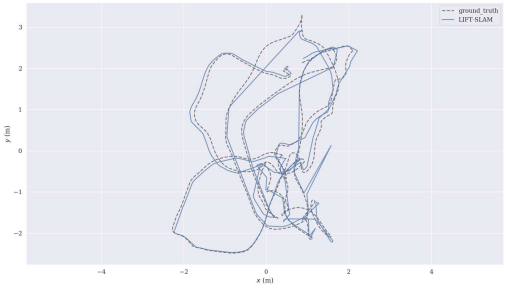
(b) EuRoC MH_02



(c) EuRoC MH_03



(d) EuRoC MH_04



(e) EuRoC V1_01

Figure 6.4: 2D trajectories comparison between LIFT-SLAM fine-tuned with KITTI sequences and ground-truth in EuRoC Dataset.

6.3 LIFT-SLAM fine-tuned with EuRoC sequences

EuRoC sequences contain much more rotations and acceleration than the KITTI dataset. We tried to take advantage of this fact to fine-tune the network with these sequences. Table 6.5 shows the quantitative results of this algorithm in the KITTI dataset. We compare the results of this approach with LIFT-SLAM. We can see that LIFT-SLAM fine-tuned with EuRoC improved results of LIFT-SLAM in some sequences. Therefore, as the algorithm's performance has improved in both datasets, we can affirm that the transfer learning allowed the network to learn important features for VO. Furthermore, qualitative results in the sequence 00 (figure 6.5a) show that the algorithm is losing fewer frames than it was before.

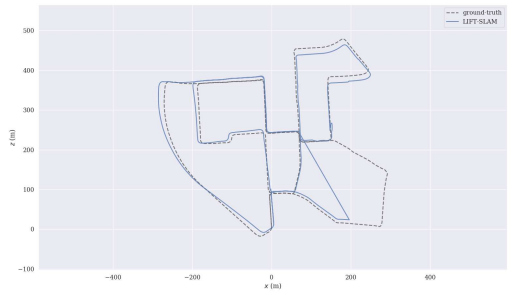
	LIFT-SLAM Fine-tuned with EuRoC			*paired t-test*		
Seq.	**RPE_{trans} (%)**	**RPE_{rot} (deg/m)**	**ATE (m)**	**RPE_{trans} (%)**	**RPE_{rot} (deg/m)**	**ATE (m)**
00	**3.4918**	2.6289	**9.8368**	+	-	+
02	**9.8424**	**2.1050**	**34.2339**	-	-	-
03	**0.8579**	0.4624	**0.9746**	=	-	=
04	2.2200	**0.5014**	0.4229	-	+	=
05	5.3541	**1.9126**	11.5047	=	=	=
06	**7.0473**	**2.3560**	**16.5814**	+	+	+
07	2.6048	3.6421	**3.9835**	=	=	=
08	28.9926	**1.9548**	82.6137	=	=	=
09	19.1622	2.0792	54.9154	=	=	=
10	**9.809**	**2.2044**	**30.3435**	=	=	=

Table 6.5: Results of LIFT-SLAM fine-tuned with EuRoC sequences in KITTI dataset. We highlight results in which this algorithm has a smaller average error than LIFT-SLAM. The paired t-test shows if LIFT-SLAM fine-tuned with EuRoC sequences has improved the performance of LIFT-SLAM with statistical significance.

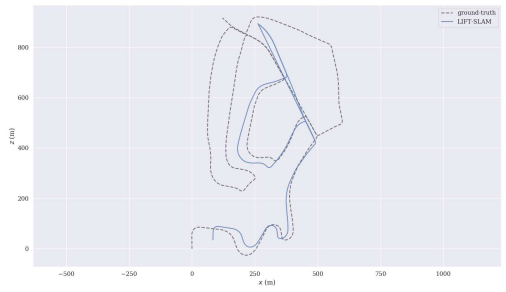
Table 6.6 shows the results of the algorithm in EuRoC sequences. As aforementioned, we do not evaluate the results of this algorithm in the sequences it was fine-tuned. In general, LIFT-SLAM fine-tuned with EuRoC sequences has presented a smaller average error than LIFT-SLAM. Qualitative results in figure 6.6 shows that, in comparison with LIFT-SLAM, the algorithm performed better for more challenging sequences, as it loses track of the pose less often.

	LIFT-SLAM Fine-tuned with EuRoC	*paired t-test*
Seq.	**ATE (m)**	**ATE (m)**
MH_01	0.1170	=
MH_02	**0.0621**	=
MH_03	**0.0528**	=
V1_01	**0.1499**	=

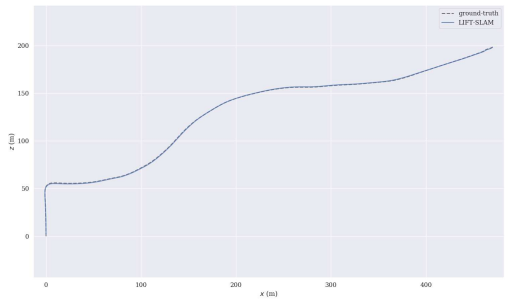
Table 6.6: Results of LIFT-SLAM fine-tuned with EuRoC sequences in EuRoC Dataset.



