Hardware Software Co-design of an Embedded RGB-D SLAM System

Imad El Bouazzaoui

université
PARIS-SACLAY

Hardware Software Co-design of an Embedded RGB-D SLAM System

Adéquation algorithme-architecture pour un système SLAM RGB-D embarqué

Thèse de doctorat de l'université Paris-Saclay

École doctorale n° 580, Sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Robotique
Graduate School : Sciences de l'ingénieur et des systèmes (SIS),
Référent : Faculté des sciences d'Orsay

Thèse préparée dans l'unité de recherche SATIE (Université Paris-Saclay, ENS Paris-Saclay, CNRS), sous la direction de Abdelhafid EL OUARDI, Maître de Conférences HDR, le co-encadrement de Sergio RODRIGUEZ, Maître de Conférences

Thèse soutenue à Paris-Saclay, le 06 décembre 2022, par

Imad EL BOUAZZAOUI

Composition du jury

Membres du jury avec voix délibérative

Mme Véronique BERGE-CHERFAOUI Professeure des universités, Université de Technologie de Compiègne	Présidente
M. Jean-François NEZAN Professeur des universités, INSA Rennes	Rapporteur & Examinateur
M. Pascal VASSEUR Professeur des universités, Université de Picardie Jules Verne	Rapporteur & Examinateur
M. Samir BOUAZIZ Professeur des universités, Université Paris-Saclay	Examinateur
Mme Myriam SERVIÈRES Maîtresse de conférences, Centrale Nantes	Examinatrice

THESE DE DOCTORAT

NNT : 2022UPAST156

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Adéquation algorithme-architecture pour un système SLAM RGB-D embarqué
Mots clés : Capteurs RGB-D, SLAM, Architecture des systèmes, Systèmes temps réel, Localisation de véhicule, Systèmes embarqués

Résumé : Les capteurs de vision délivrant des images en couleur et l'information de profondeur ont récemment gagné en popularité. Les véhicules autonomes bénéficient de nouvelles méthodes de perception 3D grâce à ces capteurs. Nous avons étudié les différentes étapes de traitement d'un système afin d'apporter des contributions au niveau du couplage capteur-algorithme et de l'architecture de calcul. Cette étude a commencé par une analyse expérimentale approfondie de l'impact des modalités d'acquisition des capteurs sur la précision de la localisation. Nous avons développé la méthode HOOFR-SLAM RGB-D, qui a été évaluée à l'aide d'un ensemble de données RGB-D. Les résultats obtenus ont montré une amélioration significative de localisation et du temps de traitement. Nous avons proposé un modèle d'architecture basé sur FPGA pour le pré-traitement du HOOFR SLAM. Cette architecture offre des performances supérieures et un compromis entre la consommation d'énergie et le temps de traitement.

Title : Hardware Software Co-design of an Embedded RGB-D SLAM System
Keywords : RGB-D sensors, SLAM, Systems architecture, Real-time systems, Vehicle localization, Embedded systems

Abstract : Vision sensors with color and depth have recently gained popularity. Autonomous vehicles benefit from new 3D perception methods thanks to these sensors. We have investigated the various processing stages of the system to make contributions at the sensor-algorithm coupling and computing architecture levels. This study began by conducting a thorough experimental analysis of the impact of sensor acquisition modalities on localization accuracy. We introduced RGB-D HOOFR-SLAM, which was assessed using a self-collected RGB-D dataset. We compared the measurement results to those of the most advanced algorithms. The results revealed a significant reduction in localization errors and a significant gain in processing speed compared to the state-of-the-art stereo and RGB-D algorithms. We proposed the implementation of the HOOFR SLAM front-end on an FPGA-based architecture. This innovative architecture delivers superior performance and a trade-off between power consumption and processing time.

Hardware Software Co-design of an Embedded RGB-D SLAM System

December 2022

Abstract

Recent research at the SATIE laboratory has developed a 2D visual SLAM system, HOOFR-SLAM, which has competitive performance in computation time and localization outcomes compared to the state-of-the-art. The method has been evaluated and validated using well-known publicly accessible datasets (KITTI, NewCollege, Malaga, MRT, and St. Lucia).

In the realm of embedded vision systems, innovative 3D vision sensors with color and depth images (stereovision, RGB-D cameras, and LiDAR camera systems) have recently gained popularity. Robots and autonomous vehicles benefit from new 3D perception methods thanks to these sensors. We have investigated the various processing stages of the system, from the sensor to the embedded architecture, to make contributions at the sensor-algorithm coupling and computing architecture levels.

Several RGB-D SLAM algorithms have been studied and evaluated using publicly available datasets without considering sensor specifications or image acquisition modalities that could improve or degrade localization accuracy. This study began by conducting a thorough experimental analysis of the impact of sensor acquisition modalities on localization accuracy. We produced an online indoor Visual Simultaneous Localization And Mapping (V-SLAM) dataset with multiple acquisition modalities to determine their impact on the accuracy of the Visual SLAM algorithm. The dataset consists of sequences recorded using various modalities, such as RGB, IR, and depth images in passive and active stereo modes. For comparison, each sequence was coupled with a Structure from Motion (SfM), and Multi-View Stereo (MVS) based reference trajectory. The datasets have a lot of different areas, some of which have low brightness, change in brightness, are wide, narrow, or have a different texture. Most known SLAM algorithms have been

selected and evaluated on these datasets. The results showed that the sensor's parameters, especially those related to the field of view, depth threshold, and IR projector, must be tightly coupled in an RGBD-based SLAM system design for accurate localization. Although various algorithms are available for SLAM RGB-D, most are designed for indoor applications and have not been assessed or adapted to outdoor vehicle applications. The second stage in the design of our RGB-D SLAM system is based on the research of sensor-algorithm coupling.

We introduced RGB-D HOOFR-SLAM: an RGB-D SLAM method for autonomous vehicle localization based on the HOOFR-SLAM stereo algorithm. This version addresses the most prevalent camera issues in outdoor contexts: environments with an image-dominant overcast sky and the presence of dynamic objects. We used a depth-based filtering method to identify outlier points based on their depth value. The method is robust against outliers and also computationally inexpensive. Improvements have been made to the processor-based SLAM kernel's algorithms by replacing the RANSAC method used for essential matrix estimation with PROSAC. We assessed the algorithm using a self-collected RGB-D dataset gathered by the SATIE laboratory instrumented vehicle. We compared the measurement results to those of the most advanced algorithms by assessing translational error and average processing time. The results revealed a significant reduction in localization errors and a significant gain in processing speed compared to the state-of-the-art stereo and RGB-D algorithms.

Finally, to move the processing as near as possible to the sensor on an embedded device and to fulfill the real-time constraints, we investigated the algorithmic complexity of the front-end task of the HOOFR SLAM and the existing hardware architectures dedicated to embedded systems. We used an approach based on examining the algorithm's complexity, workloads, and functional blocs partitioning. Each block's processing time is evaluated according to the architecture's constraints. We proposed the implementation of the HOOFR SLAM front-end on a CPU-FPGA architecture, including feature extraction and matching functional blocks. A high-level synthesis (HLS) approach employing the OpenCL paradigm has been used to design a new system architecture. The performance of the FPGA-based architecture was compared to a high-performance CPU. This innovative

architecture delivers superior performance and a trade-off between power consumption and processing time compared to existing systems.

Résumé

Les travaux de recherche récents du laboratoire SATIE ont permis de développer un système SLAM visuel 2D, appelé HOOFR-SLAM, qui présente des performances compétitives en termes de temps de calcul et de résultats de localisation par rapport à l'état de l'art. La méthode a été évaluée et validée à l'aide de jeux de données publics bien connus (KITTI, NewCollege, Malaga, MRT et St. Lucia).

Dans le domaine des systèmes de vision embarqués, les capteurs de vision 3D innovants, qui fournissent des images en couleur et en profondeur (stéréovision, caméras RGB-D et systèmes de caméras LiDAR), sont devenus de plus en plus répandus ces dernières années. Grâce à ces capteurs, les robots et les véhicules autonomes peuvent bénéficier de nouvelles méthodes de perception 3D. Nous avons étudié les différentes étapes de traitement du système, du capteur à l'architecture embarquée, afin d'apporter des contributions au niveau du couplage capteur-algorithme et de l'architecture de calcul.

Plusieurs algorithmes SLAM RGB-D ont été étudiés et évalués à l'aide de jeux de données disponibles au public sans tenir compte des spécifications du capteur ou des modalités d'acquisition des images qui pourraient améliorer ou dégrader la précision de la localisation. Cette étude a commencé par une analyse expérimentale approfondie de l'impact des modalités d'acquisition des capteurs sur la précision de la localisation. Nous avons réalisé un jeu de données, qui est disponible en ligne, dédié à la localisation et à la cartographie visuelles simultanées (V-SLAM) en indoor avec plusieurs modalités d'acquisition afin de déterminer leur impact sur la précision de l'algorithme SLAM visuel. Le jeu de données se compose de séquences enregistrées à l'aide de diverses modalités, telles que des images RGB, IR et de profondeur en modes stéréo passif et actif. À des fins de comparaison, chaque séquence a été couplée à une trajectoire de référence basée sur la méthode

d'estimation de la structure à partir du mouvement (Structure from Motion - SfM) et la stéréo multi-vues (Multi-View Stereo - MVS). Les ensembles de données comportent de nombreuses zones différentes, dont certaines ont une faible luminosité, varient en luminosité, sont larges, étroites ou présentent une texture variée. Les algorithmes SLAM les plus connus ont été sélectionnés et évalués sur ces jeux de données. Les résultats ont montré que les paramètres du capteur, en particulier ceux liés au champ de vision, au seuil de profondeur et au projecteur IR, doivent être fortement couplés dans la conception d'un système SLAM basé sur RGB-D pour une localisation précise. Bien que plusieurs algorithmes soient disponibles pour le SLAM RGB-D, la plupart sont conçus pour des applications en environnements internes (indoor) et n'ont pas été évalués ou adaptés aux applications de véhicules (outdoor). La deuxième étape de la conception de notre système SLAM RGB-D est basée sur cette étude de couplage capteur-algorithme.

Nous introduisons RGB-D HOOFR-SLAM : une méthode SLAM RGB-D pour la localisation de véhicules autonomes basée sur l'algorithme stéréo HOOFR-SLAM. Cette version aborde les problèmes de caméra les plus courants dans les contextes extérieurs : environnements avec un ciel couvert dominant l'image et présence d'objets dynamiques. Nous avons utilisé une méthode de filtrage basée sur la profondeur pour identifier les points aberrants en fonction de leur valeur de profondeur. Cette méthode est robuste contre les points aberrants et peu coûteuse en termes de calcul. Des améliorations ont été apportées à l'algorithme du calcul de la pose, en remplaçant la méthode RANSAC utilisée pour l'estimation de la matrice essentielle par PROSAC. Nous avons évalué l'algorithme en utilisant un ensemble de données RGB-D collectées par le véhicule instrumenté du laboratoire SATIE. Nous avons comparé les résultats des mesures à ceux des algorithmes les plus avancés en évaluant l'erreur de translation et le temps de traitement moyen. Les résultats ont révélé une réduction significative des erreurs de localisation et un gain important de la vitesse de traitement par rapport aux algorithmes stéréo et RGB-D les plus avancés.

Enfin, pour pousser le traitement au plus près du capteur sur un dispositif embarqué et afin de respecter les contraintes de temps réel, nous avons étudié la complexité algorithmique de la partie front-end du HOOFR-SLAM et les architectures matérielles existantes dédiées

aux systèmes embarqués. Nous avons utilisé une approche basée sur l'examen de la complexité de l'algorithme, des charges (Workloads) et du partitionnement des blocs fonctionnels. Le temps de traitement de chaque bloc est évalué en fonction des contraintes de l'architecture. Nous avons proposé une implémentation du front-end du HOOFR-SLAM sur une architecture CPU-FPGA, y compris les blocs fonctionnels de l'extraction et la mise en correspondance des primitives. Une approche de synthèse de haut niveau (HLS) employant le paradigme OpenCL a été utilisée pour concevoir une nouvelle architecture. Les performances de l'architecture basée sur FPGA ont été comparées à celles d'un CPU de haute performance. Cette architecture innovante offre des performances supérieures et un compromis entre la consommation d'énergie et le temps de traitement par rapport aux systèmes existants.

Publications

- *Journal:*

1. Imad El Bouazzaoui, Sergio A. Rodríguez Florez and Abdelhafid El Ouardi, "Enhancing RGB-D SLAM Performances Considering Sensor Specifications for Indoor Localization," in **IEEE Sensors Journal**, vol. 22, no. 6, pp. 4970-4977, **15 March15, 2022**, doi: 10.1109/JSEN.2021.3073676.

2. Imad El Bouazzaoui, Sergio Alberto Rodriguez Florez, Bastien Vincke, Abdelhafid El Ouardi. Indoor Visual SLAM Dataset With Various Acquisition Modalities. **Data in Brief,** Elsevier, **2021**, pp.107496. ⟨10.1016/j.dib.2021.107496⟩. ⟨hal-03400312⟩

Acknowledgement

First, I would like to thank Ms. Véronique Berge-Cherfaoui for honoring me by chairing my thesis jury and for her role as an examiner. I would also like to thank Mr. Pascal Vasseur and Mr. Jean-François Nezan for their work as reviewers and Ms. Myriam Servières and Mr. Samir Bouaziz for their work as examiners.

I want to express my gratitude to my thesis supervisor Mr. Abdelhafid El Ouardi and my co-supervisor, Mr. Sergio Rodriguez, who guided me throughout these three years. They advised me, supported me, and helped me during the whole of my thesis.

I had the pleasure of working in the MOSS group of the laboratory Systèmes et Applications des Technologies de l'Information et de l'Énergie (SATIE) at the Ecole Normale Supérieure de Cachan (ENS). Thank you for having welcomed me to carry out my thesis work within the group and for the pleasant working environment you offered me throughout these three years.

I would also like to thank Mr. Mohamed Abouzahir for welcoming me to the Mohamed V University of Rabat to evaluate the algorithms developed on a hardware platform and use the associated tools.

Finally, I would like to thank my parents, my wife, and my whole family, without whom I would not have achieved the academic path I did. I am eternally grateful to you for encouraging me in every moment.

Contents

List of Tables

List of Figures

List of Abbreviations

AAA	Algorithm-Architecture Adequacy
ADAS	Advanced Driver Assistance System
AEKF	Adaptive Extended Kalman Filter
AER	Azimuth-Elevation-Range
AI	Artificial Intelligence
AMD	Advanced Micro Devices
APS	Altus Positioning Systems
AR	Augmented Reality
ARM	Advanced RISC Machines / Acorn RISC Machine
ATE	Absolute Trajectory Error
BA	Bundle Adjustment
BAD SLAM	Bundle Adjusted Direct SLAM
BoW	Bag of Word
BRIEF	Binary Robust Independent Elementary Features
BSP	Board Support Package
CNN	Convolutional Neural Network

CPLD	Complex Programmable Logic Device
CPM	Cycles Per Match
CPU	Central Processing Unit
CSM	Correlative Scan Matching\{ }\{ }
DAG	Directed Acyclic Graph
DDR4 SDRAM	Double Data Rate 4th generation Synchronous Dynamic Random Access Memory
DNN	Deep Neural Network
DoF	Degrees of Freedom
DRAM	Dynamic Random Access Memory
DSO	Direct Sparse Odometry
DSP	Digital Signal Processor
DTAM	Dense Tracking And Mapping
DVO-SLAM	Dense Visual Odometry and SLAM
ECEF	Earth-Centered Earth-Fixed
EKF	Extended Kalman Filters
EME	Essential Matrix Estimation
ENU	East-North-Up
F2F	Frame-To-Frame
FAST	Features from Accelerated Segment Test
FIFO	First In First Out
FMCW	Frequency-Modulated Continuous-Wave

FN	False Negative
FOV	Field Of View
FP	False Positive
FPGA	Field-Programmable Gate Array
FPR	False Positive Rate
FPS	Frames-Per-Second
g2o	General Graph Optimization
GFTT	GoodFeaturesToTrack
GMSK SLAM	Grid-based Motion Statistics K-means SLAM
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GPU	Graphics Processing Unit
HD	High Definition
HDL	Hardware Description Language
HDR	High Dynamic Range
HERO	Heterogeneous Extensible Robot Open
HIL	Hardware-In-the-Loop
HLS	High-Level Synthesis
HOOFR	Hessian ORB-Overlapped FREAK
ICL-NUIM	Imperial College London and National University of Ireland Maynooth
ICP	Iterative Closest Point
IMU	Inertial Measurement Unit