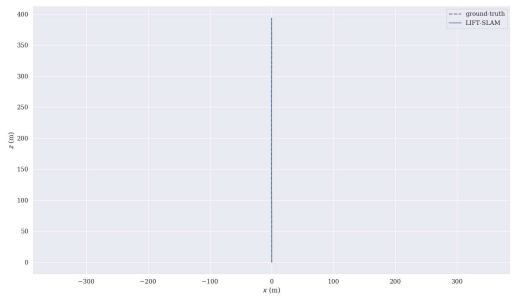
(a) KITTI 00



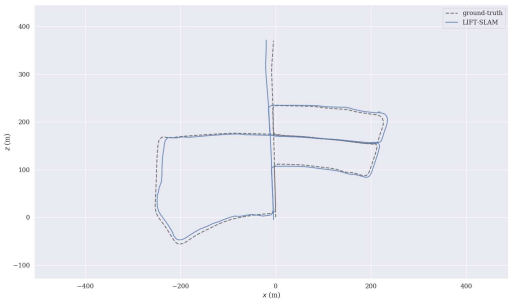
(b) KITTI 02



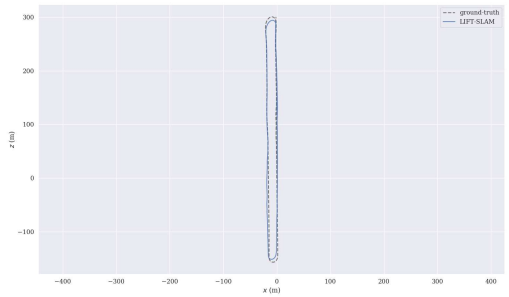
(c) KITTI 03



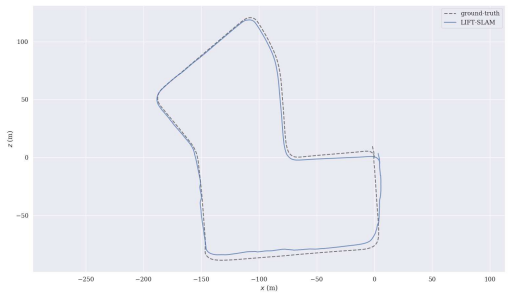
(d) KITTI 04



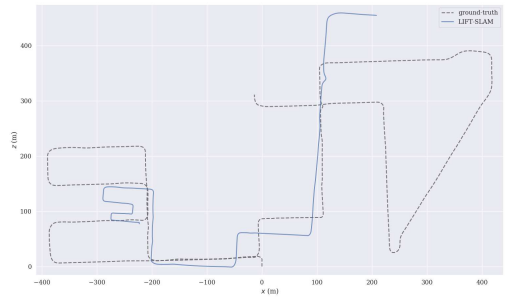
(e) KITTI 05



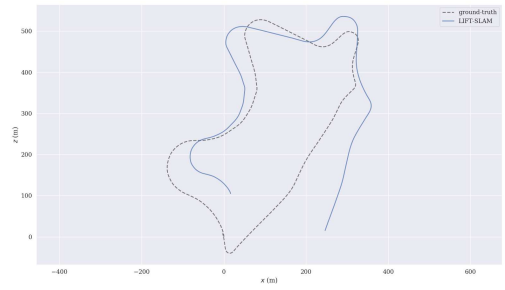
(f) KITTI 06



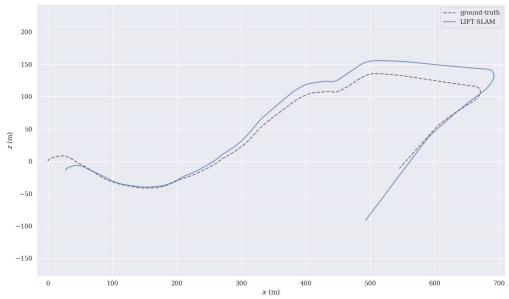
(g) KITTI 07



(h) KITTI 08

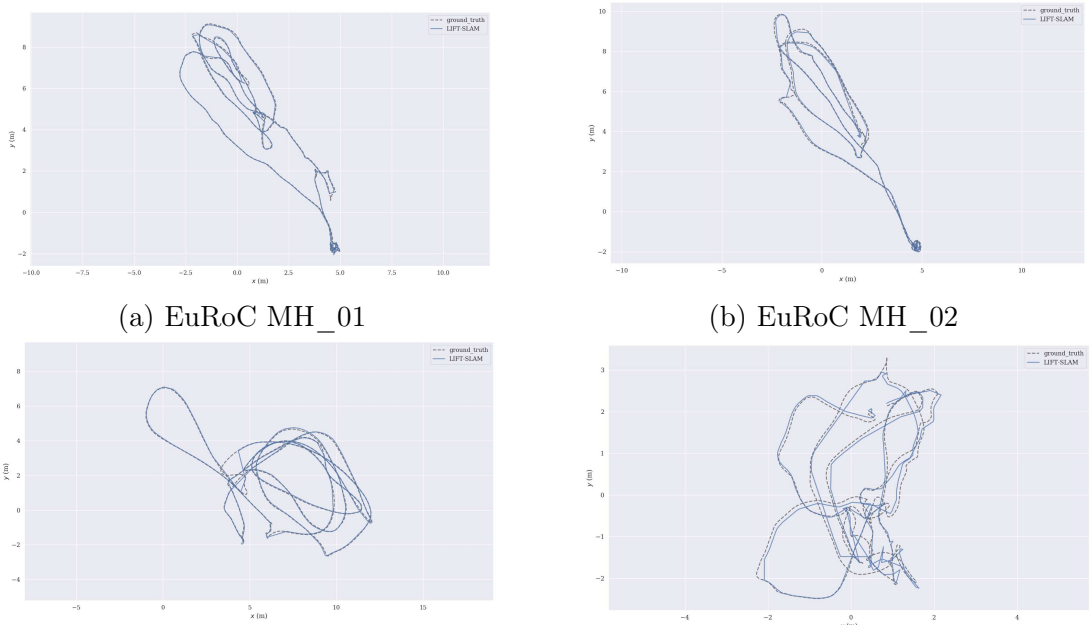


(i) KITTI 09



(j) KITTI 10

Figure 6.5: 2D trajectories comparison between LIFT-SLAM fine-tuned with EuRoC sequences and ground-truth in KITTI Dataset.



(a) EuRoC MH_01 (b) EuRoC MH_02

(c) EuRoC MH_03 (d) EuRoC V1_01

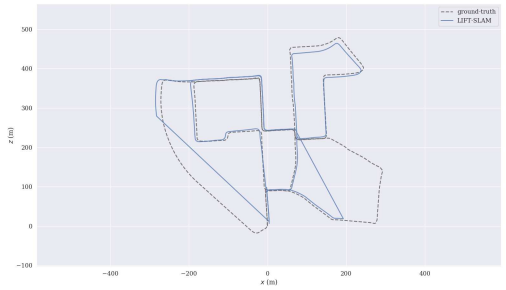
Figure 6.6: 2D trajectories comparison between LIFT-SLAM fine-tuned with EuRoC sequences and ground-truth in EuRoC Dataset.

6.4 Adaptive LIFT-SLAM

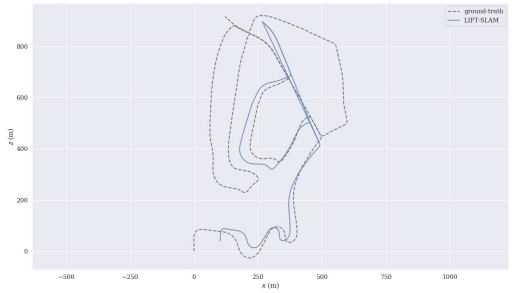
Another version of LIFT-SLAM we created is Adaptive LIFT-SLAM, as described in chapter 5. In this version, we change the matching thresholds while performing VO/VSLAM. We compare the results of this version with LIFT-SLAM. Table 6.7 shows the quantitative results of this algorithm. We can notice that this approach has improved the results in most sequences. Furthermore, in qualitative results in KITTI dataset (Figure 6.7), we can see that this algorithm is capable of tracking the pose in a considerable part of the sequence 02.

	Adaptive LIFT-SLAM			*paired t-test*		
Seq.	**RPE_{trans} (%)**	**RPE_{rot} (deg/m)**	**ATE (m)**	**RPE_{trans} (%)**	**RPE_{rot} (deg/m)**	**ATE (m)**
00	**2.6445**	4.9554	**13.6972**	+	=	-
02	**11.5396**	**2.2221**	**40.3285**	-	-	-
03	**0.7825**	0.3788	**0.8435**	=	=	+
04	2.2185	**0.5973**	0.4760	-	=	=
05	5.4926	2.9657	10.8523	=	=	=
06	**7.5031**	**2.4211**	**17.8257**	=	+	=
07	2.6724	**3.4200**	4.0863	=	=	=
08	28.4933	**2.0484**	81.6931	=	=	=
09	**19.2791**	2.1731	**57.7414**	+	-	+
10	**4.9604**	**1.5718**	**10.5114**	+	+	+

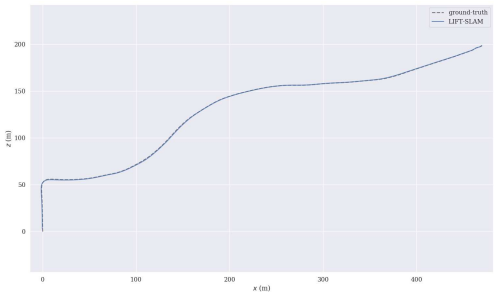
Table 6.7: Results of Adaptive LIFT-SLAM in KITTI dataset. We highlight results in which this algorithm has a smaller average error than LIFT-SLAM. The *paired t-test* shows if Adaptive LIFT-SLAM has improved the performance of LIFT-SLAM with statistical significance.



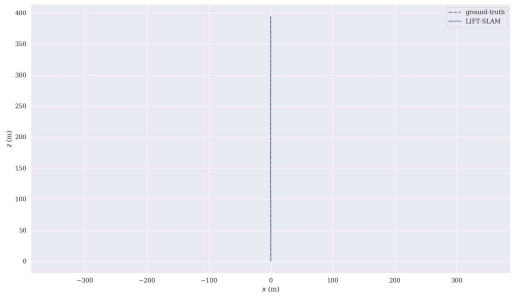
(a) KITTI 00



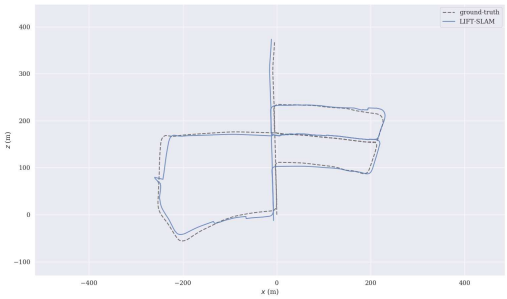
(b) KITTI 02



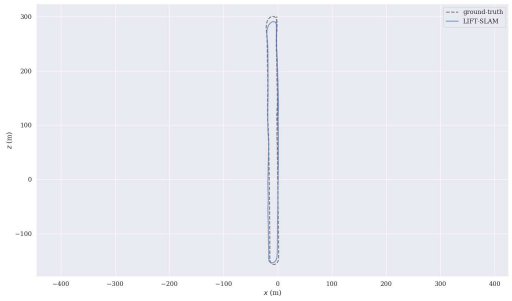
(c) KITTI 03



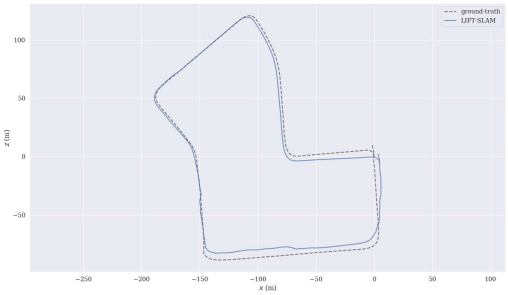
(d) KITTI 04



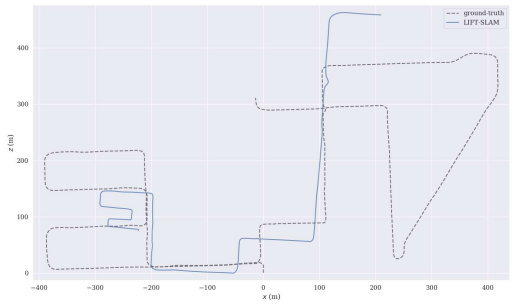
(e) KITTI 05



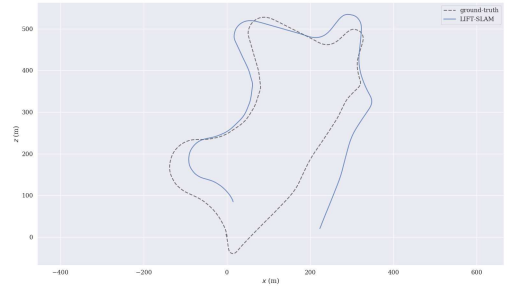
(f) KITTI 06



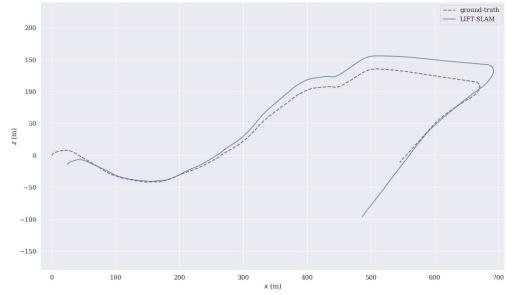
(g) KITTI 07



(h) KITTI 08



(i) KITTI 09



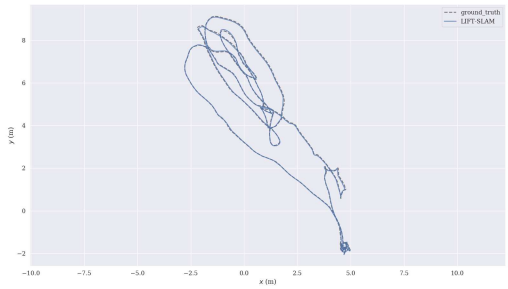
(j) KITTI 10

Figure 6.7: 2D trajectories comparison between Adaptive LIFT-SLAM and ground-truth in KITTI Dataset.

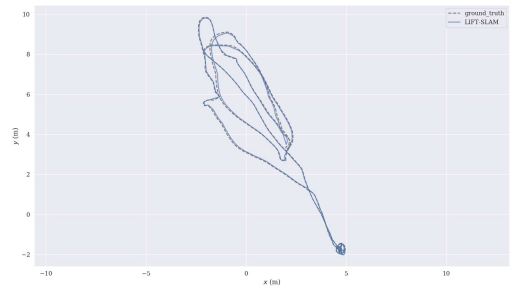
In EuRoC sequences, this algorithm could reduce the average error for easy sequences (MH_01, MH_02, and V1_01), as shown in table 6.8. However, the algorithm could not track the pose for the other sequences. The qualitative results for this algorithm in the EuRoC dataset are shown in table 6.8. Then, we decided to combine the adaptive approach with the fine-tuned networks.

	Adaptive LIFT-SLAM	*paired t-test*
Seq.	**ATE (m)**	**ATE (m)**
MH_01	**0.0465**	=
MH_02	**0.0343**	=
MH_03	-	-
MH_04	-	-
V1_01	**0.1006**	=
V1_03	-	-
V2_03	-	-

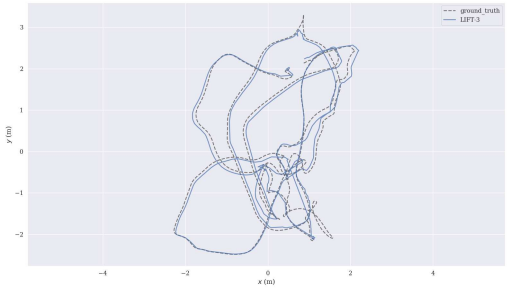
Table 6.8: Results of Adaptive LIFT-SLAM in EuRoC dataset.



(a) EuRoC MH_01



(b) EuRoC MH_02



(c) EuRoC V1_01

Figure 6.8: 2D trajectories comparison between Adaptive LIFT-SLAM and ground-truth in EuRoC Dataset.

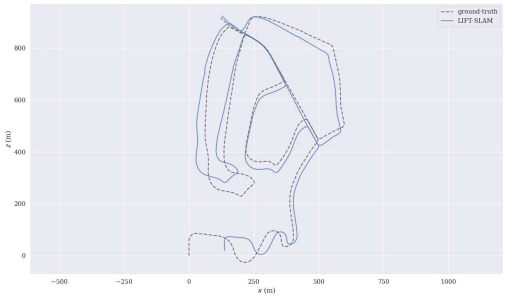
6.5 Adaptive LIFT-SLAM fine-tuned with KITTI sequences

In this approach, we combine the adaptive method with LIFT-SLAM fine-tuned with KITTI sequences. Therefore, we compare this algorithm with the LIFT-SLAM fine-tuned with KITTI sequences to check the improvements caused by this algorithm's adaptive method. Table 6.9 shows the quantitative results of this approach in the KITTI dataset.

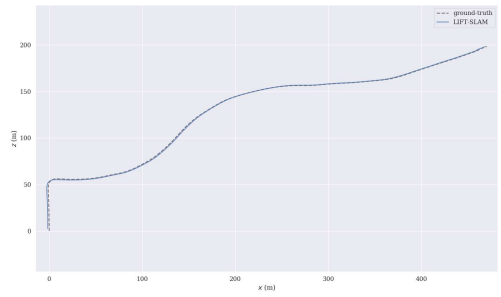
We can see that in general, the results of this approach and LIFT-SLAM fine-tuned with KITTI dataset are statistically the same. The qualitative results for this approach are shown in figure 6.9. In these results, we can see that this approach could track the entire sequence of 02 (figure 6.9a), as opposed to previous approaches.

	Adaptive LIFT-SLAM Fine-tuned with KITTI			*paired t-test*		
Seq.	**RPE_{trans} (%)**	**RPE_{rot} (deg/m)**	**ATE (m)**	**RPE_{trans} (%)**	**RPE_{rot} (deg/m)**	**ATE (m)**
02	9.5702	2.4325	48.0893	=	=	=
03	**1.2929**	**0.3375**	1.9095	=	=	=
04	**2.1190**	0.5727	0.4237	=	=	=
05	**4.6392**	2.9258	**10.3497**	=	=	=
07	2.6377	3.5143	4.1007	=	=	=
08	**47.1989**	**1.9999**	**185.147**	=	+	=

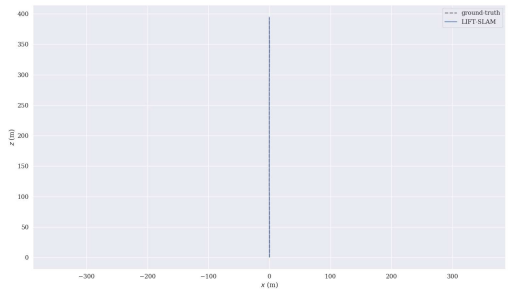
Table 6.9: Results of Adaptive LIFT-SLAM fine-tuned with KITTI sequences in KITTI dataset. We highlight results in which this algorithm has a smaller average error than LIFT-SLAM fine-tuned with KITTI sequences. The paired t-test shows if Adaptive LIFT-SLAM fine-tuned with KITTI sequences has improved the performance of LIFT-SLAM fine-tuned with KITTI sequences.