INS	Inertial Navigation System
IR	Infrared
KITTI	Karlsruhe Institute of Technology and Toyota Technological Institute
LeGO-LOAM	Lightweight and Ground Optimized Lidar Odometry And Mapping
LiDAR	Light Detection And Ranging
LM	Levenberg-Marquardt
LOAM	Lidar Odometry and Mapping
LPDDR4	Low Power Double Data Rate 4
LSD-SLAM	Large-Scale Direct Monocular SLAM
LTM	Long-Term Memory
MAD	Median Absolute Deviation
MAV	Micro Aerial Vehicle
ME	Measurement Error
MEE	Mean Euclidean Error
MNC	Multi-task Network Cascades
MRT	Mess und Regelungstechnik (Measurement and Control Technology)
MVS	Multi-View Stereo
NCC	Normalized Cross-Correlation
NED	North-East-Down
OpenCL	Open Computing Language
OpenMP	Open Multi-Processing
ORB	Oriented FAST and Rotated BRIEF

PAL	Programmable Array Logic
PCIe	Peripheral Component Interconnect express
PL-GM SLAM	Point-Line Geometric constraint Model based SLAM
PNF	Previous Neighbor Frame
PnP	Perspective-n-Point
PROSAC	PROgressive SAmple Consensus
PTAM	Parallel Tracking And Mapping
RADAR	Radio Detection And Ranging
RAM	Random Access Memory
RANSAC	RANdom SAmple Consensus
RGB-D	Red Green Blue - Depth
RMSE	Root Mean Square Error
ROC	Receiver Operating Curve
RPE	Relative Pose Error
RS-BRIEF	Rotationally Symmetric - Binary Robust Independent Elementary Features
RTK	Real-Time Kinematic
S-PTAM	Stereo Parallel Tracking And Mapping
SATIE	Systèmes et Applications des Technologies de l'Information et de l'Energie
SDK	Software Development Kit
SfM	Structure from Motion

SIFT	Scale-Invariant Feature Transform
SIMD	Single Instruction on Multiple Data
SLAM	Simultaneous Localization And Mapping
SM	Streaming Multiprocessor
SoC	System on a Chip
SOFT	Stereo Odometry algorithm relying on Feature Tracking
SSD	Solid-State Drive
STM	Short-Term Memory
SURF	Speeded Up Robust Features
SVO	Semi-Direct Visual Odomtery
TN	True Negative
ToF	Time of Flight
TP	True Positive
TPR	True Positive Rate
TSDF	Truncated Signed Distance Function
TUM	Technische Universität München
UAV	Unmanned Aerial Vehicles
UWB	Ultra Wide Band
V-SLAM	Visual SLAM
VGA	Video Graphics Array
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VI-SLAM	Visual Inertial-SLAM

VO	Visual Odometry
VPU	Vision Processing Unit
VR	Virtual Reality
WLAN	Wireless Local Area Network
WM	Working Memory
WSG84	World Geodetic System of 1984
WSN	Wireless Sensor Network

Introduction

Navigation is a fundamental functionality of autonomous robots. The autonomous navigation of a robot answers two fundamental questions: Where am I, and what does the surrounding environment look like? The answer to these two questions has been introduced by a method known as SLAM (Simultaneous Localization And Mapping). It allows a mobile robot to simultaneously identify its surrounding environment and localize itself. Simultaneous localization and mapping (SLAM) have been considered a pillar of genuinely autonomous robots and, in this regard, is an essential task of autonomous vehicles.

With the development of sensors, several SLAM algorithms have been developed to solve the mapping and localization problems. The earliest SLAM algorithms were mainly based on laser sensors capable of providing accurate information on the depth of objects in the scene [13, 14]. Then, the monocular vision was introduced to substitute laser sensors [15, 16, 17]. Therefore, much research has been conducted to develop mathematical algorithms for Visual SLAM that provide the 3D reconstruction of the scene using a simple camera. In the field of embedded vision systems, innovative 3D vision sensors that provide color and depth images (Stereovision, RGB-D cameras, and LiDAR camera systems) have recently gained popularity, paving the way for new 3D perception methods [18, 19, 20] for robots and autonomous vehicles.

In earlier days, the use of RGB-D cameras was limited to gaming and entertainment [21]. As this type of sensor has evolved, RGB-D cameras are increasingly used in robotics and autonomous vehicles. RGB-D cameras provide an RGB image and the associated depth map, making it possible to solve the problem of scale drift with less complexity and to create a dense 3D environment representation. However, the use of RGB-D sensors is still limited to indoor environments due to some limitations (i.e., Infrared (IR) distortion with

ambient IR interference). With the development of new advanced RGB-D cameras (Intel-RealSense), the field of application has also been extended to outdoor environments [22].

In the state-of-the-art, we find that most SLAM algorithms are often evaluated on high-performance computers due to their complexity to ensure real-time processing and to guarantee the consistency of localization and mapping results. The widespread use of SLAM algorithms on robots and autonomous vehicles requires our attention to consider three axes of performance: processing time, consistency, and energy efficiency.

Nowadays, there are various embedded computers with specificities to be used efficiently in the context of SLAM to achieve computational optimizations and efficient embedded SLAM systems. These optimizations can be achieved using multicore processors and massively parallel architectures such as graphical or programmable FPGA architectures. Nevertheless, the implementation of these algorithms is strongly driven by the nature of the algorithm and the target architecture. As a result, algorithmic and hardware constraints must be considered to define adequate algorithm/architecture mapping.

The latest trend is pushing processing closer to the sensor. FPGAs constitute the perfect architecture for designing smart sensors by providing low latency suitable for real-time applications, such as video streaming, as they supply data directly into the FPGA without needing the CPU. Several studies have been conducted to design smart sensors to perform computer vision tasks [23, 24, 25].

Motivation

The field of autonomous vehicles is currently a trend in many research works. Several approaches have been adopted to solve the SLAM problem for autonomous vehicles. The Global Navigation Satellite System (GNSS) is a commonly used technology for localization. Nevertheless, this system has been considered limited due to its signal degradation in dense urban areas and scenarios with shadow effects. The Advanced Driver Assistance System (ADAS) was an alternative method that was evaluated, which was intended to assist vehicle localization. However, the availability of all road information, such as lane markings and road edges, is not ensured on all roads, making this method ineffective. On

the other hand, 4G/5G cellular systems, Ultra Wide Band (UWB), Wireless Local Area Network (WLAN), Wireless Sensor Network (WSN), and Bluetooth remain limited in terms of cost, accuracy, security, complexity, and scalability. As a result, vision-based SLAM methodologies, called visual SLAM , have become the mainstream of current research.

Visual SLAM algorithms need to be more robust to cope with the complex and dynamic parameters of the urban environment. Unlike mobile robots on which most SLAM methods developed have been evaluated, autonomous vehicles have more challenging parameters to consider if autonomous driving is desired. These challenges include the environment's size, loop closure, and data association, which are among the problematic parameters that appear in more dynamic environments, such as those found in urban spaces [26]. For this purpose, the design process needs to consider several parameters starting from the sensors to the embedded architecture. Moreover, the emergence of low-power heterogeneous embedded architectures provides a great opportunity to explore the potential of pushing the processing closer to the sensor and ensuring on-the-fly processing.

Objectives and Contribution

This thesis aims to design an RGB-D SLAM system for autonomous vehicle applications. In this context, several approaches have been proposed by the research community. We build on the HOOFR-SLAM algorithm and extend its use to RGB-D sensors to improve its performance in terms of localization accuracy and processing time. The choice of HOOFR-SLAM is motivated by its accuracy evaluated on numerous datasets and by its low complexity. As our system is devoted to autonomous vehicles, several factors are considered in the design, including the sensor specifications, the environmental dynamics, and the processing rate. This thesis presents four contributions:

- In the first contribution, we propose a methodology of sensor-algorithm coupling, performed on one of the most well-known state-of-the-art algorithm ORB-SLAM2, based on our indoor dataset. This methodology involves exploring and evaluating

different acquisition modalities, identifying the parameters correlated to the sensors, and applying an optimization protocol.

- The second contribution aims to extend the HOOFR-SLAM to RGB-D sensors in outdoor environments. We address common problems cameras face in outdoor environments, such as scenes with a dominant sky and dynamic objects. We propose a method of keypoint filtering based on the depth map and optimize the pose computing algorithm.
- To meet the two objectives mentioned above, we have realized several datasets in indoor and outdoor environments to evaluate the different SLAM methods and the optimization protocol.
- The fourth contribution is an optimized partionning of the HOOFR-SLAM front-end (including HOOFR extractor and features matching block) on a heteregeneous architecture (CPU-FPGA), to achieve on-the-fly processing and meet real-time constraint.

Manuscript Organization

The manuscript is organized as follows:

- Chapter 1 : In this chapter, we give an overview of SLAM systems. We outline the various sensors used in the SLAM. We discuss the importance of sensor characterization to achieve an optimal SLAM design. Next, we introduce visual SLAM systems in detail and the various architectures used to implement them.
- Chapter 2 : This chapter presents the methodology adopted in this thesis, beginning with the choice of the sensor, then the selection of algorithms, the indoor and outdoor data acquisition method, and finally, the choice of the architecture and the High-Level Synthesis approach used for the hardware design.
- Chapter 3 : This chapter presents the sensor-algorithm coupling method applied to a state-of-the-art algorithm. The algorithm is evaluated on our dataset. The impact of

the different acquisition modalities on the quality of trajectory and the importance of sensor-algorithm coupling are discussed.

- Chapter 4 : This chapter is dedicated to the extension of the HOOFR-SLAM to use RGB-D sensors. We propose an approach based on the depth map to filter the key-points detected on dynamic objects. For faster processing, we suggest optimization of the pose estimation block. Finally, we evaluate the algorithm on our outdoor dataset using a PC and an embedded architecture.

- Chapter 5 : This chapter presents an implementation based on the bucketing-based HOOFR extractor on FPGA, with which we incorporate a matching block while ensuring real-time processing at the rate of the sensor.

Finally, we summarize the work done in this thesis and give our research perspectives.

Chapter 1

SLAM Systems

1.1 Introduction

Over the last few years, researchers have devoted their efforts to a concept used by autonomous robots and vehicles called Simultaneous Localization And Mapping, or SLAM. SLAM accomplishes its goal by generating a map and estimating the robot's position in an unknown environment. Regarding the design of SLAM systems, one of the most critical and complex challenges is to make the most of the onboard sensors' ability to accurately interpret their surroundings while respecting real-time and hardware resource constraints. Due to the sensor's low cost, compact size, and ease of usage, the camera-based SLAM algorithm is the most extensively utilized. State-of-the-art describes various techniques and strategies for implementing visual-based SLAM systems. This chapter examines the most widely employed sensors and the underlying ideas behind SLAM algorithms. Next, we will dive into the most commonly used visual-based SLAM techniques (monocular, stereo, RGB-D), providing an overview of the fundamental algorithms and highlighting the significant methodology benefits and drawbacks. Furthermore, we talk about the different hardware architectures used, how the functional blocks of these algorithms could be split up on these architectures, and how well they perform. Lastly, we discuss various datasets used to evaluate the visual SLAM algorithm.

1.2 SLAM Systems

A SLAM system aims at estimating the state of a robot equipped with onboard sensors and reconstructing a map of an unknown environment simultaneously. In general, the state of a robot is described by its pose (position and orientation). Other quantities, such as speed, acceleration, calibration parameters, and sensor biases, can be included. On the other hand, the map is a model of the robot's operating environment describing interesting aspects such as the positions of landmarks and obstacles. The map is used mainly for path planning and localization. Furthermore, the map allows for correcting the localization error when revisiting known places, also known as loop closure.

For many years, SLAM has been the topic of technical research. SLAM is employed extensively for practical applications in many fields, thanks to significant advances in computing speed and the availability of low-cost sensors. Unmanned aerial vehicles (UAVs) [27, 28, 29], robots for indoor service, Virtual/Augmented reality systems (VR/AR) [16, 30], and autonomous vehicles [31, 32, 20] are just a few of the applications for this field of research that have gotten a lot of interest.

SLAM's history started with filter-based solutions. This era is known as the classical age (1986-2004). In this period, the SLAM problem was formulated in the 1980s [33, 34, 35], and probabilistic formulations were introduced, including approaches based on Extended Kalman Filters (EKF), and particle filters [36, 15]. Recently, modern SLAM systems have adopted an optimization-oriented approach. This approach led to better localization accuracy and lower memory utilization compared to filtering-based methods [37].

In the state-of-the-art, a SLAM system is generally decomposed into two blocks as shown by Figure 1.1: the front-end block, which handles the sensors' signals, and the back-end block, which is sensor-agnostic and in charge of optimizing the pose and the map.

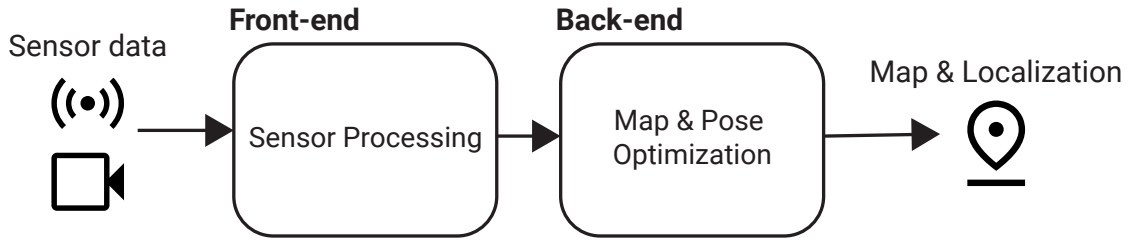

Figure 1.1: General structure of the SLAM system.

With the emergence of autonomous vehicles, simultaneous localization and mapping have garnered more attention. The rivalry for the most accurate system that fulfills real-time constraints has risen, especially in light of the added complexities of high dynamics and the large-scale outdoor environment. Several SLAM systems have been developed with different approaches and different hardware. Hereafter state-of-the-art methodologies are categorized following by sensor type, front-end processing type, and back-end processing type as in [37, 38].

1.2.1 Sensors

Several sensor technologies are integrated in SLAM systems, alone or in a multi-sensor configuration. Such sensors may contribute in two perception modalities: proprioceptive and exteroceptive. Proprioceptive sensors measure the ego-vehicle's state, while exteroceptive sensors gather information about the environment.

1.2.1.1 Proprioceptive Sensors

The most commonly used proprioceptive sensors in SLAM systems is the Inertial Measurement Unit (IMU). The inertial measurement unit (IMU) integrates a gyroscope, an accelerometer, and sometimes a magnetometer. It offers measurements of angular velocity (provided by a gyroscope) and acceleration (provided by an accelerometer) along the x, y, and z axes, in addition to the magnetic field that surrounds the instrument (magnetometer). Although IMU is more accurate in the short term and can provide continuous data at a very high rate (at several hundred Hz), its major drawback is its performance degradation over time. Such sensors' data are often combined with other data sources such

as cameras and Light Imaging Detection and Ranging (LiDAR). Several SLAM methods use the IMU to increase localization accuracy. For instance, Visual Inertial-SLAM is one of those approaches. According to sensor fusion type, VI-SLAM may be loosely or tightly coupled [39]. The loosely coupled approach process the IMU and camera data separately and use both information to track the pose. Instead, the tightly coupled method combines directly visual and inertial raw data [40]. As an illustration of loose coupling, Weiss et al. [41] proposed an inertial-optical flow module for pose initialization and a fallback option when a keyframe-based VSLAM module's tracking fails. In SOFT-SLAM, Cvisic et al. [42] employ IMU to reduce computation complexity by eliminating outliers and substituting 5-point RANSAC with 1-point RANSAC. The recent research is focused on a tightly-coupled approach. VI-ORB-SLAM [43] uses a visual-inertial initialization to estimate accurate states including sensor pose, velocity, and IMU biase before fixing states by tracking and local Bundle Adjustment (BA). They estimate the gyroscope's bias, approximate the scale and the gravity without considering accelerometer bias, and then estimate the accelerometer bias with scale and gravity direction refinement and, finally, the velocity vector. ORB-SLAM3 [44] improved on this method by offering a quick and accurate IMU initialization mechanism and extended to monocular-inertial and stereo-inertial SLAM using pinhole and fisheye cameras. Adding an IMU may improve the environment's information density and accuracy, increasing the algorithm's complexity, particularly during the initialization phase (15 seconds to converge within 1% scale error [44]). In addition, VI-SLAM has exhibited significant performance in indoor environments. However, its performance in outdoor spaces and on long trajectories is still minor compared to that of Visual-only SLAM [45].

1.2.1.2 Exteroceptive Sensors

In the exteroceptive category, the most commonly used detectors are Global Navigation Satellite System (GNSS), RADAR (short for Radio Detection and Ranging), LiDAR (short for Light Imaging Detection and Ranging), and cameras. The Global Navigation Satellite System (GNSS) is based on satellites that continually broadcast a radio signal containing the current time and data about their position. Each GNSS receiver needs four

satellites to figure out four unknowns: X, Y, Z, and the difference between the clock's time and the GNSS reference time. Coordinates are generally expressed using the ellipsoidal World Geodetic System of 1984 (WGS84) model. The 3D coordinates can be expressed either in global systems such as Earth-Centered Earth-Fixed (ECEF) and geodetic systems or in local systems such as East-North-Up (ENU), North-East-Down (NED), and Azimuth-Elevation-Range (AER). Global systems describe the position of an object using a triplet of coordinates. In contrast, local systems require one triplet to describe the origin's location and the other triplet to describe the object's location with respect to the origin. The requirement for clear line-of-sight visibility to the satellites is a limitation of GNSS in general. Buildings and dense trees block many satellite signals, limiting satellite availability. Liu et al. [46] used an inaccurate Global Positioning System (GPS) with a monocular SLAM to fix the temporal GPS drift problem by relating the vision-based camera pose estimation from SLAM to the position information received through GPS in the pose optimization step. Hening et al. [29] proposed an Adaptive Extended Kalman Filter (AKF) for estimation of the velocity and position of a UAV by fusing LiDAR SLAM local position updates, GPS corrections, and an Inertial Navigation System (INS).

Radar is a sensor that identifies the existence of a distant object, its size, velocity, and direction by sending radio waves and detecting changes in the reflected wavelengths or the frequency difference between the transmitted and received signals. RadarSLAM [47] uses Frequency-Modulated Continuous-Wave (FMCW) technology, a type of radar whose transmitter sends waveforms continuously, and the receiver waits for the echo reflected from the targets. Although this technology is robust to weather conditions, it is more susceptible to noise than LiDAR. This technology operates well in variable lighting and weather conditions indoors and outdoors, providing excellent reliability and consistency, it has lower resolution and updates measurements less often than LiDAR and cameras. Its output frequently includes spurious detections (i.e., false detections) and other undesirable artifacts [47].

LiDAR, like RADAR, is a ranging technology. It emits infrared light pulses instead of radio waves and then measures the distance of the scanning points from its center by the time-of-flight method. Every second, LiDAR gathers millions of accurate measurement points, from which a 3D map of its surroundings can be created. LiDAR is unaffected

by variations in ambient light and works well in all low-light situations. LiDAR may also be utilized both indoors and outdoors. Unlike RADAR, LiDAR has limitations in adverse weather conditions (snow, rain, fog). One of the most popular LiDAR-based SLAM systems is LOAM [48], a low-drift and read-time odometry and mapping method based on LiDAR data. In this algorithm, the authors extract features by determining the roughness of a point in its local region. The main novelty in this approach is dividing the problem into two parallel threads. The first one estimates the velocity with low accuracy while running at a high frequency. The second one maps the environment with higher fidelity but at lower frequency. Finally, both estimates are fused. An extension to this method was proposed in LeGO-LOAM [49], where the authors presented a lightweight and ground-optimized LOAM for the pose estimation of ground vehicles. This strategy has three main advantages over LOAM. First, it segments ground points which helps to remove unreliable features. Second, it uses a two-step Levenberg-Marquardt (LM) method to speed up the optimization process. Finally, it integrates the ability to perform loop closures to correct the motion estimation drift.

Camera-based SLAM is the most attractive system due to its low cost, ease of configuration, and compact size. We found standard, RGB-D, and event cameras in the family of visual sensors. In monocular SLAM, a single camera moving through its environment is used to estimate the pose by detecting and tracking salient spots using image processing algorithms. The main drawback of monocular cameras is the drift of the scale factor when calculating distances between points due to the lack of depth information.

Inspired from the human eyes, a stereoscopic camera uses two monocular cameras mounted on a platform at a fixed distance (called baseline) with an overlapping view. A stereo camera is used to compute scene depth information by matching and triangulation. However, its range is influenced by the baseline. An increased baseline will increase the depth range and the minimum distance to the camera.

The RGB-D cameras save us all expensive computing done in stereo approach and provide us directly with a depth image associated with an RGB image. Different technologies are used in RGB-D cameras, such as stereo-vision, structure-light, or time-of-flight. However, this type of sensor suffers from limited range and sensitivity to sunlight. The new

generation of RGB-D cameras is characterized by a wide range thanks to a broad baseline and an auto-projector that improves the depth calculation in low textured environments, allowing indoor and outdoor operation.

From an energy standpoint, an event camera is the best option. Unlike standard cameras, which acquire entire pictures at a rate determined by an external clock, event-based cameras asynchronously capture per pixel intensity variation and provide the output variable data-rate sequence of digital events. Each event is encoded with information, including the triggered time, pixel localization, and the sign of the intensity change. Event cameras have substantial benefits such as high temporal resolution, low latency, low power, and High Dynamic Range (HDR) [50, 51]. Nevertheless, they are still expensive [52]. Most of the methods and concepts used in visual SLAM were developed for intensity images (feature detection, matching, etc.) and do not apply to a sequence of asynchronous events. Therefore, the challenge is to design new SLAM techniques that can take advantage of the benefits of event-driven cameras. In the last decade, research works have been conducted to design event-based SLAMs, and they are still in their early stages of research [37]. Hence, in visual SLAM, several approaches are proposed in the literature and classified into three categories based on the sensor type: monocular, stereo, and RGB-D SLAM.

1.2.2 Sensor Characterization

Before using a sensor, the assessment of its capacities and limitations is crucial for an accurate system design. For example, Schops et al. [53] proved that direct RGB-D SLAM systems are highly sensitive to rolling shutter, RGB and depth sensor synchronization, and calibration errors. Andreopoulos et al. [54] proposed an evaluation of the effects of camera shutter speed and voltage gain under simultaneous changes in illumination and demonstrated significant differences in the sensitivities of popular vision algorithms under those variations. Wu & Tsotsos [55] showed the sensor bias of vision algorithms that requires a finer control of camera parameters to make these algorithms functional in real-world applications. In Chapter 3, we investigate the effect of sensor acquisition modalities on localization accuracy and provide a parametric optimization technique to enhance localization accuracy in a given environment.

1.2.3 Visual-based SLAM Systems

Visual SLAM systems are the most common type in the literature. Visual SLAM has undergone several improvements, starting with the keyframe-based parallel tracking and mapping (PTAM) solution by Klein et al. in 2007 [16]. The Visual SLAM has become more reliable by incorporating efficient loop closure, global optimization, and memory management methods such as keyframe and culling, with real-time processing achieved through multithreaded parallelization.

1.2.3.1 Visual SLAM system formalization

Typically, Visual SLAM systems consist of two blocks: the front-end and the back-end, as represented by Figure 1.2. The front-end starts by processing the data from the sensor by considering the entire raw image (the direct method) or by considering only points of interest known as keypoints (feature-based approach). Next, matching data is performed to compute the pose between two consecutive frames. Then, the initialization of the trajectory is done by defining the global coordinate system and providing an initial estimation of the variables to be optimized through the back-end block. This latter employs algorithms based on filtering or optimization to reduce the cumulative error and increase estimation accuracy. The loop closure identifies the places previously visited to estimate and correct the drift accumulated during the sensor movement between the pose of a previously visited place and the current pose. The process starts with place recognition. Most of the place recognition method compares new keyframes with a database of previously obtained views using a bag-of-words approach, as the DBoW2 method proposed by [56]. When a potential similarity has been identified, multiple verification stages are used to decide whether or not it corresponds to a loop. The loop closure then corrects the map and poses. Closing a loop can be computationally intensive. Therefore, it is typically performed in a separate thread.

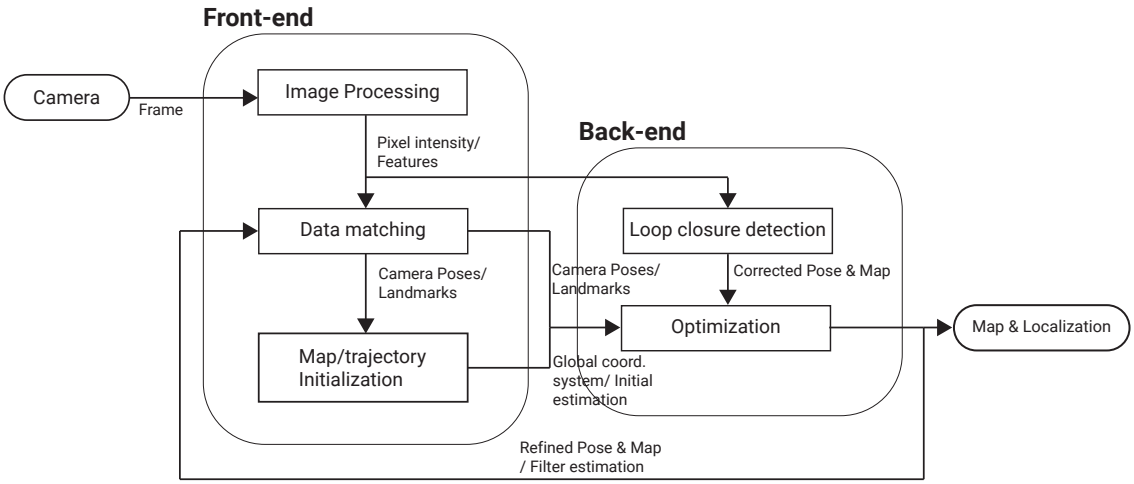

Figure 1.2: Visual-based SLAM diagram

Front–end

The visual SLAM front-end consists of three steps: Image processing, data matching, and map/trajectory Initialization. The image processing step consists of extracting the valuable data contained in frames. This data will be used to infer the map structure and the trajectory. There are two types of image processing: direct and feature-based (indirect).

- The direct approach exploits the raw image pixels' intensities to minimize the photometric error. It can be categorized as dense or semi-dense. Semi-dense methods employ only pixels with a brightness gradient above a defined threshold, whereas dense methods use every pixel.

- The feature-based approach relies on detecting points of interest in the image, associating a description to each point, and finding the match between different points on successive frames. To ensure good tracking, the extractor must provide keypoints that verify repeatability and uniqueness [57]. Famous descriptors include SURF [58], SIFT [59], ORB [60], and HOOFR [57]. Feature-based methods are robust to geometric distortions due to rolling shutter, automatic exposure changes, and lens vignetting [39]. Historically, the feature-based approach exhibited low computational complexity, making it suitable for embedded systems [61, 62, 10].

Image processing methods can be associated with different map representations. These include sparse maps consisting of a cloud of sparse features, dense maps which use the full image information, and semi-dense maps which use a dense representation in specific zones of interest. The most common approaches are indirect/sparse and direct/dense or semi-dense.

After the image processing step, features or dense raw information are fed to the data association bloc to establish the correspondences used to compute the pose between two successive frames. The matching step, in turn is classified as a direct and indirect method.

- The direct method is based on brightness consistency constraint, it aims to find the true motion that minimize the overall photometric difference in the image, which is a sum of pixel-wise photometric errors.
- Indirect methods can be applied by matching 2D features detected on successive frames (2D-2D), by matching given a set of 3D points in the world with their corresponding 2D projections in the new frame (2.5D), or by matching 3D points in the world frame with the reconstructed map (3D-3D). This latter is more prone to uncertainties than 2.5D, while 2D-2D is the standard solution for pure Visual Odometry (VO) methods [39].

Finally, the map and trajectory initialization determine the global coordinate system and provide an initial set of map points by computing spatial transformation between successive frames. Map initialization is mandatory for monocular SLAM as the depth cannot be recovered from a single image. For example, ORB-SLAM computes two geometric models in parallel: a homography assuming a planar scene and a fundamental matrix assuming a non-planar scene. The model is then selected using a heuristic based on symmetric transfer errors [63, 1]. Other SLAM algorithms, such as LSD-SLAM, use a random initialization, the algorithm initializes the first keyframe with a random depth map and large variance, and after a couple of keyframes, the algorithm converges to a correct depth configuration [3].

Back-end

This block represents the core of the SLAM. Modern SLAM uses optimization-based methods that perform batch processing in contrast to filter-based approaches, which correspond to iterative processes suited to online SLAM. Optimization-based SLAM can be divided into two approaches: Bundle Adjustment (BA) and Graph SLAM.

Bundle Adjustment (BA) is a technique that simultaneously refines a 3D structure and the camera pose, given a sequence of images presenting several 3D points from different viewpoints. The concept used in SLAM consists of optimizing an objective function Equ.1.1, which minimizes the reprojection error, using the Levenberg-Marquardt algorithm [64]. The bundle adjustment (BA) formulation is:

$$\min_{\{X_i\},\{(R_j,t_j)\}} \sum_{i,j} \left\| u_{i,j} - f(X_i|R_j,t_j) \right\|_2^2 \tag{1.1}$$

where $u_{i,j}$ is a set of observations, $\{X_i\}$ is the 3D coordinates of the scene points, $\{(R_j,t_j)\}$ are the 6DOF poses of the images I_j and $f(X_i|R_j,t_j)$ is the projection of X_i onto I_j (assuming calibrated cameras) [65].

This optimization can reduces the reprojection error resulting in the best camera and landmark positions. However, the computation can be expensive due to the optimized variables dimension [66].

Graph SLAM models the SLAM problem using a graph. Nodes represent the trajectory and the landmark map. The sensor measurements are associated with Gaussian noise and give spatial constraints between the nodes. Edges model such spatial constraints. Among them two types are distinguished:

- Motion edges connect two consecutive pose nodes;
- Observation edges connect landmarks to pose nodes if observed from it.

GraphSLAM consists of re-estimating, in addition to the current position, the whole trajectory from the initial state by considering all the measurement history. Once the graph is built, the system's state can be estimated via a global optimization of the graph. The

optimization consists in finding a configuration (trajectory and map) that best fits the constraints introduced by the edges. This optimization problem can be solved by finding the minimum of a cost function that follows this form Equ.1.2.

$$F(x,m) = \sum_{ij} e_{ij}(x,m)^T \Omega_{ij} e_{ij}(x,m) \tag{1.2}$$

where x is the vector of poses, m is the map, $e_{i,j}$ is the error function which computes the distance between the expected observation and the real observation and Ω_{ij} is the associated information matrix between the node i and the node j. In practice, the minimization of $F(x,m)$ is typically solved by a local approximation using common methods like Gauss-Newton or Levenberg-Marquardt [67, 66].

1.2.3.2 Sensor-based Visual SLAM classification

The V-SLAM may be categorized based on the visual sensor employed. Three varieties of V-SLAM are considered: Monocular, Stereo, and RGB-D. Since our thesis focuses on RGB-D SLAM systems, and most of existing algorithms are based on monocular implementations, it seems worthwhile to review the different works based on the monocular approach and their evolution towards stereo and RGB-D-based systems. The algorithms to be discussed are presented on the timeline in Figure 1.3.

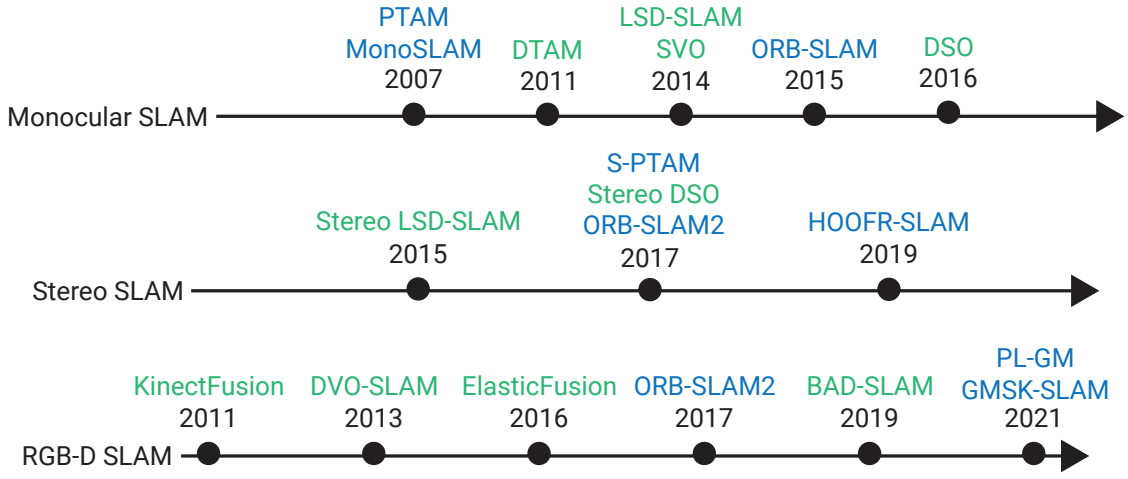

Figure 1.3: Timeline representing well-known SLAM systems and their chronological evolution. In blue, feature-based V-SLAM methods. In green, direct approaches.

Monocular SLAM

The first SLAM proposed in this category was **MonoSLAM** (2007) [15]. This method uses a feature-based approach on the front-end and an extended Kalman filter on the back-end. The algorithm operates in real-time, and the literature has several embedded implementations based on this algorithm [68, 69]. However, its complexity increases proportionally to the size of the environment.