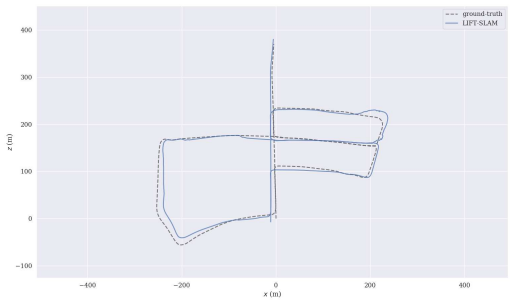
(a) KITTI 02



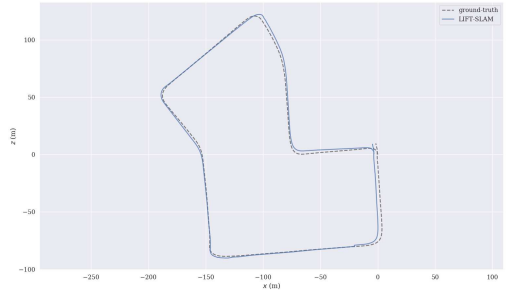
(b) KITTI 03



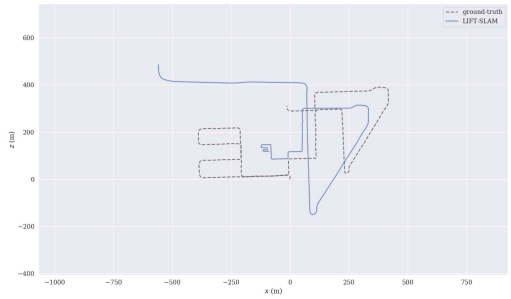
(c) KITTI 04



(d) KITTI 05



(e) KITTI 07



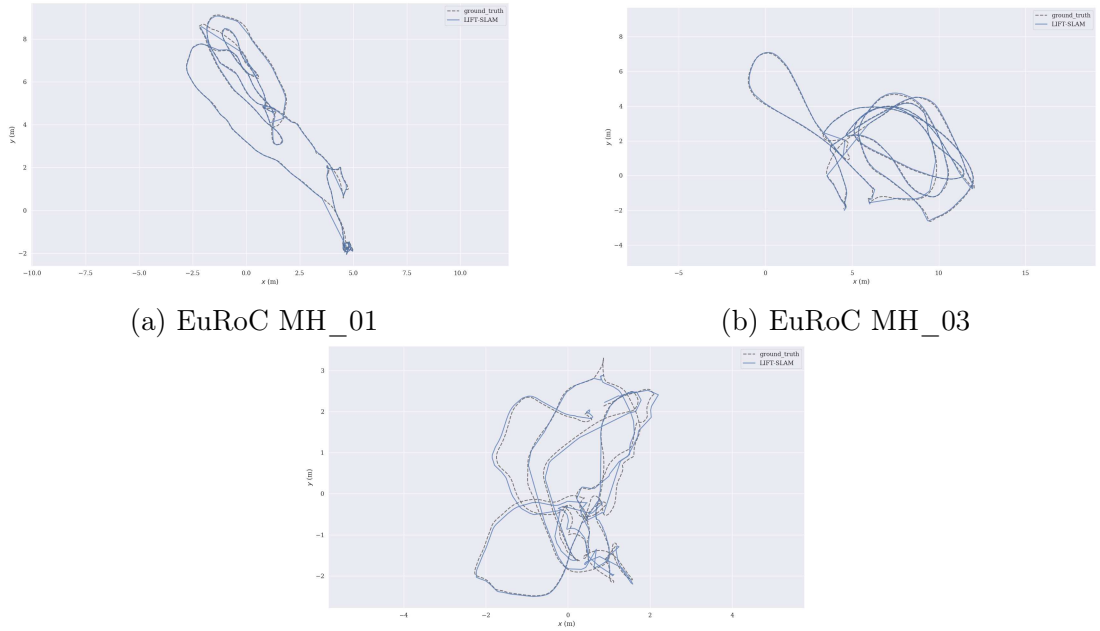
(f) KITTI 08

Figure 6.9: 2D trajectories comparison between Adaptive LIFT-SLAM fine-tuned with KITTI sequences and ground-truth in KITTI Dataset.

Moreover, table 6.10 shows the quantitative results for this approach in EuRoC dataset. This approach did not lead to any improvements in this dataset. Furthermore, it could not track the camera pose in four sequences. Qualitative results are shown in figure 6.10.

	Adaptive LIFT-SLAM Fine-tuned with KITTI	*paired t-test*
Seq.	**ATE (m)**	**ATE (m)**
MH_01	0.455	=
MH_02	-	-
MH_03	0.1162	=
MH_04	-	-
V1_01	0.1939	=
V1_03	-	-
V2_03	-	-

Table 6.10: Results of Adaptive LIFT-SLAM fine-tuned with KITTI sequences in EuRoC dataset.



(a) EuRoC MH_01

(b) EuRoC MH_03

(c) EuRoC V1_01

Figure 6.10: 2D trajectories comparison between Adaptive LIFT-SLAM fine-tuned with KITTI sequences and ground-truth in EuRoC Dataset.

6.6 Adaptive LIFT-SLAM fine-tuned with EuRoC sequences

Lastly, we apply the adaptive method in LIFT-SLAM fine-tuned with EuRoC sequences. Table 6.11 shows the quantitative results of this algorithm in KITTI sequences. This algorithm has a similar performance compared with LIFT-SLAM fine-tuned with EuRoC sequences. This means that adding the adaptive method into our algorithm did not affect its performance. This can be an advantage since the adaptive method solves the problem mentioned in chapter 5, where the matching thresholds have to be set every time we need to execute the algorithm in another dataset.

Qualitative results also show a great advantage in using this method, it can track camera pose without getting lost in sequences where the other versions of LIFT-SLAM where lost: sequences 00 (figure 6.11a) and 02 (figure 6.11b). However, as in the other algorithms, Adaptive LIFT-SLAM fine-tuned with EuRoC has its worse performance in the sequence 08.

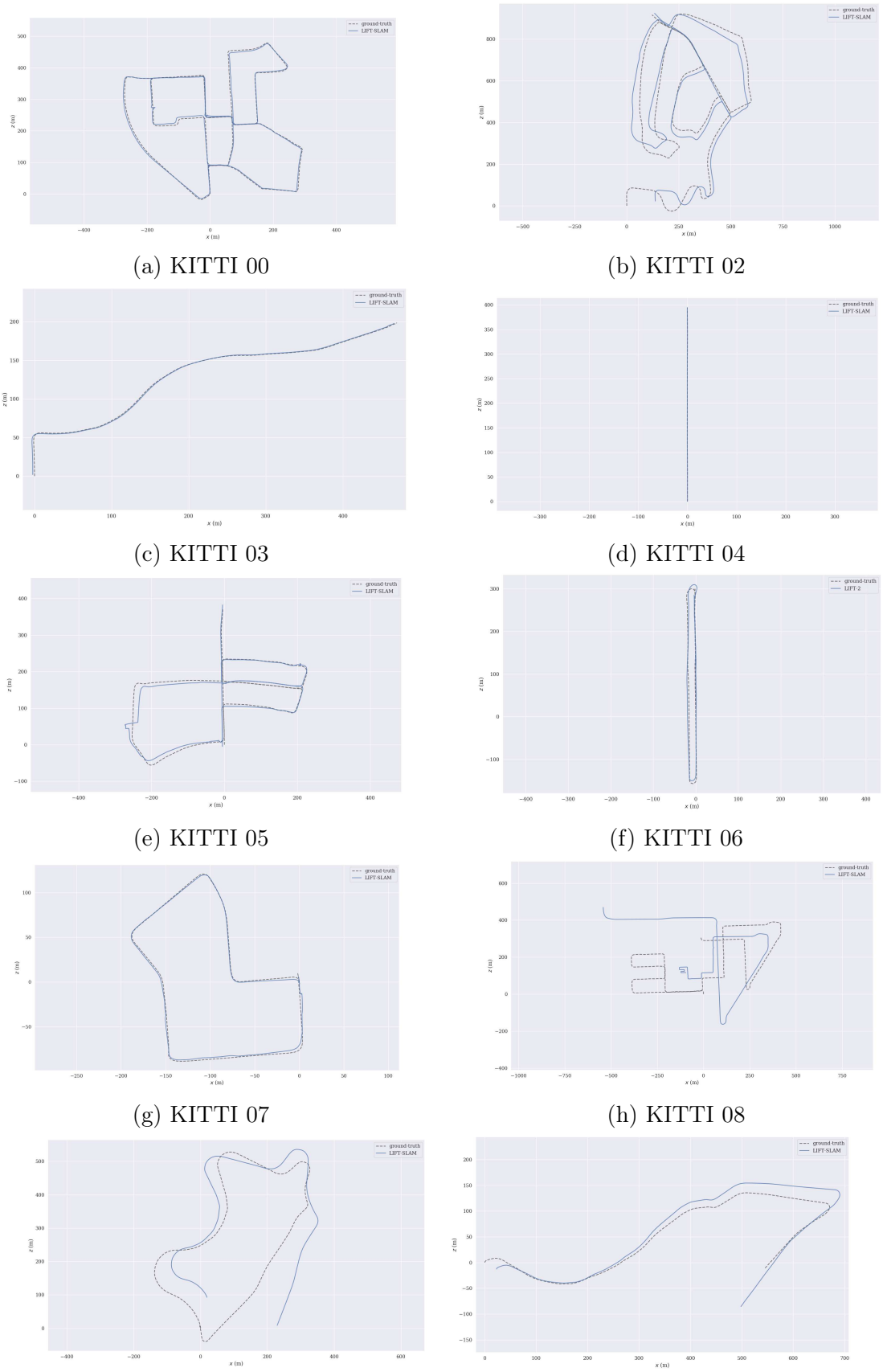
	Adaptive LIFT-SLAM Fine-tuned with EuRoC			*paired t-test*		
Seq.	**RPE_{trans} (%)**	**RPE_{rot} (deg/m)**	**ATE (m)**	**RPE_{trans} (%)**	**RPE_{rot} (deg/m)**	**ATE (m)**
00	**3.1826**	2.9918	**8.0572**	=	-	=
02	**8.7287**	2.4858	40.0375	=	=	=
03	1.4576	**0.3452**	2.2348	-	+	-
04	2.2204	**0.4834**	0.5102	=	=	=
05	6.0932	3.1084	13.5476	=	=	=
06	12.2425	2.9118	30.3805	=	=	=
07	**2.416**	4.0226	**3.6308**	=	=	=
08	47.0996	2.0244	184.4345	-	-	-
09	19.9073	2.1406	59.6246	=	-	=
10	**9.7188**	2.2391	**29.8671**	=	=	=

Table 6.11: Results of Adaptive LIFT-SLAM fine-tuned with EuRoC sequences in KITTI dataset. We highlight results in which this algorithm has a smaller average error than LIFT-SLAM fine-tuned with EuRoC sequences.