The key breakthrough in V-SLAM was the introduction of **Parallel Tracking and Mapping (PTAM)** (2007) [16], a feature-based algorithm. It is the first algorithm to separate tracking and mapping into two threads running in parallel and the first to use the concept of keyframes. Although, this algorithm uses the concept of keyframes to reduce computational consumption. It is worth noticing that BA optimization entails a high complexity and requires a high-performance computing system. That makes this approach unsuitable for low-cost and low-power embedded systems due to its significant power consumption [70].

Inspired by PTAM, **ORB-SLAM** (2015) [1] uses three parallel threads: tracking, local mapping, and loop closing. Figure 1.4 represents the diagram block of ORB-SLAM. The tracking thread keeps track of features by finding matches and minimizing the reprojection error by applying motion-only Bundle Adjustment. The local mapping manages and optimizes the local map using a local BA. The thread loop closing detects loop closures and corrects the accumulated drifts by performing the graph-pose optimization. Finally, to ensure the consistency of the whole structure and the estimated motion, the algorithm applies a full BA.

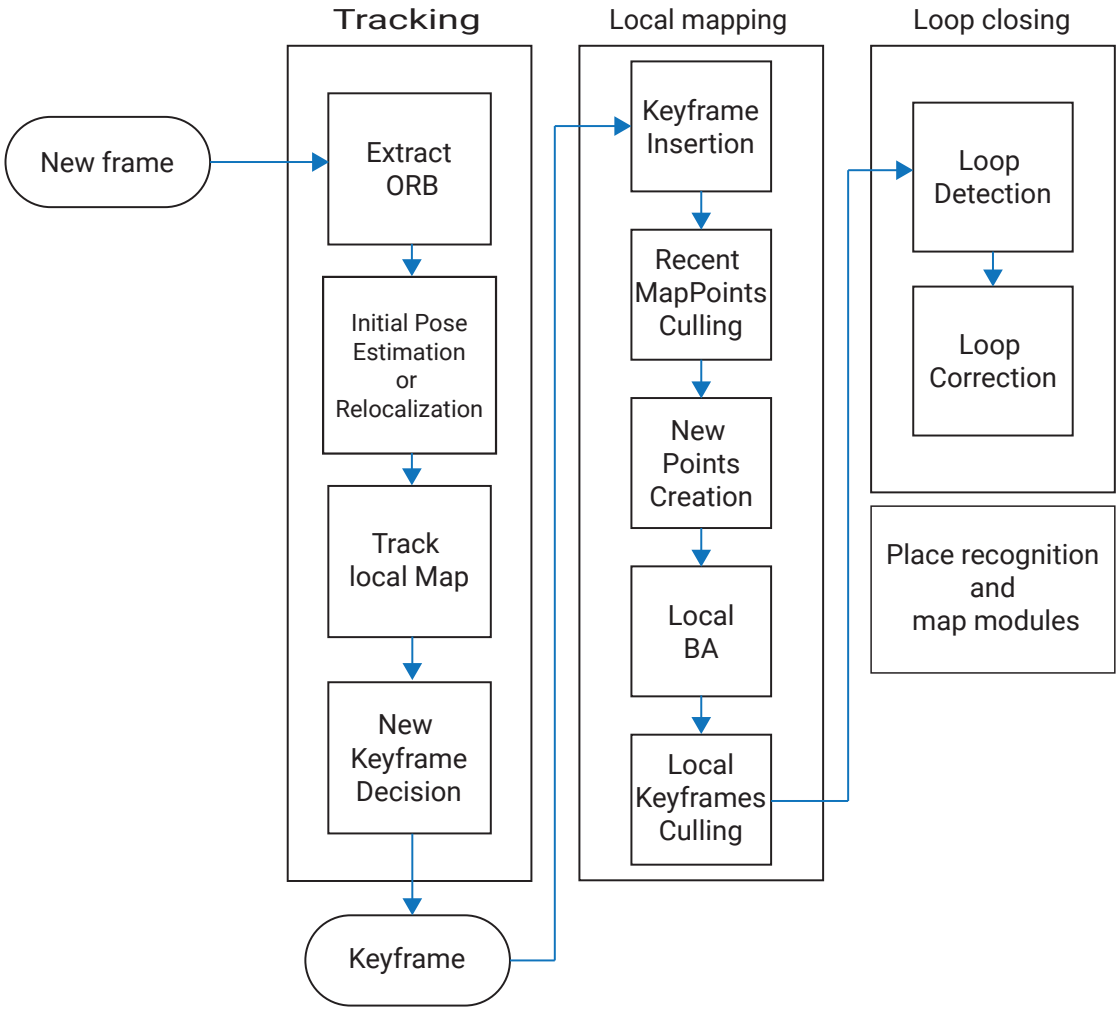

Figure 1.4: Diagram representing the ORB-SLAM algorithm [1]

In the category of direct approaches, **Dense Tracking and Mapping (DTAM)** is the first direct algorithm proposed by Newcombe et al. in 2011 [17]. The algorithm comprises two main blocks: Dense Mapping and Dense Tracking. In a dense mapping block, a global energy minimization framework estimates the inverse of the depth map. The energy function is composed of the sum of photometric errors and a robust spatial regularization. The dense tracking block deals with the alignment of an image of the dense model projected into a virtual camera and the current image to estimate the motion parameters. This method is computationally intensive and can only be implemented through extensive GPU parallelization [2]. Also, the algorithm assumes brightness constancy in all reconstruction stages, which makes the algorithm not robust to real-world global illumination changes.

A precise and faster algorithm was proposed by Forster et al. [2]; **Semi-Direct Visual Odometry (SVO)** (2014) uses a hybrid approach combining the feature-based and direct methods. Like PTAM [16], it uses two parallel threads as shown in Figure 1.5, one for estimating the camera motion and the other for mapping. In the mapping thread, for each 2D feature corresponding to the 3D point to be estimated, a probabilistic depth-filter is initialized with a large depth uncertainty, and at each incoming frame, the estimated depth is updated. Once the uncertainty is small enough, the new 3D point is inserted into the map and used for the motion estimation. SVO can run at high rates since feature extraction and matching are not required for motion estimation, making it possible to be embedded on low-cost embedded systems. However, being a pure VO approach, it only performs short-term data association, limiting its accuracy [44].

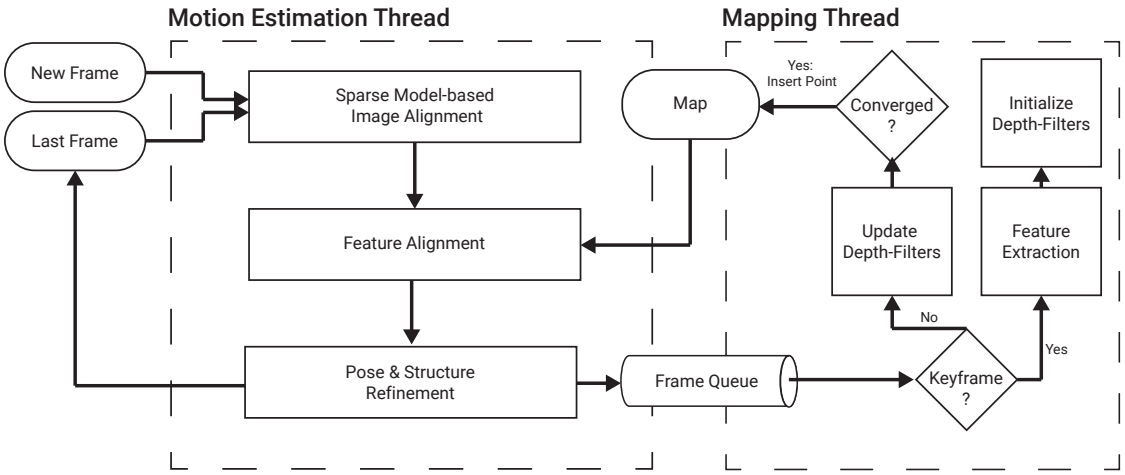

Figure 1.5: Semi-Direct Visual Odometry (SVO) tracking and mapping pipeline [2]

Large-Scale Direct Monocular SLAM (LSD-SLAM) (2014) [3] is a direct algorithm that tracks camera motion and builds a semi-dense map of a large-scale environment. The Figure 1.6 shows the three main components of the algorithm: tracking, depth map estimation, and map optimization. The tracking estimates the pose of each image. In the depth map estimation, the tracked frames are used to replace the current keyframe or to refine the current keyframe's depth by using several small-baseline stereo comparisons. The final step is map optimization, which adds the new keyframe to the map and optimizes the pose graph. Although, LSD-SLAM can build semi-dense maps of large-scale environments, its accuracy is less than that of PTAM and ORB-SLAM [44].

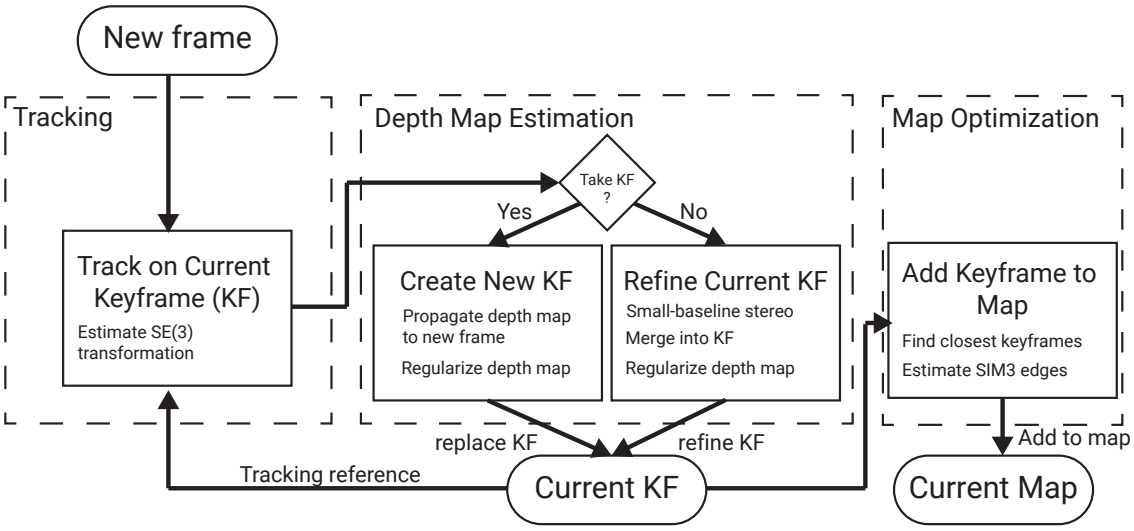

Figure 1.6: Diagram summarizing the LSD-SLAM algorithm [3].

LSD-SLAM has been expanded to **Direct Sparse Odometry (DSO)** (2016) [71], where it applies a local photometric bundle adjustment on a sliding window of keyframes and the inverse depth map. Xiang et al. [72] extended DSO to a monocular visual SLAM system with loop closure detection and pose-graph optimization (LDSO).

Stereo SLAM

As we saw, visual SLAM can be done using only a monocular camera. Since depth isn't observable from one camera, the map's and estimated trajectory's scale are up to an unknown global scale factor. The system bootstrapping requires multi-view or filtering approaches to create an initial map, which cannot be triangulated from the first frame. Also, monocular SLAM suffers from scale drift and may fail in rotations. Using a stereo or RGB-D camera overcomes all these concerns and makes Visual SLAM more reliable. Most of the stereo algorithms we will mention in this section are based on the above monocular versions.

Based on its monocular version, **Stereo LSD-SLAM** (2015) [73] combines temporal and static stereo allowing multiple baseline directions. The authors used a modified cost function similar to the normalized cross-correlation (NCC) invariant to affine lighting changes to solve the violated brightness constancy assumption in the real world. **Stereo DSO** (2017) [74], also based on its monocular version DSO, uses the combination of temporal

and static stereo. **S-PTAM** (2017) [75] integrates stereo constraints in initialization, mapping, and tracking to improve accuracy and robustness. **ORB-SLAM2** (2017) [76] has extended its monocular version to support stereo and RGB-D cameras. The stereo version extracts ORB features in the rectified stereo image pairs. Then, matched keypoints of the image pairs are selected as stereo keypoints and classified into close or far depending on whether their associated depth is less than a threshold related to the baseline distance. Close keypoints can be triangulated to estimate depth, scale, translation, and rotation information, while far keypoints provide accurate rotation information. This way, camera poses are estimated and optimized using motion-only BA. **HOOFR-SLAM** (2019) [8], a recent feature-based algorithm with competitive performance, exploits the HOOFR extractor for feature detection and matching [57]. The HOOFR-SLAM uses stereo images to compute the depth of the keypoints. The stereo images are assumed rectified, and the disparity computation is performed using the five pixels-SSD method. The scale factor is then obtained by applying a 1-point scheme to the different factors computed from the ratio of static and temporal stereo. HOOFR-SLAM implements a processing structure that maximizes parallelism and avoids the need to optimize camera poses by applying bundle adjustments on keyframes or saving the history of map points by estimating the relative poses of the current input frame with a set of previous neighboring frames. The optimal pose is obtained by averaging the relative poses with weighted factors.

Traditional visual SLAM systems use a monocular or stereo camera as input, which involves complex initialization of the map and computationally intensive triangulation steps of the map points required for 3D map reconstruction. These problems were solved with the advent of the RGB-D camera, which provides an RGB image and an associated depth map.

RGB-D SLAM

RGB-D cameras can simultaneously provide colored and depth images for all regions in the field of view with or without textures, making dense reconstruction straightforward and removing the need for map initialization. Also, the RGB-D camera is an excellent asset for embedded SLAM systems since computing the depth map of stereo images is

computationally intensive [77, 78]. The emergence of RGB-D sensors has motivated researchers to develop innovative SLAM systems.

The KinectFusion (2011) [4] was the first direct RGB-D camera method. The method generates a dense vertex and normal map pyramid using the raw depth. Each frame's vertex map and normal map are used to build a global model represented by a volumetric, truncated signed distance function (TSDF). The pose estimation is then performed using Iterative Closest Point (ICP) alignment between the current surface and the predicted one. An overview of the algorithm's steps is shown in Figure 1.7. The algorithm is limited to small workspaces and accumulates drift errors due to the lack of loop closing [79].

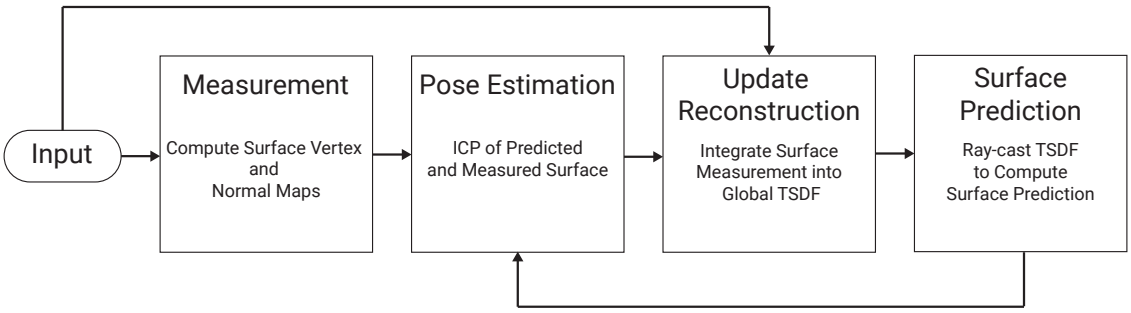

Figure 1.7: KinectFusion diagram [4]

Dense Visual Odometry (DVO-SLAM) (2013) [80] is a dense direct SLAM that minimizes the photometric and depth error for pose calculation. The algorithm uses an entropy-based method for keyframe selection and loop closure, significantly reducing the drift. The map is represented by a pose graph optimized using the g2o framework [81].

Shifting away from the focus on pose graphs initially founded on sparse methods, **ElasticFusion** (2016) [82] is a map-centric system reconstructing a surfel-based map. By incorporating many small local model-to-model loop closures in conjunction with larger-scale global loop closures, the algorithm produces globally consistent reconstructions without using pose graph optimization. Although the reconstruction is well detailed and the localization is pretty accurate, the algorithm is limited to room size maps as the complexity increases with the number of surfels [76].

The environment scale issue has been solved for RGB-D systems by extending ORB-SLAM to the use of RGB-D sensors. **ORB-SLAM2** (2017) [76] uses depth information to

synthesize a stereo coordinate, so the system is agnostic of the input being stereo or RGB-D. They demonstrated that using RGB-D with bundle adjustment performs better than direct methods or ICP, as well as being less computationally expensive and not requiring GPU processing to run in real-time.

On the other hand, Sch et al. [83] showed that their direct approach, **BAD SLAM** (2019), outperforms ORB-SLAM2 on their RGB-D SLAM benchmark with synchronized global shutter cameras, consequently stating that existing datasets only give a partial picture of the performance of SLAM algorithms. **BAD SLAM** uses surfels and keyframes to represent the map, reducing the amount of data for BA. The front-end part of this algorithm tracks the RGB-D camera's movement in real-time. The back-end refines the camera trajectory and geometry using a direct adjustment bundle using geometric constraints based on depth maps and photometric constraints.

Most indoor SLAM methods assume that the environment is static. A significant problem visual SLAM algorithms face in real life is the dynamic environment. Therefore, many variable factors are involved in the scene, such as lighting, dynamic targets, occlusion, etc. **GMSK-SLAM** (2021) [5] is a method that combines Grid-based Motion Statistics (GMS) feature points matching with the K-means clustering method to distinguish the dynamic areas of the image and keep the static points in these areas. The algorithm is based on ORB-SLAM2 [76] and runs the dynamic area detection thread in parallel with the tracking thread as shown in Figure 1.8.

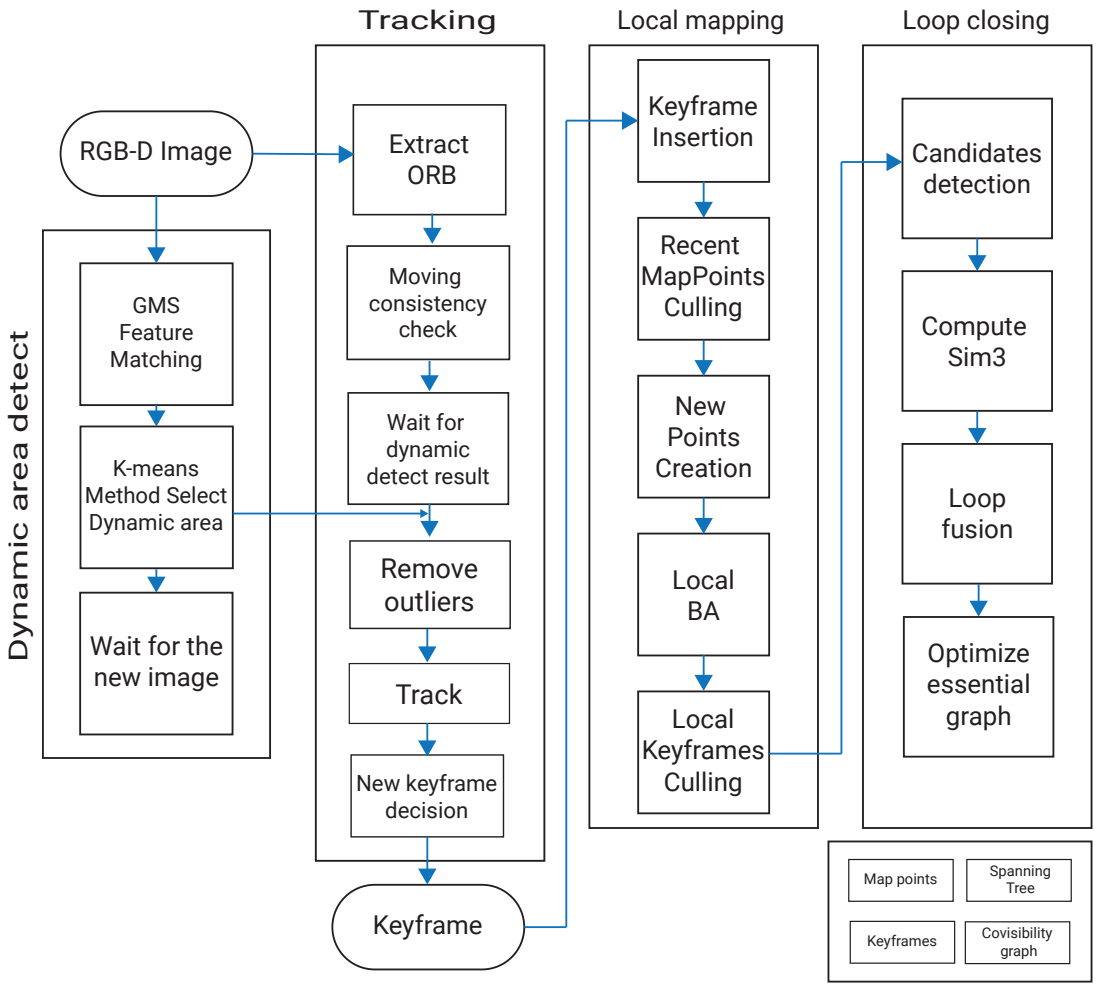

Figure 1.8: The block diagram of GMSK-SLAM [5]

PL-GM (2021) is an indirect SLAM combining point and line feature extraction. The 3D points and lines are retrieved from the depth image and combined with the points and lines features to build a geometric constrained model. Then, the model is extended to a BA model. The algorithm was evaluated on two indoor datasets [84, 85] and a real scenario in a corridor. The algorithm improved the accuracy and robustness, especially in low-textured areas and blurred sequences. Table 1.1 summarizes the main characteristics of the visual SLAM algorithms presented above.

In Chapter 4, we choose for our study the HOOFR-SLAM for two main reasons: Its high accuracy evaluated on many outdoor datasets and its adequacy to be implemented on embedded architectures. Our contribution is extending the HOOFR-SLAM towards RGB-D sensors to increase the localization accuracy and performance.

Algorithm	Year	Sensor	Front-end	Back-end	Loop Cl.	Evaluation Datasets	Platform
MonoSLAM	2007	Monocular	Feature	Filter	No	Indoor	PC
PTAM	2007	Monocular	Feature	Optim	No	Indoor	PC
DTAM	2011	Monocular	Direct	Optim	No	Indoor	PC
KinectFusion	2011	RGB-D	Direct	Optim	No	Indoor	PC
DVO-SLAM	2013	RGB-D	Direct	Optim	Yes	Indoor [84]	PC
LSD-SLAM	2014	Monocular	Direct	Optim	Yes	Indoor/Outdoor [84, 86]	PC
SVO	2014	Monocular	Hybrid	Optim	No	Outdoor [87]	Embedded computer
ORB-SLAM	2015	Monocular	Feature	Optim	Yes	Indoor/Outdoor [84, 88, 89]	PC
Stereo LSD-SLAM	2015	Stereo	Direct	Optim	Yes	Indoor/Outdoor [89, 90]	PC
ElasticFusion	2016	RGB-D	Direct	Optim	Yes	Indoor [84]	PC
DSO	2016	Monocular	Direct	Optim	No	Indoor/Outdoor [91, 92, 90]	PC
S-PTAM	2017	Stereo	Feature	Optim	Yes	Outdoor [89]	PC
Stereo DSO	2017	Stereo	Direct	Optim	No	Outdoor [89, 93]	-
ORB-SLAM2	2017	Mono, Stereo, RGB-D	Feature	Optim	Yes	Indoor/Outdoor [89, 84, 90]	PC
HOOFR-SLAM	2019	Stereo	Feature	Optim	Yes	Outdoor [89, 94, 95, 96, 97]	Nvidia Jetson TX1
BAD-SLAM	2019	RGB-D	Direct	Optim	Yes	Indoor [84], Outdoor [83]	PC
PL-GM	2021	RGB-D	Feature	Optim	Yes	Indoor [84, 92]	PC
GMSK-SLAM	2021	RGB-D	Feature	Optim	Yes	Indoor [84]	PC

Table 1.1: Relevant properties of the visual SLAM algorithms

1.3 Hardware architectures based SLAM applications

SLAM is intended for robotic and autonomous vehicle applications. These targets require an optimal embedded implementation that respects the real-time constraints, limited memory and CPU resources, and energy consumption. SLAM algorithms are computationally intensive to run on embedded targets, and often the algorithms are deployed on laptop-level devices, as shown in Table 1.1. With the growth of powerful embedded heterogeneous computing systems, such as NVIDIA Jetson AGX Xavier, and HERO heterogeneous platform, research work is increasingly interested in the algorithm-architecture mapping of existing SLAM algorithms. Based on the formalization of the visual SLAM discussed in 1.2.3.1, there are two groups of algorithmsembedding studies in the literature. The first focuses on embedding the front-end processing [8, 10, 62], and the second is concerned with the back-end [28, 98]. To bring the processing as near as possible to the sensor, we will focus on works performed on the SLAM front-end.

1.3.1 CPU-GPU based SLAM

The CPU-GPU architectures are widely used in robotics, especially in computer vision, since a GPU can offer many cores for parallel Single Instruction, Multiple Data (SIMD) processing. Based on DTAM [17], Ondruvska et al. [99] exploited the GPU of various mobile phones to implement a pipeline that creates a connected 3D surface model directly on the device in real-time. They assigned sequential tasks, including keyframe selection and dense camera alignment, to the CPU, as the camera alignment requires an accumulation of errors across the entire input image. Also, they used SIMD instructions which led to a processing of 4 pixels at a time. On the other hand, GPU was run in parallel carrying stereo depth computation, model update, and raycasting. Even though this architecture allows volumetric surface reconstruction and dense 6DoF camera tracking in real-time, the GPU hardware constraints limit the voxel resolution.

In the category of indirect approaches, Aldegheri et al. [6] modified ORB-SLAM2 to operate on the Nvidia Jetson TX2 in real-time. Besides the current parallelization of the algorithm on CPU (Parallel PThreads on shared-memory multi-core CPUs and automatic

parallel implementation (i.e., through OpenMP directives) of the bundle adjustment sub-block), the authors have added two layers of parallelism. The first one consists of a parallel implementation of the tracking sub-blocks on GPU. The second is the implementation of an 8-stage pipeline of such sub-blocks. The acceleration targets the feature extraction block as it is the bottleneck of the processing flow. For this purpose, they modeled the extraction block with a directed acyclic graph (DAG), as shown in Figure 1.9, adopting the OpenVX standard.

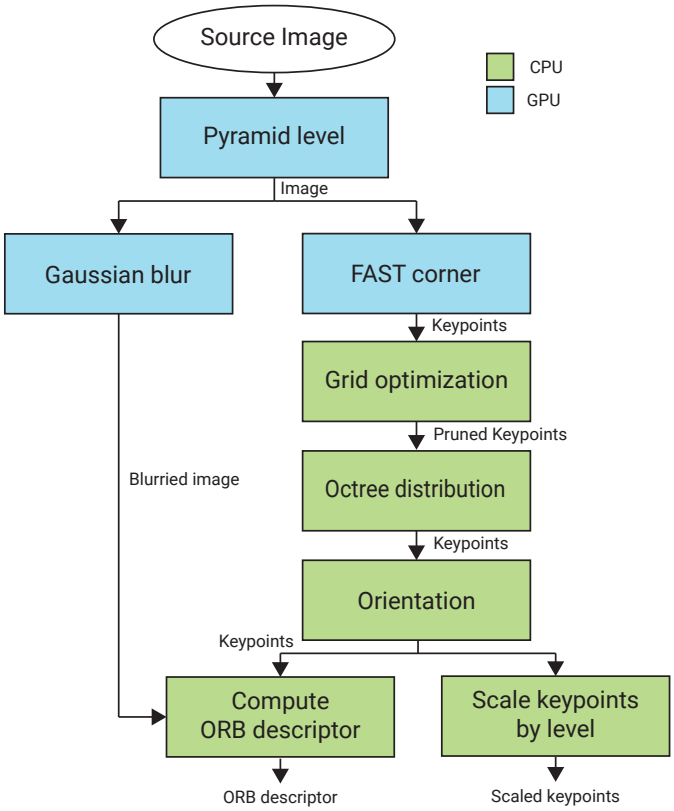

Figure 1.9: Feature extraction block represented by a direct acyclic graph (DAG) and the corresponding sub-block implementations [6]

Ma et al. [7] presented a front-end processing parallelization of the ORB-SLAM2 algorithm on the Jetson TX2. As front-end processing consumes more than half of computing resources and operates on images, so this part is well suitable for parallelization. The parallelization involves the feature extractor and the local point selection. This latter is used to reduce the input data of the matching block. The feature extraction parallelization involves constructing the Gaussian pyramid of the image on the GPU, then feature detection, orientation calculation, and description are performed on the GPU, as shown

in Figure 1.10. Because of the asynchronous operation of CPU and GPU, task and thread allocation were adjusted to decrease the idle time of the GPU and to increase the usage of the streaming multiprocessor (SM).

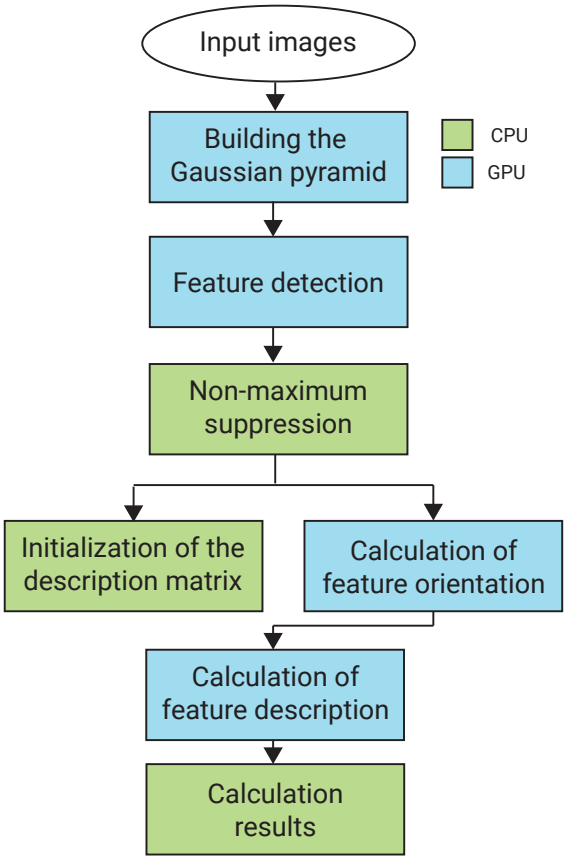

Figure 1.10: Feature extraction parallelization [7]

Nguyen et al. [8] found that the features matching block has a high computational cost and low data dependence, so they proposed to parallelize the HOOFR-SLAM feature matching block on GPU. On the other hand, they employed OpenMP to implement HOOFR feature extraction for two reasons: firstly, the extractor uses FAST detection, where a pixel can be rejected after one or two pixel tests, leading to a difference in processing cost for each pixel, so the GPU computation resources are not well used due to unbalanced complexity. Secondly, the Hessian filtering is much more rapid on CPU thanks to the binary classification, which needs a dynamic memory allocation that is not supported on GPU. Figure 1.11 shows the CPU-GPU mapping of HOOFR-SLAM. The performance obtained on the Jetson TX1 is real-time if we consider the KITTI [89] dataset's acquisition rate.

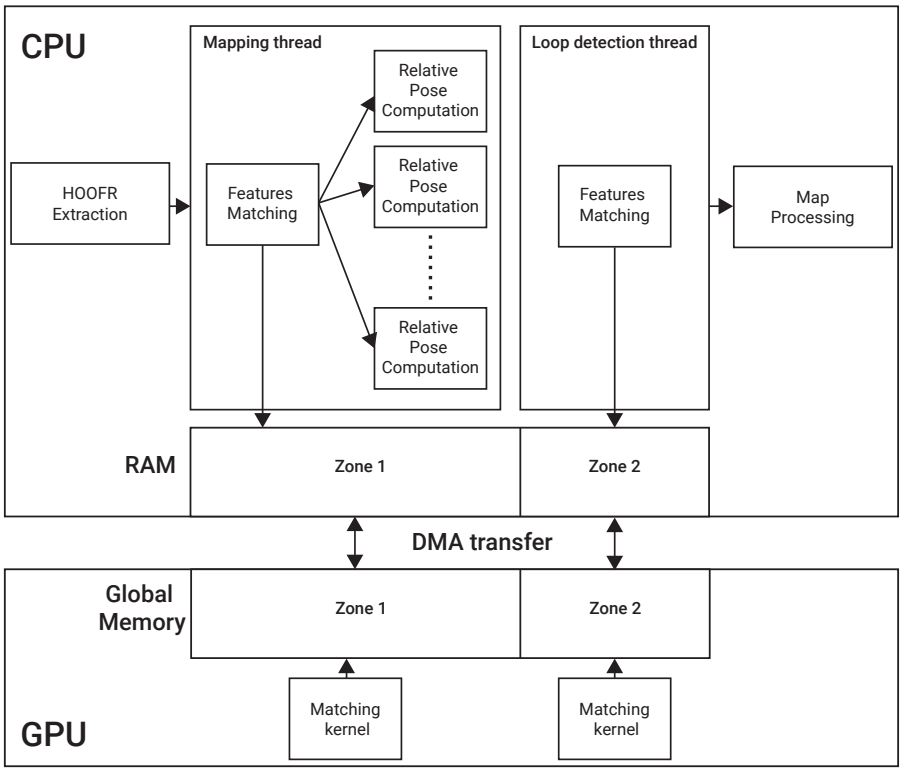

Figure 1.11: HOOFR-SLAM mapping on a CPU-GPU architecture [8]

Although these GPU implementations provide real-time processing, the energy consumption of such architecture is still a challenge for robotics and autonomous vehicle applications, where autonomy is a crucial asset.