Table 6.12 shows the quantitative results of this approach in EuRoC dataset. The average error is smaller than LIFT-SLAM fine-tuned with EuRoC sequences in three sequences, which shows that the adaptive method was capable of improving this approach. The trajectories for EuRoC sequences are shown in figure 6.12.

	Adaptive LIFT-SLAM Fine-tuned with EuRoC	*paired t-test*
Seq.	**ATE (m)**	**ATE (m)**
MH_01	**0.0443**	=
MH_02	**0.0529**	=
MH_03	**0.0487**	=
V1_01	0.1567	=

Table 6.12: Results of Adaptive LIFT-SLAM fine-tuned with EuRoC sequences in EuRoC dataset.



(a) KITTI 00

(b) KITTI 02

(c) KITTI 03

(d) KITTI 04

(e) KITTI 05

(f) KITTI 06

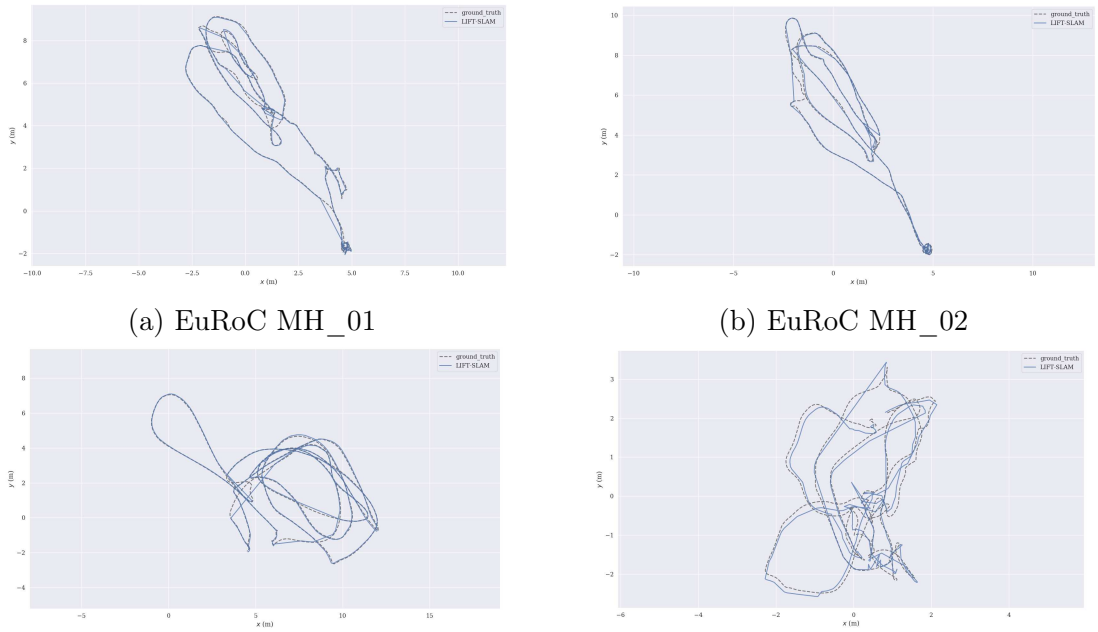
(g) KITTI 07

(h) KITTI 08

(i) KITTI 09

(j) KITTI 10

Figure 6.11: 2D trajectories comparison between Adaptive LIFT-SLAM fine-tuned with EuRoC sequences and ground-truth in KITTI Dataset.



(a) EuRoC MH_01

(b) EuRoC MH_02

(c) EuRoC MH_03

(d) EuRoC V1_01

Figure 6.12: 2D trajectories comparison between Adaptive LIFT-SLAM fine-tuned with EuRoC sequences and ground-truth in EuRoC Dataset.

6.7 Robustness tests

In order to test the robustness of our system to camera sensor noise, we create different image distortion in KITTI sequence 03 simulating camera under and overexposure. These scenarios were emulated with the application of gamma power transformation and quantile-based truncation, as proposed in [77]. More details about these operations are described next:

- Gamma power transformation: as explained in chapter 5, this transformation creates a new image by applying a non-linear operation. We used four values of γ: 0.25, 0.5, 2 and 4. Values of $\gamma < 1$ results in data loss for bright regions emulating camera overexposing and $\gamma > 1$ results data loss for dark regions emulating camera underexposing [77];

- Quantile-based truncation: We have truncated the first (Q_1) and third (Q_3) quantiles of the pixels' intensities distribution to reproduce the effects of low dynamic range imaging sensors. When truncating pixels in Q_1, we emulate sensor underexposing, and in Q_3, we emulate sensor overexposing.

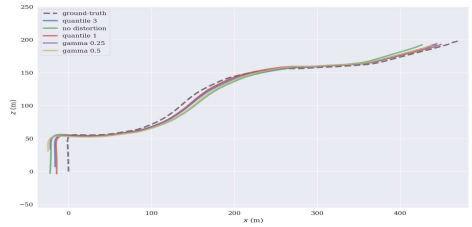
As Adaptive LIFT-SLAM fine-tuned with EuRoC sequences obtained the best overall results, we tested this algorithm under the described scenarios and compared its performance with ORB-SLAM under the same scenarios. The results of these tests are shown in table 6.13.

Sequence	Distortion	ORB-SLAM			LIFT-SLAM		
		RPE_{trans} (%)	RPE_{rot} (deg/m)	ATE (m)	RPE_{trans} (%)	RPE_{rot} (deg/m)	ATE (m)
KITTI 03	no distortion	9.75	2.78	15.13	1.46	0.34	2.23
	$\gamma = 0.25$	7.68	1.95	11.72	1.02	0.40	1.23
	$\gamma = 0.5$	8.25	2.24	11.38	1.28	0.36	1.74
	$\gamma = 2$	X	X	X	1.07	0.51	1.47
	$\gamma = 4$	X	X	X	2.82	0.70	5.23
	Truncation in Q_1	8.34	1.41	13.63	1.36	0.45	2.07
	Truncation in Q_3	9.78	2.23	15.66	1.10	0.46	1.27
KITTI 06	no distortion	8.11	2.88	20.26	12.24	2.91	30.38
	$\gamma = 0.25$	8.97	2.11	21.85	7.94	2.22	19.07
	$\gamma = 0.5$	9.55	2.16	24.11	7.61	2.27	18.19
	$\gamma = 2$	11.69	4.50	27.24	6.91	2.30	16.22
	$\gamma = 4$	11.12	5.71	28.10	8.29	2.68	18.60
	Truncation in Q_1	15.01	4.69	36.52	6.44	2.36	15.09
	Truncation in Q_3	X	X	X	8.09	2.29	19.33
KITTI 07	no distortion	7.43	3.58	13.47	2.42	4.02	3.63
	$\gamma = 0.25$	7.61	2.42	12.54	2.09	3.17	3.30
	$\gamma = 0.5$	6.37	2.02	9.58	1.99	3.88	2.86
	$\gamma = 2$	5.89	2.11	9.36	3.43	4.32	5.91
	$\gamma = 4$	6.61	7.06	7.84	8.03	7.40	16.13
	Truncation in Q_1	8.50	2.47	10.69	2.45	4.10	3.58
	Truncation in Q_3	7.01	2.40	7.08	2.69	3.77	4.49
KITTI 10	no distortion	8.65	3.62	19.94	9.72	2.24	29.87
	$\gamma = 0.25$	13.52	3.01	26.56	10.72	2.15	30.77
	$\gamma = 0.5$	12.40	3.95	25.55	10.03	2.24	31.93
	$\gamma = 2$	16.88	4.59	28.07	X	X	X
	$\gamma = 4$	X	X	X	X	X	X
	Truncation in Q_1	20.79	2.95	34.79	X	X	X
	Truncation in Q_3	X	X	X	X	X	X
Euroc MH_02	no distortion	-	-	0.037	-	-	0.053
	$\gamma = 0.25$	-	-	0.055	-	-	0.035
	$\gamma = 0.5$	-	-	0.040	-	-	0.039
	$\gamma = 2$	-	-	0.061	-	-	0.037
	$\gamma = 4$	-	-	0.010	-	-	0.194
	Truncation in Q_1	-	-	0.039	-	-	0.043
	Truncation in Q_3	-	-	0.043	-	-	X

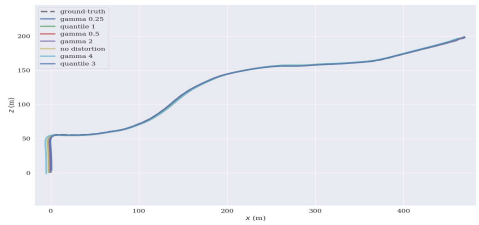
Table 6.13: Results of the robustness tests. The LIFT-SLAM version used in these tests is the adaptive fine-tuned with Euroc sequences. We fill with "X" the sequences unavailable due to tracking failure and with "-" the sequences we do not execute the algorithms.

The results presented in table 6.13 show that ORB-SLAM could not track the pose of the camera with some distortion in sequences KITTI 03, 06, and 10, while LIFT-SLAM failed in some cases for sequences KITTI 10 and Euroc MH_02. However, we can notice that in most of the sequences, LIFT-SLAM improved its performance when we applied the distortions. This fact occurs because the distortions remove some outliers from the images, which allows the algorithms to select better keypoints. Moreover, the learned features are more robust to ill exposure as the datasets used to fine-tune the network naturally contain varying illumination.

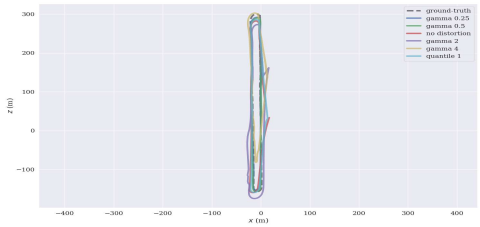
Figure 6.13 shows the comparison of the trajectories performed by the algorithms with each distortion in KITTI and Euroc sequences. In sequences KITTI 03 and KITTI 06, LIFT-SLAM's trajectories were not much affected by distortions (figures 6.13b and 6.13d). On the other hand, ORB-SLAM trajectories are worse, especially in sequence KITTI 06 (figure 6.13c). Furthermore, the trajectories of both algorithms were more affected in KITTI 07 (figures 6.13e and 6.13f), were ORB-SLAM could not track a considerable part of the trajectory. In MH_02, both algorithms' trajectories were less affected, but they lost track of the pose and relocalized in some parts of the sequence.



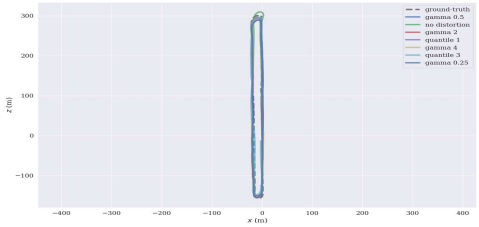
(a) ORB-SLAM in KITTI 03



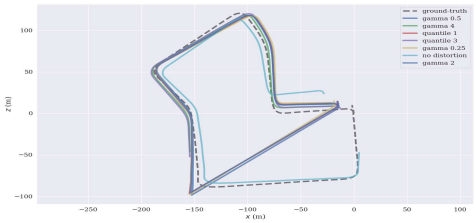
(b) LIFT-SLAM in KITTI 03



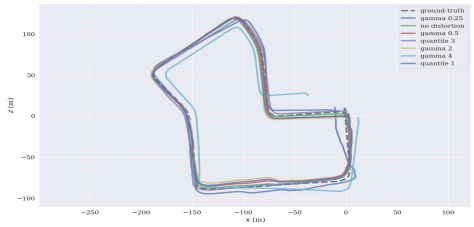
(c) ORB-SLAM in KITTI 06



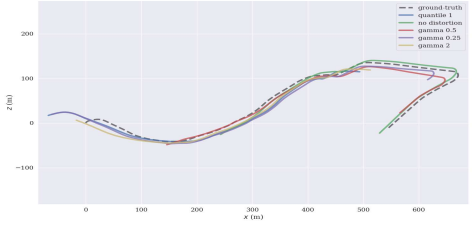
(d) LIFT-SLAM in KITTI 06



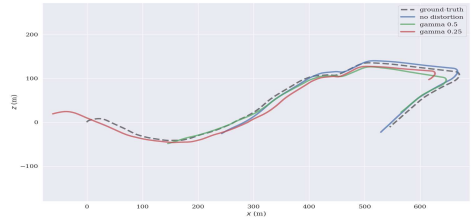
(e) ORB-SLAM in KITTI 07



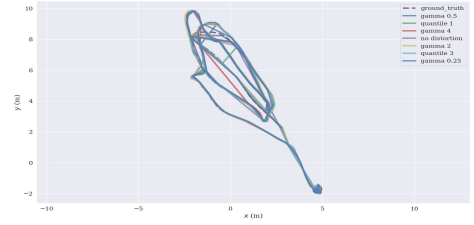
(f) LIFT-SLAM in KITTI 07



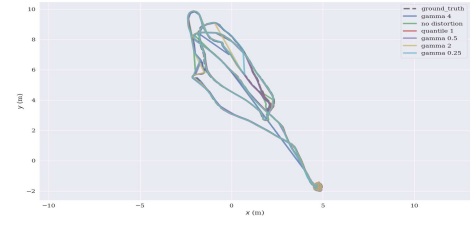
(g) ORB-SLAM in KITTI 10



(h) LIFT-SLAM in KITTI 10



(i) ORB-SLAM in Euroc MH_02



(j) LIFT-SLAM in Euroc MH_02

Figure 6.13: Comparison of the 2D trajectories performed by the algorithms with and without distortions.

6.8 Discussion and comparison with literature

The results obtained have shown that LIFT is capable of improving a typical VO/VSLAM algorithm. Moreover, transfer learning proved to be a crucial process in our system since it improved our algorithms in different VO/VSLAM problems. Furthermore, in general, the adaptive versions of LIFT-SLAM loses track of the trajectories less often than the other versions. The main drawback of our system is that good performance in large environments depends on loop closure detection. If a loop is not recognized, the error increases indefinitely since the drift accumulation is not corrected with pose graph optimization, as in sequences 08 and 09.

Considering the quantitative and qualitative results of all LIFT-SLAM versions, Adaptive LIFT-SLAM fine-tuned with EuRoC sequences is the one with better overall results. Therefore, we chose this version of LIFT-SLAM to make a comparison with some other results available in the literature. We provide in table 6.14 the comparison with other algorithms in KITTI dataset, we chose different types of monocular VO/VSLAM algorithms to compare with LIFT-SLAM: traditional methods, hybrid deep learning-based methods and end-to-end methods. Unfortunately, there are not many monocular algorithms that evaluate EuRoC available in the literature.

Algorithm	**Type**	**Metric**	**00**	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**10**
LIFT-SLAM	Hybrid	ATE (m)	8.06	X	40.04	2.23	0.51	13.55	30.38	3.63	184.43	59.62	29.87
		RPE_{trans} (%)	3.18	X	8.73	1.46	2.22	6.09	12.24	2.42	47.10	19.91	9.72
		RPE_{rot} (deg/m)	2.99	X	2.49	0.34	0.48	3.11	2.91	4.02	2.02	2.14	2.24
ORB-SLAM*	Traditional	ATE (m)	11.54	X	X	15.13	4.29	7.74	20.26	13.47	39.51	49.67	19.94
		RPE_{trans} (%)	4.46	X	X	9.75	3.71	3.35	8.11	7.43	12.16	26.51	8.65
		RPE_{rot} (deg/m)	3.28	X	X	2.78	2.15	3.57	2.88	3.58	3.05	11.13	3.62
ORB-SLAM [59]	Traditional	ATE (m)	5.33	X	21.28	1.51	1.62	4.85	12.34	2.26	46.68	6.62	8.80
		RPE_{trans} (%)	-	-	-	-	-	-	-	-	-	-	-
		RPE_{rot} (deg/m)	-	-	-	-	-	-	-	-	-	-	-
DeepVO [87]**	End-to-end	ATE (m)	-	-	-	-	-	-	-	-	-	-	-
		RPE_{trans} (%)	-	-	-	8.49	7.19	2.62	5.42	3.91	-	-	8.11
		RPE_{rot} (deg/m)	-	-	-	6.89	6.97	3.61	5.82	4.60	-	-	8.83
NeuralBundler [48]	Hybrid	ATE (m)	-	-	-	-	-	-	-	-	-	-	-
		RPE_{trans} (%)	3.24	-	4.85	-	-	1.83	2.74	3.53	-	6.23	-
		RPE_{rot} (deg/m)	1.35	-	1.60	-	-	0.7	2.6	2.02	-	2.11	-
UnDeepVO [47]**	End-to-end	ATE (m)	-	-	-	-	-	-	-	-	-	-	-
		RPE_{trans} (%)	4.14	-	5.58	-	-	3.40	-	3.15	4.08	-	-
		RPE_{rot} (deg/m)	1.92	-	2.44	-	-	1.50	-	2.48	1.79	-	-

* Our executions.

** Only VO.

Table 6.14: Comparison of LIFT-SLAM with results from monocular VO/VSLAM algorithms available in the literature. We fill with "X" results that are unavailable due to tracking failure and with "-" results that were not given by the authors.

Although the presented papers do not present results for all sequences with all metrics, table 6.14 shows that, in general, the hybrid methods (Ours and Neural Bundler [48]) achieved better performance than traditional and end-to-end methods. This fact shows that the use of DNNs combined with traditional back-end methods improves the performance of VSLAM algorithms.