1.3.2 CPU-FPGA based SLAM

Recently, CPU-FPGA architectures have gained considerable interest in the scientific community thanks to the advantages of this type of architecture, which include lower power consumption and data flow pipelining, which makes it more suitable for on-the-fly processing applications [100]. Much research on this type of architecture is devoted to the front-end part of the SLAM.

Liu et al. [9] proposed eSLAM, an implementation based on ORB-SLAM on a Zynq platform. The front-end part, including feature extraction and matching, has been accelerated on FPGA and was compared to the ARM processor's version. Figure 1.12 shows the architecture of eSLAM. First, the ORB extractor was reformulated as a rotationally

symmetric pattern for hardware-friendly implementation. To reduce the computation cost of the rotation procedure, they pre-computed the rotated BRIEF patterns and built it as a lookup table to obtain the descriptors when necessary. Moreover, to reduce the extra resources needed to store the lookup table, they proposed a 32-fold rotationally symmetric BRIEF pattern (RS-BRIEF) generated by rotating two sets of seeded locations. Then, a parallelized pipeline mechanism is proposed. They adopted two cases pipeline, standard frame and keyframe case. For standard frame processing, the ORB Extractor and BRIEF Matcher are launched to do feature extraction and feature matching for the next frame, while the ARM processor performs pose estimation and optimization. In the case of keyframe, The feature extraction is performed on FPGA in parallel with the ARM processor, but the BRIEF Matcher is idle until map updating is finished. Despite the acceptable loss of accuracy, the proposed design achieved significant factors in speedup and energy efficiency.

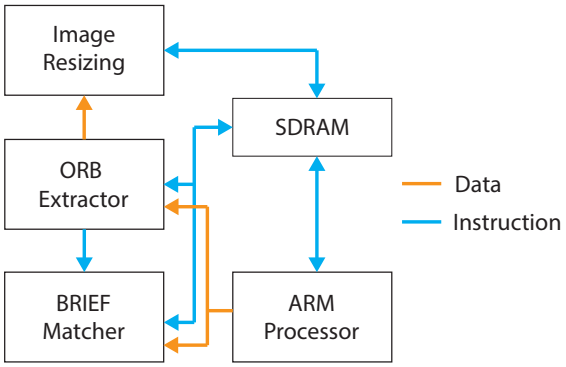

Figure 1.12: eSLAM architecture [9]

Several Visual SLAM systems use semantic information to enhance the robustness of the dynamic scene. Deep learning increases system complexity, making it hard to implement real-time semantic SLAM on a low-power embedded platform. A semantic segmentation module cannot be processed on the CPU in real-time. On the other hand, using a GPU for computing acceleration limits the deployment of battery-powered mobile robots. Wu et al. [101] proposed the acceleration of the inference of the semantic segmentation network, SegNet [102], on an FPGA using the high-level synthesis tool OpenCL [103]. They started by adopting a quantization strategy by combining convolution and batch normalization into one operation and grouping convolution filters with different data distributions. The

quantization operation is processed on the CPU and is used to reduce the needed storage and computational complexity without significantly affecting the precision. The accelerator architecture uses a multi-level memory access optimization scheme, including off-chip DRAM and on-chip memory (channels, registers). The architecture includes the design of convolution kernel, pooling kernel, and unpooling kernel implemented in a configurable pipeline, allowing flexibility and expansibility in implementing different network frameworks. Although the accuracy of the DS-SLAM accelerator slightly decreased as a result of model quantization, the accelerator achieves a significant frame rate and energy efficiency improvement.

Following the work proposed in [8], Nguyen et al. [10] presented the implementation of the HOOFR extractor on an Arria 10 SoC-FPGA using the OpenCL paradigm. They design a feature extraction system incorporating a bucketing method to ensure the homogeneous distribution of keypoints. The HOOFR extractor has been divided into four functional blocks (4 kernels), including FAST detection, Hessian score computing, filtering, and description, as shown in the Figure 1.13. All four kernels are launched concurrently. The bucketing detection is performed by dividing the image into a grid, and a specific number of keypoints is extracted for each image cell. The image cells are thus processed in a pipeline. When a kernel finishes processing an image cell, the following kernel starts to process it immediately. The communication between the kernels is ensured using channels. During experiments, the authors found that the FAST and Hessian score computing kernels were bottlenecks of the algorithm flow, so they decided to physically duplicate those kernels as they do not consume many logic resources. When evaluating the architecture on a publicly available dataset, the speedup factor was up to 9x compared to the implementation on the embedded ARM processor while achieving the same detection result on hardware as on software.

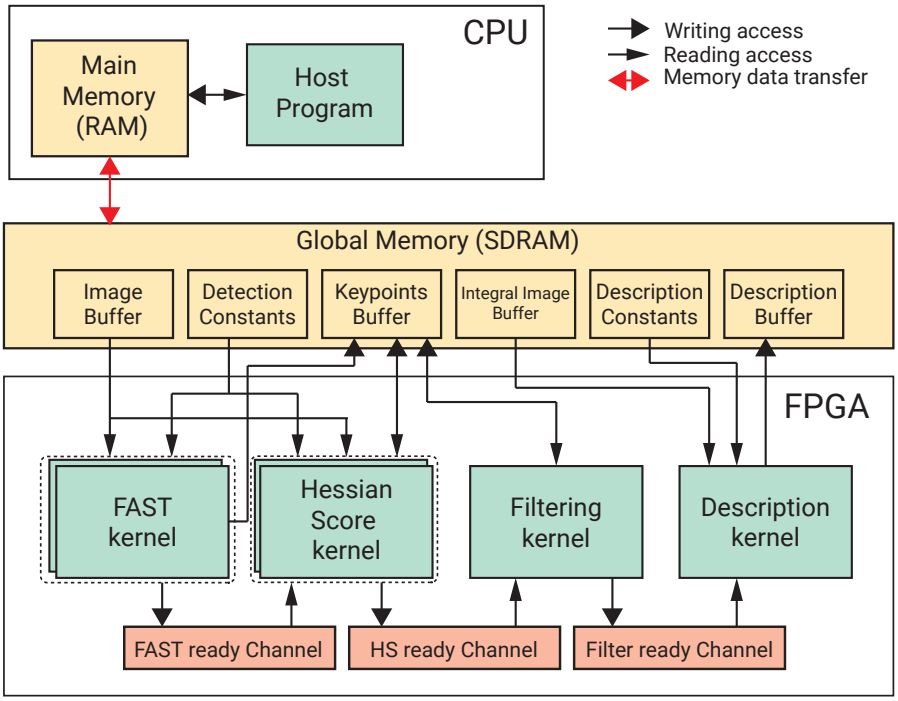

Figure 1.13: HOOFR extractor mapping on CPU-FPGA architecture [10].

Sugiura et al. [104] proposed a universal FPGA-based accelerator, on a low-cost FPGA SoC Pynq-Z2 board, compatible with various 2D LiDAR SLAM methods, including scan matching-based, particle filter-based, and graph-based SLAM. Their implementation focused on the scan matching step, which is the main bottleneck. They optimize the Correlative Scan Matching (CSM) proposed by Olson et al. [105] to exploit the inherent parallelism and reduce resource utilization. The FPGA design includes two CSM cores, which can be used for scan matching and loop detection simultaneously. The architecture was evaluated with different SLAM methods, leading to a significant speed up with low energy consumption.

In Chapter 5 and following the work done in [10], we propose a full implementation of the HOOFR extractor and the matching block on FPGA while maintaining the quality of the results on the CPU and assuring real-time processing at the rate of most modern cameras (30 FPS), to ensure an on-the-fly processing.

1.4 Visual SLAM Datasets

As research in SLAM systems progresses, the need for diverse datasets representing the real world arises. Several datasets exist with various sensor types and data. As we are developing an RGB-D system, our scope in this section is limited to RGB-D datasets. The TUM RGB-D [84] is the most popular dataset in the state-of-the-art. The dataset consists of several sequences in indoor environments, recorded with a Microsoft Kinetic on two platforms: robot and handheld. The dataset contains color images, depth maps, and associated ground-truth camera information acquired using a motion capture system. Also, the authors propose two evaluation metrics that can be used to assess the performance of visual odometry and visual SLAM system: Relative pose error and absolute trajectory error. Another widely used reference dataset is the ICL-NUIM [92]. This dataset focuses on RGB-D algorithms and provides data for evaluating 3D reconstruction across eight synthetically generated indoor scenes. The ground truth includes a 3D surface model and the estimated trajectory by a SLAM algorithm [106]. The Bonn RGB-D Dynamic Dataset [107] is a dataset containing 24 dynamic indoor sequences. The authors provide the ground truth pose of the sensor recorded with a motion capture system. The sequences are in the same format as the TUM RGB-D Dataset, so the same evaluation tools can be used. Unfortunately, to date, there are no outdoor RGB-D datasets due to the limited capabilities of RGB-D cameras. However, the evolution of technology has led to the development of more powerful RGB-D cameras like the Intel Realsense D455/435i/435. In Chapter 2, we use these sensors to record our first indoor and outdoor dataset. Table 1.2 summarize the RGB-D datasets.

Dataset	Year	Env.	Platform	Ground-truth	Availability
TUM RGB-D	2012	Indoor	Robot/Handheld	Motion capture	[84]
ICL-NUIM	2014	Indoor	Handheld	3D surface model SLAM estimation	[106]
Bonn RGB-D	2019	Indoor	Handheld	Motion capture	[107]

Table 1.2: Summary of the most known RGB-D dataset

1.5 Conclusion

In this chapter, we presented the different SLAM systems' sensors. We have noticed that most of the works focus on evaluating the algorithm on the publicly available datasets without considering the characteristics of the sensor. We also introduced the various well-known visual SLAM algorithms. We have seen that most algorithms are tested in indoor environments and evaluated on laptops. We explored how many works exploit the advantage of heterogeneous architectures to accelerate processing and that CPU-FPGA architectures are the trend in robotics and automotive due to their energy efficiency required to ensure long autonomy. Finally, we cited the various well-known datasets used for evaluating RGB-D SLAM algorithms and how they were limited to indoor environments due to the restricted capabilities of such cameras. The following chapter 2 presents the methodology for the design of a dedicated RGB-D SLAM system for autonomous vehicle applications that respects localization accuracy and real-time constraints. Chapter 2 includes the characterization of a new RGB-D sensor for Indoor/Outdoor use, the acquisition of indoor and outdoor datasets, and the choice of platforms used for the evaluations.

Chapter 2

System Design and Evaluation Methodology

2.1 Introduction

This thesis aims to develop a SLAM system for automotive applications. Therefore, the system must be equipped with sensors capable of providing much information about the environment while also handling highly dynamic and large-scale environments. Accordingly, an RGB-D camera is a good choice because of its low cost and properties well suited for SLAM, including RGB images, their associated depth maps, and the pattern projector. Earlier generations of RGB-D cameras had range limitations, restricting their use in indoor environments. As RGB-D cameras have been developed to meet higher range requirements, the use cases have also been extended to outdoor environments.

As we have seen, SLAM systems can be sensitive to the sensors' parameters, so our strategy involves characterizing the RGB-D sensor's influence on localization accuracy, then integrating the RGB-D sensor in the HOOFR-SLAM algorithm and improving its accuracy in outdoor conditions. In order to ensure real-time processing on the fly on an embedded architecture, algorithm-architecture mapping is performed. In this chapter, we will present our choice of sensor and algorithms used for the evaluation, then the indoor and outdoor datasets acquired for the assessment. Finally, we introduce the paradigm used

for the implementation and the different architectures used for the performance evaluation of the RGB-D HOOFR-SLAM algorithm.

2.2 Sensor Choice

RGB-D cameras are sensors providing RGB images and depth maps which are images where each pixel has a value representing the distance to the camera. This information is a significant asset for measuring the exact dimensions of a physical object, which remains challenging even for machine learning algorithms.

Different technologies are used to get the depth map, including passive stereo, structured light, Time-of-flight (ToF), and active stereo. Passive stereo uses two cameras to acquire two images from different viewpoints. Given the calibration of the camera parameters, depth is computed by matching pixels between the images from each camera and triangulating the pixel depth using the baseline. The major limitation of stereo cameras is that the scene should not be poorly textured to find the correspondence, which can only succeed if both cameras see the same features. A wide operating range characterizes this technology, making most of them suitable for acquisition at distances up to $15m$. However, it is not suitable for close-range use, as a wider baseline and focal length allow for better accuracy at long range, but at the same time increase the minimum distance at which the depth can be determined [108]. This fact makes this type of sensor highly recommended for outdoor use.

Active systems incorporate an infrared (IR) projector with a single or stereo camera. They use IR pattern projection to analyze the distortion of these patterns and extract depth, such as structured light, or by directly measuring the depth by employing the time it takes for IR light to be captured after being reflected from objects, such as the time of flight (ToF). The main advantage of this type of approach is that it works very well in low-light environments. However, this type of system performs poorly in outdoor environments where objects are out of reach of the projector, or the projection is overloaded by ambient light, thus explaining their poor performance in outdoor environments.

Figure 2.1 shows the cameras used for the evaluation. The first part involves evaluating an RGB-D sensor in an indoor environment to identify the key factors impacting the quality of the trajectory. For the indoor environment, we chose the Intel Realsense D435i camera. This camera will allow us to compare the impact of the different acquisition modalities thanks to its multiple sensors. The depth camera incorporates a Vision Processing Unit (VPU), left and right imagers for stereo vision with a wide IR projector, an RGB color sensor, and an Inertial Measurement Unit (IMU). Depth features, high resolution, long-range capability (up to approximately 10 m), and global shutter technology enable fast motion capture without blurring depth images. The depth map can be generated using either active stereo technology by turning the projector on or passive stereo technology by turning it off. The IR projector helps increase the texture in low-textured scenes by projecting a static IR pattern. The vision processor generates the depth map by matching each pixel in the right and left IR images, using the image on the left as a reference for stereo matching. The sensor has an RGB camera with rolling shutter technology, a high resolution, and a narrower field of view.

For the autonomous vehicle application, we chose the Intel Realsense D455 camera for the outdoor environment, a version more suitable for outdoor environments. This camera features a broader 95mm baseline, improving depth error to less than 2% at 4m. The camera has an acquisition rate of 30 FPS, ensuring a good overlap for most vehicle movement applications. Also, the camera has a sufficiently wide field of view, which is well suited for large environments so that the SLAM is accurate and robust without worrying about the distortion problems that must be managed in the case of fisheye and omnidirectional cameras.

Figure 2.1: Intel® RealSense™ Depth Cameras

2.3 Algorithm Choice

Visual SLAM Direct methods estimate camera movement by minimizing photometric errors between consecutive images. Their strength lies in high accuracy, thanks to dense image exploitation. The drawback is the sensitivity of these methods to brightness variations or illumination changes. In contrast, feature-based methods use an indirect representation of images, usually in the form of feature points, tracked and then used to estimate the pose by minimizing projection errors. Although indirect methods provide relevant results in well-textured environments, they suffer from failure in poorly textured scenes or motion blur, temporarily wiping out feature points. Direct methods are likely to be computationally expensive, while indirect methods are less computationally expensive. This study's framework is part of the design of an embedded system using RGB-D SLAM for autonomous vehicle applications. For this reason, the choice of an efficient algorithm regarding accuracy and complexity is a critical asset.

For the first part of the algorithmic study, we have selected ORB-SLAM2, a baseline algorithm providing satisfactory results and can be embedded [109], for its low complexity compared to other algorithms. ORB-SLAM2 is for monocular, stereo, and RGB-D cameras, allowing us to compare stereo and RGB-D (front-end part) without worrying about the back-end. Therefore, the correlation between the sensor and the front-end can be studied. To consolidate our research, we chose RTAB-Map [110], a library implementing SLAM with different methods and supporting various sensors (including stereo and RGB-D). RTAB-Map offers real-time processing thanks to its appearance-based loop closure approach with memory management making it suitable for large-scale and for long-term operation. The RTAB-Map is used as a baseline method for comparing the acquisition modes' effect without optimizing its parameters.

For the second part of the algorithmic study, our system is devoted to autonomous vehicle applications, and the algorithm selection must satisfy the requirements of localization accuracy, real-time processing, and limited embedded architecture resources. We choose the HOOFR-SLAM for those purposes:

- The HOOFR-SLAM incorporates the HOOFR bio-inspired extractor, which provides a better balance between speed and matching quality than other state-of-the-art methods.
- The processing complexity of HOOFR-SLAM is decreased to accommodate embedded systems while preserving a significant localization accuracy.
- The ability to integrate the front-end on an embedded CPU-FPGA SoC architecture.

2.4 Dataset Acquisition

2.4.1 Indoor dataset

Several datasets exist in state-of-the-art dedicated to the evaluation of SLAM algorithms. However, these datasets do not consider assessing the impact of the sensor-algorithm coupling on the trajectory quality since each dataset is recorded with only one modality. In our research aiming at studying the sensor algorithm coupling and its impact on localization accuracy, we need a dataset representing the same environment recorded with different modalities (Stereo, RGB-D, IR-D, with and without pattern projector). Since state-of-the-art does not provide such a dataset, we have collected our dataset in the laboratory's corridors and the basement parking, as shown in Figure 2.2. We recorded using an RGB-D camera with a laptop equipped with an Intel Celeron N4100 Quad-Core CPU, 8G RAM and 512GB SSD memory. The experiment was carried out using different acquisition modes, including IR-Stereo, RGB-D Active Stereo (Active: IR Projector on), RGB-D Passive Stereo (Passive: IR Projector off) and IR-D Passive Stereo (Passive: IR Projector off). The IR-D could not be recorded in active stereo since IR patterns interfere with the features extraction generating spurious detections. The images are recorded with a resolution of 1280×720 pixels and a frame rate of 30 frames per second.

(a)

(b)

Figure 2.2: : Images illustrating the environment (laboratory corridor and basement parking) where the dataset was collected: (a) The scene on the right represents a narrow and textureless environment, on the left, we have a narrow scene with more texture. (b) These scenes represent textured and larger environments.

The recorded sequences are summarized in the Table 2.1.

Dataset	Description	Projector state	Acquisition mode
Digiteo_seq1 [111]	Lab corridors 1	Passive stereo	IR-D
			RGB-D
			Stereo
Digiteo_seq2 [112]	Lab corridors 2	Passive stereo	RGB-D
		Active stereo	RGB-D
Digiteo_seq3	Basement parking	Passive stereo	IR-D [113]
			RGB-D [114]
			Stereo [115]
		Active stereo	RGB-D [116]

Table 2.1: Recorded sequences using various acquisition modes

A reference trajectory for comparison was created, as shown in Figure 2.3, based on temporal subsampled (subsampled by 1/5 for seq1 and seq2 and by 1/10 for seq3) monocular images of the environment using the Structure from Motion (SfM) and Multi-View Stereo (MVS) pipeline COLMAP [117, 118]. COLMAP's sub-pixel reprojection error [117]

provides a well-dense reconstruction of large-scale environments, making it a reliable reference for comparison. The provided images are synchronized IR/RGB images with the depth images. The images used for the reference reconstruction are non-synchronized images that have been sub-sampled while keeping the timestamp of each image. Subsampled images allow for faster processing while providing sufficient visual overlap. The first step is to detect and extract features from all images and describe them using a numerical descriptor. The feature extraction uses a pinhole camera model [63] with the camera's intrinsics and extrinsics parameters, as shown in the Tables 2.2 and 2.3. The extractor used is SIFT and executed on GPU with a maximum number of primitives of 8192.

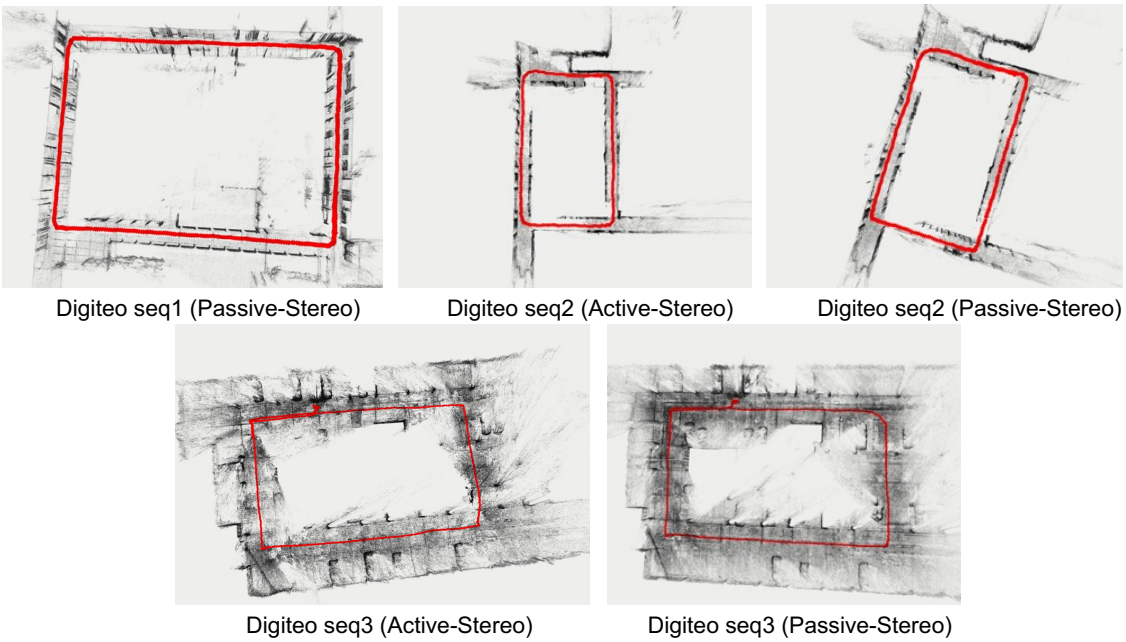

Figure 2.3: Reference trajectory in red using COLMAP. COLMAP is a pipeline of SfM and MVS providing a dense reconstruction of large-scale environments with sub-pixel reprojection error make it a reliable reference for comparison.

	IR (Left & Right)	RGB
f_x (pixel)	638.14	912.36
f_y (pixel)	638.14	910.26
c_x (pixel)	639.75	648.57
c_y (pixel)	356.51	363.66

Table 2.2: Intrinsic parameters of RGB and IR cameras of the D435i camera used in our dataset, including focal length (f_x,f_y) and optical center (c_x,c_y).

	"Color" to "Depth"	"IR1" to "Depth"	"IR2" to "Depth"
Rot	$\begin{bmatrix} 0.99 & -0.0089 & -0.044 \\ 0.0089 & 0.99 & 0.0009 \\ 0.0044 & -0.0009 & 0.99 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
Transl (m)	$\begin{pmatrix} -0.0148 \\ -0.0001 \\ -0.0002 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0.0502 \\ 0 \\ 0 \end{pmatrix}$

Table 2.3: Extrinsics of the D435i camera

The intrinsic parameters are fed to the algorithm and shared between all images. Then, the geometric matching and verification are performed using sequential matching, which is best suited for consecutive frames with sufficient visual overlap. The overlap is set to 20 images, with quadratic overlap and loop detection enabled. The values of the remaining parameters are kept as default. Tables 2.4 and 2.5 summarize all parameters values.

Camera model	Pinhole
Shared for all images	Yes
Custom parameters	RGB: 912.36 px, 910.26 px, 648.57 px, 363.66 px
	IR: 638.14 px, 638.14 px, 639.75 px
Max_image_size	3200
Max_num_features	8192
First_octave	-
Num_octaves	4
Octave_resolution	3
Peak_threshold	0.00667
Edge_threshold	10
Estimate_affine_shape	No
Max_num_orientations	2
Upright	No
Domain_size_pooling	No
Dsp_min_scale	0.16667
Dsp_max_scale	3
Dsp_num_scales	10
Num_thread	-1
Use_gpu	Yes
GPU_index	-1

Table 2.4: Feature extraction COLMAP parameters

Loop closure detection is used through a pre-trained vocabulary tree. The GPU accelerates the matching process. Once the matching step is finished, the sparse reconstruction is launched. Data is loaded from the database into memory during this process, and the

Overlap	20
Quadratic_overlap	Yes
Loop_detection	Yes
Loop_detection_period	10
Loop_detection_num_images	50
Loop_detection_num_nearest_neighbors	1
Loop_detection_num_checks	256
Loop_detection_num_images_after_verification	0
Loop_detection_max_num_features	-1
Vocab_tree_path	32K words (small-scale)
	256K words (medium-scale)
Num_threads	-1
Use_gpu	Yes
GPU_index	-1
Max_ratio	0.8
Max_distance	0.7
Cross_check	Yes
Max_num_matches	32768
Max_error	4
Confidence	0.99
Max_num_trials	10000
Min_inlier_ratio	0.25
Min_num_inliers	15
Multiple_models	No
Guided_matching	No

Table 2.5: Feature matching COLMAP parameters

scene is expanded by incrementally registering the images from an initial image pair seed. Finally, a model can be exported, containing the camera information, the images including all the keypoints and the reconstructed pose of an image specified as the projection of the world to the camera coordinate system of an image using a quaternion and a translation vector, and finally the 3D points in the dataset. After the model is acquired, the reconstructed poses of the images are used to calculate the coordinates of the center of the projection/camera using Eq. 2.1.

$$c_c = -R^T t \tag{2.1}$$

where c_c is the coordinates vector of the camera center, R^T is the transpose of the rotation matrix obtained from the quaternions, and $\mathbf{t}$ is the translation vector. For the scaling of the

trajectory, we proceed to a dense reconstruction of the environment. This step consists of importing the sparse 3D model and launching the MVS, which first involves undistorting the images. The normal and depth maps are computed to be fused into a dense point cloud. Finally, the dense surface is estimated using Poisson or Delaunay reconstruction. This dense point cloud will allow us to recover the distances of some objects with known dimensions. Scale is computed as the ratio between distances on the point cloud and their corresponding measured with a rangefinder. This scale factor allowed us to scale our reference trajectory. Figure 2.4 summarizes the process of trajectory reconstruction using COLMAP.

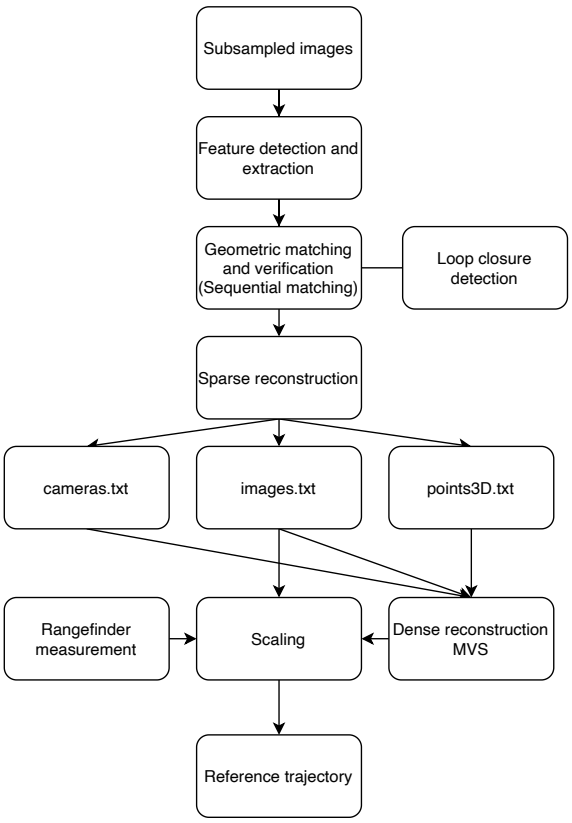

Figure 2.4: Reference trajectory process using COLMAP. Reconstructing the reference trajectory consists of sub-sampling the images (to reduce the processing time), detecting features, and sequentially matching those features. Then the sparse reconstruction is launched to generate a point cloud and to compute the camera poses. Finally, the dense reconstruction is performed by the Poisson or Delaunay method. The camera trajectory is scaled by calculating the scale factor between the point cloud and known distances provided by a rangefinder.

2.4.2 Outdoor dataset

As discussed in the previous chapter, most RGB-D datasets are created in indoor environments. To our knowledge, there is currently no RGB-D dataset for vehicle applications like the one from KITTI [89] due to the limitations of RGB-D cameras. Recent generations of RGB-D cameras can operate in both indoor and outdoor environments. As part of our thesis, we need to evaluate our algorithm on an RGB-D dataset recorded by a vehicle. We conducted measurement experiments with the instrumented laboratory vehicle Figure 2.5 to record the outdoor dataset and evaluate SLAM performance in outdoor use conditions. The vehicle is equipped with a Realsense Intel D455 camera, a LiDAR, and an RTK-GNSS receiver to record the ground truth. The D455 camera is characterized by its long range of up to 20m provided by the projector, while the depth images provide a more extensive range, making it suitable for outdoor applications. D455 is equipped with two IR cameras, one RGB camera, and an IR projector for active stereo mode. The intrinsic camera parameters resulting from the calibration are presented in the Table 2.6.

The GNSS receiver is an Altus Positioning Systems (APS-3) using Real-Time Kinematic (RTK) correction signals to provide centimeter-level accuracy in positioning. This ground-truth output was collected with a frequency of 25Hz. The environment in which this dataset was recorded was cloudy and wooded. The positioning solution was post-processed using RTKLIB and only the highly reliable positions with a quality flag equal to 1 or 2 in our dataset were retained.

	IR (Left & Right)	RGB
f_x (pixel)	644.49	632.02
f_y (pixel)	644.49	631.28
c_x (pixel)	642.78	641.33
c_y (pixel)	347.77	372.88

Table 2.6: Intrinsic parameters of RGB and IR cameras of the D455 camera used in our dataset, including focal length (f_x,f_y) and optical center (c_x,c_y).

Figure 2.5: SATIE laboratory-instrumented vehicle embedding an Intel Realsense D455 RGB-D camera, a GNSS receiver (Altus Positioning Systems (APS-3) Real-Time Kinematic (RTK)), a Velodyne LiDAR PUCK with 16 channels (VLP-16) and a data logger.

In sequence 1, we recorded two IR images and the depth map with a frame rate of 30 FPS and a resolution of 1280×720 pixels. The car speed is up to 40km/h. This scene is full of dynamic cars and buses. Also, in several frames, most of the image is dominated by the sky, as shown in Figure 2.6. The sequence consists of a loop in a dynamic environment over 978.57m. In sequence 1, the left IR camera allows us to have images aligned with the depth map since it is used as the reference for the stereo matching, which exempts us from performing the alignment in post-processing. Sequence 2 represents a trajectory along 189.81m in a straight path.

Figure 2.6: The outdoor dataset depicts dynamic scenes, sky-dominated scenes, and shaded scenes.

2.5 Evaluation Metrics

Evaluation metrics introduced by Sturm et al. [84] are used: The absolute trajectory error (ATE) provides a measure of the translational error, from a comparison of the absolute distances between the referenced trajectory and the estimated trajectory. The referenced trajectory and the estimated one are aligned and time-synchronized. Given a sequence of poses from the estimated trajectory $P_1, ..., P_n \in SE(3)$ and from the referenced trajectory $G_1, ..., G_n \in SE(3)$, we can find the rigid-body transformation S using the Least-Squares Rigid Motion method, then the trajectory are aligned, and the ATE at time step i is computed as

$$ATE_i = G_i^{-1} S P_i \tag{2.2}$$

The root mean squared error over all time indices of the translational components is computed as

$$RMSE(ATE_{1:n}) = (\frac{1}{n}\sum_{i=1}^{n} \|trans(ATE_i)\|^2)^{1/2} \tag{2.3}$$

The Relative Pose Error (RPE) is used to find the rotational error. These metrics allow an assessment of the estimated trajectory quality compared to the referenced trajectory. RPE metric is measured over a fixed time interval of $\Delta = 1s$ (30 frames) which gives us the drift per second on a sequence recorded at 30 Hz. RPE at time step i is defined as

$$RPE_i = (G_i^{-1} G_{i+\Delta})^{-1} (P_i^{-1} P_{i+\Delta}) \tag{2.4}$$

The root mean squared error over all time indices of the translational components is computed as

$$\begin{gathered} RMSE(RPE_{1:n}, \Delta) = (\tfrac{1}{m}\textstyle\sum_{i=1}^{m} \|trans(RPE_i)\|^2)^{1/2} \\ m = n - \Delta \end{gathered} \tag{2.5}$$

2.6 Hardware architectures

By leveraging the latest hardware and software implementation technologies, the embedded system design addressed the need to improve overall system performance and reliability while minimizing design and production costs. As the performance is articulated on three bases, timing, power consumption, and reliability, researchers have turned their attention to multicore and reconfigurable designs. The transition to multicore architectures results from the challenge of enhancing serial performance. For a single core to execute instructions faster, more silicon area is needed, increasing power consumption and thermal output. Also, due to memory access latencies, increasing the frequency of single-core processors is no longer adequate, prompting the development of multicore and reconfigurable designs [119].

Reconfigurable and multicore architectures are capable of processing a large number of tasks in parallel to speed up the processing flow. In the quest for higher speed, computers have come a long way, from the first central processing units (CPUs) to modern parallel designs like graphics processing units (GPUs) and field programmable gate arrays (FPGAs). We distinguish two types of computing systems: homogeneous and heterogeneous systems. A homogeneous system includes only one type of computational unit. On the other hand, a heterogeneous system combines several computational units of different types (Multicore CPUs, GPUs, DSPs, FPGAs...). Each type of calculator has its pros and cons. Using a given computational unit may be more suitable for a given task than for another. The Algorithm-Architecture Adequacy (AAA) [68, 69, 8, 120] approach simultaneously studies the algorithmic and architectural aspects by considering their interactions. Algorithm-architecture adequacy allows for the distribution of the tasks of an algorithm on the different available processing units to achieve optimal performance.