Chapter 7

Conclusion and Future Work

In this work, we successfully apply a deep neural network called LIFT in the front-end of a traditional visual SLAM algorithm. This approach showed that it is possible to combine deep neural networks with conventional VO/VSLAM pipelines. Moreover, this combination can improve the performance of VSLAM algorithms, which proves our hypothesis H_1 (presented in Chapter 1).

We also showed that we could employ transfer learning to fine-tune feature description networks with VO/VSLAM datasets to improve the system's performance on cross-datasets. Moreover, we created a method to adapt the matching thresholds while executing the VO pipeline, depending on the number of outliers found. All of these methods allowed us to evaluate five LIFT-SLAM variations: LIFT-SLAM fine-tuned with KITTI sequences, LIFT-SLAM fine-tuned with EuRoC sequences, Adaptive LIFT-SLAM, Adaptive LIFT-SLAM fine-tuned with KITTI sequences and Adaptive LIFT-SLAM fine-tuned with EuRoC sequences.

The proposed hybrid system can operate in different environments (indoors and outdoors), which proves our hypothesis H_2. Furthermore, the algorithm improves its results with an artificial distortion applied to the images (gamma power transformation and quantile-based truncation). This fact showed us that a selection of the learned features could improve the performance of the algorithm. However, it still has some drawbacks. It is based on a constant velocity model, and therefore it assumes that the motion between two frames is smooth. This assumption can lead us to a significant drift accumulation if the environment is large and without loops or if the camera's acceleration is too big. Moreover, this system is not real-time, as the LIFT network cannot execute a feed-forward in the same frequency as a usual camera.

For future work, we intend to create an end-to-end method by training a neural network that learns the VO motion estimation, which means that it reads the feature descriptors and outputs a pose. We can also explore in future works a way of training jointly the full pipeline; in this way, the feature extraction step can be optimized directly from VO data. However, to do this, unsupervised or self-supervised approaches need to be explored since VO/VSLAM datasets are still not large enough to outperform traditional systems. The absence of data could be approached by following recent research that builds large datasets on photo-realistic simulated scenarios, like Microsoft Airsim [73], fine-tuning the resulted model in real-world scenes.

We also plan to create hybrid feature descriptors (e.g., LIFT+ORB) to leverage each descriptor's best characteristics in different scenarios. Moreover, an attention-based mechanism can be used to perform feature selection to avoid outliers and improve the algorithm's performance.

Bibliography

[1] H. Aanæs, A.L. Dahl, and K. Steenstrup Pedersen. Interesting interest points. *International Journal of Computer Vision*, 97:18–35, 2012.

[2] Mohammad O. A. Aqel, Mohammad H. Marhaban, M. Iqbal Saripan, and Napsiah Bt. Ismail. Review of visual odometry: types, approaches, challenges, and applications. Springer Plus, `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5084145/`, 10 2016.

[3] Vassileios Balntas, Edgar Riba, Daniel Ponsa, and Krystian Mikolajczyk. Learning local feature descriptors with triplets and shallow convolutional neural networks. In *Proceedings of the British Machine Vision Conference (BMVC)*. BMVA Press, September 2016.

[4] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008. ISSN 1077-3142. doi: 10.1016/j.cviu.2007.09.014. URL `https://doi.org/10.1016/j.cviu.2007.09.014`.

[5] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[6] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W. Achtelik, and Roland Siegwart. The euroc mav datasets. *The International Journal of Robotics Research*, 2016.

[7] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian D. Reid, and John J. Leonard. Past, present, and future of simultaneous localization and mapping: Present, future, and the robust-perception age. *IEEE Transactions on Robotics*, 32:1309–1332, 2016.

[8] Olivier Chapelle and Mingrui Wu. Gradient descent optimization of smoothed information retrieval metrics. *Information Retrieval*, 13(3):216–235, June 2010. ISSN 1386-4564. doi: 10.1007/s10791-009-9110-3. URL `https://doi.org/10.1007/s10791-009-9110-3`.

[9] Yang Cheng, Mark Maimone, and Larry Matthies. Visual odometry on the mars exploration rovers. *IEEE International Conference on Systems, Man and Cybernetics*, 2005.

[10] L. Contreras and W. Mayol-Cuevas. Towards cnn map representation and compression for camera relocalisation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 405–4057, 2018.

[11] I. Cvišić and I. Petrović. Stereo odometry based on careful feature selection and tracking. In *2015 European Conference on Mobile Robots (ECMR)*, pages 1–6, 2015.

[12] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29 (6):1052–1067, 2007.

[13] J. Delmerico, T. Cieslewski, H. Rebecq, M. Faessler, and D. Scaramuzza. Are we ready for autonomous drone racing? the uzh-fpv drone racing dataset. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6713–6719, 2019.

[14] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Self-improving visual odometry. *CoRR*, abs/1812.03245, 2018. URL `http://arxiv.org/abs/1812.03245`.

[15] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 337–349. IEEE, 2018.

[16] Nigel Dias and Gustavo Laureano. Accurate stereo visual odometry based on keypoint selection. In *2019 Latin American Robotics Symposium (LARS), 2019 Brazilian Symposium on Robotics (SBR) and 2019 Workshop on Robotics in Education (WRE)*, pages 74–79, 10 2019. doi: 10.1109/LARS-SBR-WRE48964.2019.00021.

[17] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics and Automation Magazine*, 13:99–110, 2006.

[18] Luigi D'Alfonso, Antonio Grano, Pietro Muraca, and Paolo Pugliese. A polynomial based slam algorithm for mobile robots using ultrasonic sensors - experimental results. *International Conference on Advanced Robotics (ICAR)*, 2013.

[19] Jakob Engel, Thomas Schops, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. *European Conference on Computer Vision. Springer International Publishing*, pages 834–849, 2014.

[20] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *CoRR*, abs/1607.02565, 2016. URL `http://arxiv.org/abs/1607.02565`.

[21] Jakob Engel, Vladyslav Usenko, and Daniel Cremers. A photometrically calibrated benchmark for monocular visual odometry. volume abs/1607.02555, 2016. URL `http://arxiv.org/abs/1607.02555`.

[22] Jakob-Julian Engel. *Large-Scale Direct SLAM and 3D Reconstruction in Real-Time*. PhD thesis, Fakultat fur Informatik, Der Technischen Universitat Munchen, 3 2017.

[23] Martín Abadi et. al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[24] Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg, Philip Häusser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. *IEEE International Conference on Computer Vision (ICCV)*.

[25] Thomas Braunl Franco Hidalgo. Review of underwater slam techniques. *International Conference on Automation, Robotics and Applications (ICARA)*, 2015.

[26] Friedrich Fraundorfer and Davide Scaramuzza. Visual odometry part ii: Matching, robustness, optimization, and applications. *IEEE Robotics Automation Magazine*, 2012.

[27] S. Selvakumar G. Nirmala, S. Geetha. Mobile robot localization and navigation in artificial intelligence: Survey. *Computational Methods in Social Sciences*, 4:12–22, 2016.

[28] D. Galvez-López and J. D. Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, 2012.

[29] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[30] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The international Journal of Robotics Research*, 32(11): 1231–1237, 2013.

[31] Clément Godard, Oisin Mac Aodha, and Gabriel J. Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[32] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. `http://www.deeplearningbook.org`.

[33] Giorgio Grisetti, Rainer Kummerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, pages 31–43, 2010.

[34] Michael Grupp. evo: Python package for the evaluation of odometry and slam. `https://github.com/MichaelGrupp/evo`, 2017.

[35] Richard I. Hartley. In defense of the eight-point algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(6):580–593, 1997.

[36] Richard I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, 2004.

[37] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

[38] Ganesh Iyer, J. Krishna Murthy, Gunshi Gupta, K. Madhava Krishna, and Liam Paull. Geometric consistency for self-supervised end-to-end visual odometry. *EEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2018.

[39] Rong Kang, Jieqi Shi, Xueming Li, Yang Liu, and Xiao Liu. DF-SLAM: A deep-learning enhanced visual SLAM system based on deep local features. *CoRR*, abs/1901.07223, 2019. URL `http://arxiv.org/abs/1901.07223`.

[40] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. *6th IEEE and ACM International Symposium*, pages 225–234, 2007.

[41] L. Kneip, D. Scaramuzza, and R. Siegwart. A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2969–2976, 2011.

[42] Kishore Konda and Roland Memisevic. Learning visual odometry with a convolutional network. *VISAPP*, pages 486–490, 2015.

[43] Kudan. Different types of visual slam systems. `https://www.kudan.eu/kudan-news/different-types-visual-slam-systems`, 2017.

[44] Rainer Kummerle, Bastian Steder, Christian Dornhege, Michael Ruhnke, Giorgio Grisetti, Cyrill Stachniss, and Alexander Kleiner. On measuring the accuracy of slam algorithms. *Autonomous Robots, Springer*, 27(387), 2009.

[45] Gim Hee Lee, Friedrich Fraundorfer, and Marc Pollefeys. Motion estimation for self-driving cars with a generalized camera. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2013.

[46] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate o(n) solution to the pnp problem. *International Journal of Computer Vision*, 81, 02 2009. doi: 10.1007/s11263-008-0152-6.

[47] Ruihao Li, Sen Wang, Zhiqiang Long, and Dongbing Gu. Undeepvo: Monocular visual odometry through unsupervised deep learning. *CoRR*, abs/1709.06841, 2017. URL `http://arxiv.org/abs/1709.06841`.

[48] Y. Li, Y. Ushiku, and T. Harada. Pose graph optimization for unsupervised monocular visual odometry. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 5439–5445, 2019.

[49] H. C. Longuet-Higgins. *A Computer Algorithm for Reconstructing a Scene from Two Projections*, page 61–62. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987. ISBN 0934613338.

[50] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004. doi: 10.1023/B:VISI.0000029664.99615.94.

[51] S. Lowry, N. Sünderhauf, P. Newman, J. J. Leonard, D. Cox, P. Corke, and M. J. Milford. Visual place recognition: A survey. *IEEE Transactions on Robotics*, 32(1):1–19, 2016.

[52] A.L. Majdik, C. Till, and D. Scaramuzza. The zurich urban micro aerial vehicle dataset. *The International Journal of Robotics Research*, 2017.

[53] Harsh Menon, Aashik Ramachandrappa, and Jake Kesinger. 37-3: Invited paper: Deep-learning based approaches to visual-inertial odometry for autonomous tracking applications. *SID Symposium Digest of Technical Papers*, 10.1002/sdtp.12603:471–474, 2018.

[54] missinglink.ai. Convolutional neural network tutorial: From basic to advanced. `https://missinglink.ai/guides/convolutional-neural-networks/convolutional-neural-network-tutorial-basic-advanced`, 2020.

[55] Hans Moravec. *Obstacle avoidance and navigation in the real world by a seeing robot rover*. PhD thesis, Carnegie-Mellon University, 9 1980.

[56] E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck, and D. Scaramuzza. The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and slam. *The International Journal of Robotics Research*, 36:142–149, 2017.

[57] Raul Mur-Artal and Juan Tardos. Fast relocalisation and loop closing in keyframe-based slam. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 846–853, 06 2014. doi: 10.1109/ICRA.2014.6906953.

[58] Raul Mur-Artal and Juan Tardos. Orb-slam2: an open-source slam system for monocular, stereo and rgb-d cameras. *IEEE Transactions on Robotics*, PP, 10 2016. doi: 10.1109/TRO.2017.2705103.

[59] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[60] Robin R. Murphy. *Introduction to AI Robotics*. The MIT Press, Cambridge, Massachusetts, 2000. ISBN 0262133830.

[61] Richard Newcombe, Steven J. Lovegrove, and Andrew J. Davison. Dtam: Dense tracking and mapping in real-time. *IEEE International Conference In Computer Vision*, pages 2320–2327, 2011.

[62] David Nistér. An efficient solution to the five-point relative pose problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):756–777, June 2004. ISSN 0162-8828. doi: 10.1109/TPAMI.2004.17. URL https://doi.org/10.1109/TPAMI.2004.17.

[63] David Nistér and Henrik Stewenius. Scalable recognition with a vocabulary tree. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, CVPR '06, page 2161–2168, USA, 2006. IEEE Computer Society. ISBN 0769525970. doi: 10.1109/CVPR.2006.264. URL https://doi.org/10.1109/CVPR.2006.264.

[64] David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1, 2004.

[65] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[66] Emilio Soria Olivas, Jose David Martin Guerrero, Marcelino Martinez Sober, Jose Rafael Magdalena Benedito, and Antonio Jose Serrano Lopez. *Handbook Of Research On Machine Learning Applications and Trends: Algorithms, Methods and Techniques - 2 Volumes*. Information Science Reference - Imprint of: IGI Publishing, Hershey, PA, 2009.

[67] Emilio Parisotto, Devendra Singh Chaplot, Jian Zhang, and Ruslan Salakhutdinov. Global pose estimation with an attention-based recurrent network. *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 350–359, 2018.

[68] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, page 2564–2571, USA, 2011. IEEE Computer Society. ISBN 9781457711015. doi: 10.1109/ICCV.2011.6126544. URL https://doi.org/10.1109/ICCV.2011.6126544.

[69] Muhamad Risqi U. Saputra, Andrew Markham, and Niki Trigoni. Visual slam and structure from motion in dynamic environments: A survey. *ACM Computing Surveys*, 51:37–73, 2018.

[70] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superglue: Learning feature matching with graph neural networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4938–4947, 2020.

[71] Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry part i: The first 30 years and fundamentals. *IEEE Robotics and Automation Magazine*, 2011.

[72] D. Schubert, T. Goll, N. Demmel, V. Usenko, J. Stueckler, and D. Cremers. The tum vi benchmark for evaluating visual-inertial odometry. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[73] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2017. URL `https://arxiv.org/abs/1705.05065`.

[74] Roland Siegwart and Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. The MIT Press, Cambridge, Massachusetts, 2004. ISBN 026219502X.

[75] Mike Smith, Ian Baldwin, Winston Churchill, Rohan Paul, and Paul Newman. The new college vision and laser data set. *The International Journal of Robotics Research*, 28:595–599, 2009.

[76] Stanford Artificial Intelligence Laboratory et al. Robotic operating system. https://www.ros.org, 2020.

[77] C. R. Steffens, L. R. V. Messias, P. L. J. Drews, and S. S. d. C. Botelho. Can exposure, noise and compression affect image recognition? an assessment of the impacts on state-of-the-art convnets. In *2019 Latin American Robotics Symposium (LARS), 2019 Brazilian Symposium on Robotics (SBR) and 2019 Workshop on Robotics in Education (WRE)*, pages 61–66, 2019.

[78] Christoph Strecha, Wolfgang Hansen, Luc Van Gool, Pascal Fua, and Ulrich Thoennessen. On benchmarking camera calibration and multi-view stereo for high resolution imagery. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 06 2008. doi: 10.1109/CVPR.2008.4587706.

[79] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2010.

[80] Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda. Visual slam algorithms: a survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, 9:16, 2017.

[81] J. Tang, L. Ericson, J. Folkesson, and P. Jensfelt. Gcnv2: Efficient correspondence prediction for real-time slam. *IEEE Robotics and Automation Letters*, 4(4):3505–3512, 2019.

[82] Shaharyar Ahmed Khan Tareen and Zahra Saleem. A comparative analysis of sift, surf, kaze, akaze, orb, and brisk. In *International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, pages 1–10, 2018.

[83] Bill Triggs. Detecting keypoints with stable position, orientation and scale under illumination changes. In *European Conference on Computer Vision (ECCV)*, volume 3024, pages 100–113, 05 2004.

[84] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(4):376–380, 1991.

[85] Yannick Verdie, Kwang Moo Yi, Pascal Fua, and Vincent Lepetit. Tilde: A temporally invariant learned detector. *Proceedings of the Computer Vision and Pattern Recognition*, 2015.

[86] Lukas von Stumberg, Vladyslav C. Usenko, Jakob Engel, Jörg Stückler, and Daniel Cremers. From monocular slam to autonomous drone exploration. *European Conference on Mobile Robots (ECMR), IEEE*, 2017.

[87] S. Wang, R. Clark, H. Wen, and N. Trigoni. Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2043–2050, 2017.

[88] Brian Williams, Mark Cummins, José Neira, Paul Newman, Ian Reid, and Juan Tardos. A comparison of loop closing techniques in monocular slam. *Robotics and Autonomous Systems*, 57:1188–1197, 12 2009. doi: 10.1016/j.robot.2009.06.010.

[89] Kyle Wilson and Noah Snavely. Robust global translations with 1dsfm. In *European Conference on Computer Vision (ECCV)*, volume 8691, pages 61–75. Springer, 2014.

[90] Oliver J. Woodman. An introduction to inertial navigation. Technical Report 696, University of Cambridge, Computer Laboratory, 15 JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom, 8 2007.

[91] C. Wu. Towards linear-time incremental structure from motion. In *International Conference on 3D Vision - 3DV*, pages 127–134, 2013.

[92] Kwang Moo Yi, Eduard Trulls, Vincent Lepetit, and Pascal Fua. Lift: Learned invariant feature transform. In *European Conference on Computer Vision (ECCV)*, volume 9910, pages 467–483. Springer, 2016. ISBN 978-3-319-46465-7. doi: https://doi.org/10.1007/978-3-319-46466-4_28.

[93] Khalid Yousif, Alireza Bab-Hadiashar, and Reza Hoseinnezhad. An overview to visual odometry and visual slam: Applications to mobile robotics. *Springer: Intelligent Industrial Systems*, 1:289–310, 11 2015. doi: 10.1007/s40903-015-0032-7.