In recent years, significant progress has been made in the design of embedded architectures. Nvidia Jetson AGX, for example, offers a high-performance CPU similar to that of the desktop and a GPU with a maximum power consumption of 30W. This architecture is widely used for AI and computer vision applications [121, 122, 123].

Regarding online processing, FPGAs are the best choice [124, 125, 126, 127]. FPGAs have I/O blocks, which allow external devices to be connected to the FPGA using memory

and interfaces. This architecture model reduces the communication path between the FPGA accelerator and the peripherals. Also, FPGAs inherently offer low latency suitable for real-time applications, such as video streaming, by ingesting the video directly into the FPGA's interfaced memories, bypassing the CPU, which remains mandatory for dataflow control. The properties of FPGAs have encouraged several works to develop FPGA-based sensors [127, 128]. In front-end processing, the processing should be the closest possible to the sensor's frequency to ensure on-the-fly processing. Different target architectures are used for the algorithm evaluation:

- **Laptop:** The algorithm is first evaluated on a high-performance PC equipped with an 8-core AMD Ryzen 9 CPU with a base frequency of 3 GHz, an L3 cache memory of 8 MB, an L2 of 4MB, and an L1 of 64KB. The RAM is 24GB DDR4. This platform also integrates an NVIDIA GeForce RTX 2060 Max-Q graphics card with 1920 Shading units, 6 GB of global memory with a bandwidth of 264.0 GB/s. The CPU consumption is 35W, while that of the GPU is 64W. These features will enable real-time processing for a well-optimized SLAM system. However, power consumption remains a hurdle for battery-based applications.

- **Nvidia Jetson AGX:** For autonomous vehicles, the NVIDIA Jetson AGX Xavier development kit is a platform that offers the performance of a GPU workstation in a less than 30W embedded module. The platform includes a 512-core NVIDIA Volta GPU with 64 Tensor Cores, as shown in Figure 2.7, with a maximum clock frequency of 1.37GHz. The GPU contains eight Volta Streaming Multiprocessors (SMs) with 64 CUDA cores and 8 Tensor Cores per Volta SM and a 128KB L1 cache. The SMs share a 512KB L2. The CPU is an 8-core NVIDIA Carmel ARMv8.2 64-bit, as shown in Figure 2.8, with a maximum clock frequency of 2.26GHz. Each core includes 128KB instruction and 64KB data L1 caches plus a 2MB L2 cache shared between the two cores. The CPU clusters share a 4MB L3 cache. The RAM is 16GB 256-bit LPDDR4x, which has 137GB/s memory bandwidth. The platform is used in several SLAM works, including [129, 123, 130].

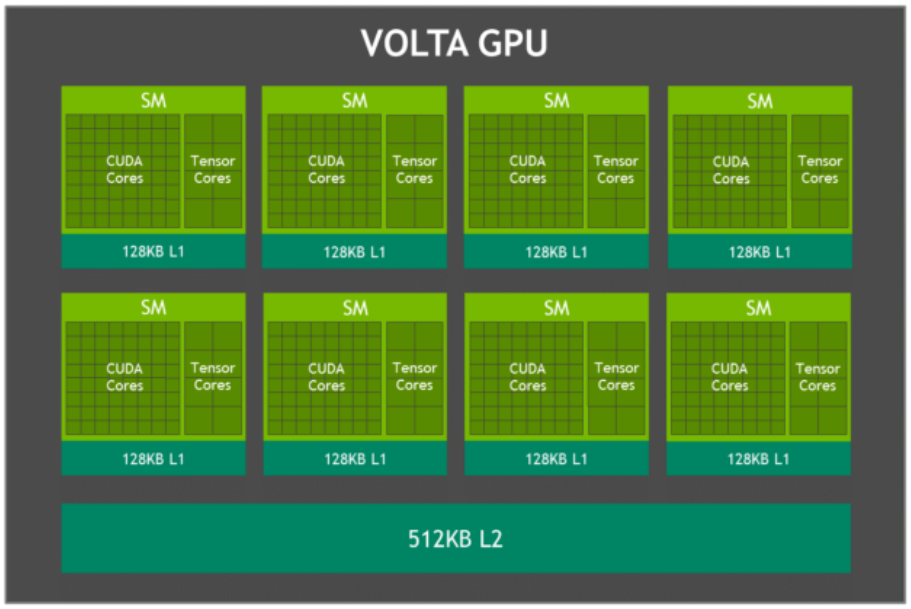

Figure 2.7: Block diagram of Jetson Xavier VOLTA GPU [11]

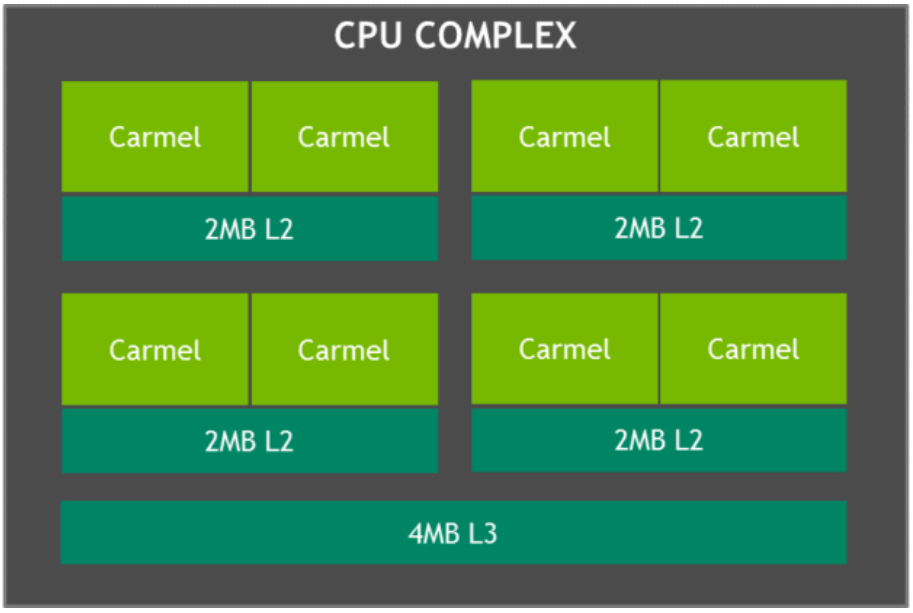

Figure 2.8: Block diagram of Jetson Xavier CPU [11]

- **Altera DE5a-Net DDR4:** DE5a-Net DDR4 Arria 10 FPGA Development Kit represents a hardware solution for designs requiring more resources. The Arria® 10 GX FPGA features integrated transceivers that transfer at up to 12.5 Gbps, allowing the DE5a-Net DDR4 to be fully PCI Express version 3.0 compliant. For designs

requiring high capacity and speed for memory and storage, the DE5a-Net DDR4 offers two independent banks of 16GB DDR4 memory module, running at over 75 Gbps, up to 7.876 GB/s data transfer via PCIe Gen 3x8 edge between FPGA and host PC. Arria 10 GX FPGA features 1150K logic elements, 67-Mbits embedded memory, and 1518 Variable-precision DSP block. As our autonomous vehicle application needs to perform on-the-fly processing, we opt for a front-end processing architecture on FPGA, thus requiring a large amount of resources, making the Arria 10 a good choice for implementing such an architecture. Figure 2.9 shows the heterogeneous architecture used for the evaluation. The DE5a-Net DDR4 board is interfaced with a host CPU via PCIe Gen 3x8. The CPU is an Intel Xeon Silver 4108 processor with eight cores with a base frequency of 1.8GHz.

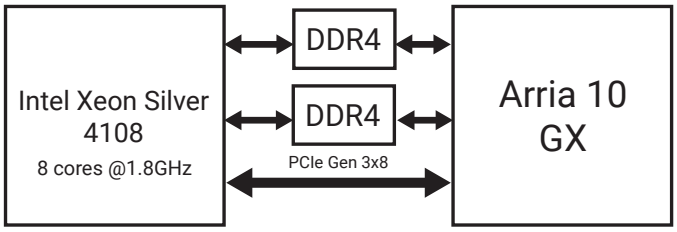

Figure 2.9: Heterogenous CPU-FPGA used architecture

2.7 FPGA HW/SW Codesign Approach

Reconfigurable architectures offer great flexibility in the design of systems based on logic circuits. In this category, we find mainly PALs (Programmable Array Logic), CPLDs (Complex Programmable Logic Device), and FPGAs (Field Programmable Gate Array). FPGA is a two-dimensional array of programmable hardware that can be reconfigured. It contains programmable logic blocks, interconnects, configurable memory modules, and DSPs [131], as shown in Figure 2.10.

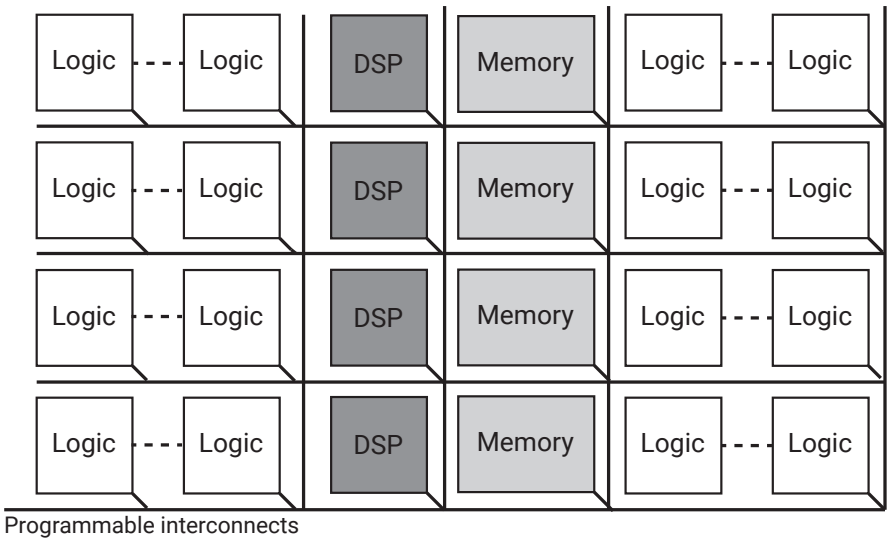

Figure 2.10: Simplified internal structure of FPGA

The FPGA offers great flexibility in the design of embedded systems, with low power consumption compared to programmable processors. Its spectrum of use is vast. It can be found in almost every field [132, 133, 101, 134]. While FPGAs offer advantages in parallelism and power consumption, the description of FPGA-based architectures is relatively complex compared to programmable processors. Since FPGAs emerged in the 80s, design approaches and tools have evolved considerably to facilitate hardware description [135]. FPGAs are typically programmed using hardware description languages (HDL), such as Verilog or VHDL. To design a high-performance accelerator, the programmer needs expertise in hardware design. Moreover, HDL-based designs require cycle-level simulations and debugging, making the design process time-consuming. As the complexity of algorithms has grown, HDL has become inefficient since it becomes challenging for designers to develop circuit details and control states for large and complex FPGAs. However, this limitation has been addressed by a technique called high-level synthesis (HLS).This latter facilitates the complete design of the entire processing flow on a heterogeneous system: Computation on the host, data transfer between the host and the accelerator, and computation on the accelerator [136].

There are two types of heterogeneous CPU-FPGA architectures: The SoC (System-on-Chip), where the CPU and FPGA are integrated on the same chip, this system is often used for low-power embedded applications, and the other system is composed of a

CPU and FPGA connected by an external bus such as PCI-express (PCIe) used for high-performance computing. In this thesis we use an FPGA connected to the host via a PCIe bus, as a proof of concept of the performance that can be achieved by embedding front-end processing.

2.7.1 OpenCL approach

In high-level synthesis based on the C language, the designer must adapt the code to the tool used. The high-level abstraction requires tool and target architecture expertise to maximize performance. The idea of the OpenCL approach is to allow the user to design a heterogeneous architecture based on the OpenCL standard. This type of architecture has two parts: the host and the FPGA accelerator. The latter then plays the role of a coprocessor, similarly to a GPU.

Open Computing Language (OpenCL) is a C-based design environment for a heterogeneous computing platform that includes a host CPU and accelerators such as GPUs and FPGAs. Designers can use OpenCL to create end-to-end computations, including computation on the host, data transfer between the host and the device, and computation on the device. As a result, by evaluating OpenCL codes, the OpenCL design environment for FPGAs may generate FPGA circuits and interface circuits. The OpenCL paradigm has several advantages:

- Significantly reduce design time.
- Compatible and reusable on different FPGA boards by recompiling the code using the boards' BSP (Board Support Package).
- Debugging through functional code verification using the CPU emulator.
- Profiling allows collecting information about memory accesses during execution.

2.7.2 OpenCL Design Flow

The OpenCL-based design flow is divided into three stages: emulation, performance optimization, and execution. We test the code's behavior during the emulation phase by

running it on a CPU. We use compilation reports and profile information to identify performance bottlenecks during the performance optimization process. Then, by removing bottlenecks, we boost performance. In the execution phase, we execute the OpenCL program on an FPGA-based computing system to evaluate its actual performance.

2.7.2.1 Emulation phase

In the first stage, we emulate the OpenCL kernel code on a CPU to test its behavior. Compiling for emulation may be done quickly, usually taking between a few seconds to a few minutes. Although an FPGA board is not required for emulation, a BSP is necessary. The code is performed sequentially in emulation, just like in a normal C-like code. Emulation ignores parallel operations such as pipelines, loop-unrolling, and SIMD operations.

2.7.2.2 Performance tuning phase

The performance tuning phase analyzes compilation reports to identify performance bottlenecks. The compilation report includes:

- A loop analysis includes pipeline information, bottlenecks, initiation interval (II), unrolling loop information, etc.
- The estimated resource utilization information.
- System viewer shows a kernel as a combination of blocks and how multiple kernels are connected.

2.7.3 FPGA-Oriented Parallel Programming

In an OpenCL device, kernels are functions executed on an OpenCL device. One work-item represents a unit of the execution of a kernel. A group of these work-items is called a work-group, and the entire collection of work items is called an NDRange. In a heterogeneous computing system, we have different types of memories:

- The host memory: accessible only by the host.
- The global memory: accessible to both the host and the device.
- The constant memory: a read-only memory for the device.
- The local memory belongs to a particular work-group, and data are shared only by its work items.
- The private memory belongs to a work-item and is not accessible to the other work-items.

In OpenCL FPGA programming, there are two types of kernels: NDRange kernels and single-work-item kernels [137]. The NDRange kernel is executed by multiple work-items in parallel in a pipeline manner, as shown in Figure 2.11. In the first cycle, work-item one is launched and loads data from memory. In the next cycle, work-item two is launched, and while loading data, work-item one performs the addition. In the third cycle, work-item three loads data while work-item two performs addition, and work-item one stores the result [12].

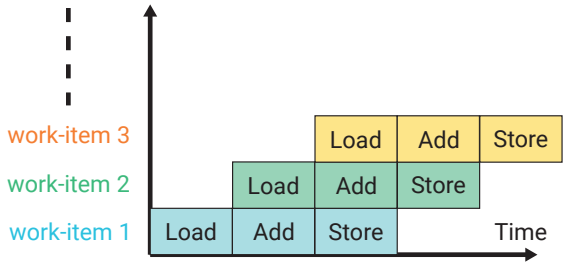

Figure 2.11: Time chart of NDRange kernel execution

In a single-work-item kernel that contains loops, loop-iterations are used as the unit of execution of a kernel, and multiple loop-iterations are computed in different pipeline stages, as shown in Figure 2.12. Similar to the NDRange time chart, at each cycle, a loop-iteration is launched, and at the third cycle, the three loop-iterations are executed in parallel at different pipeline stages [12].

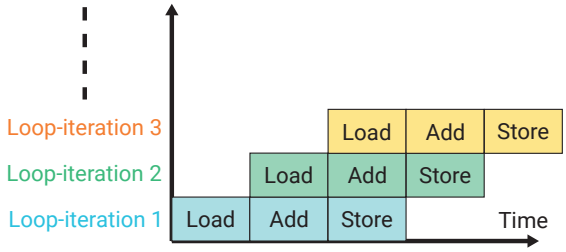

Figure 2.12: Time chart of single-work-item kernel execution [12]

The NDRange kernels are commonly used when no data dependencies exist or if we want to use the same kernel in both FPGAs and GPUs. On the other hand, the single-work-item is easy to implement and can provide better performance than the NDRange kernels if there are data dependencies [12].

2.8 Conclusion

In this chapter, we have presented the methodology followed and the tools used to evaluate our contributions. In this thesis, we use our datasets since existing ones do not fulfill our study's needs. In order to be consistent with the evaluations made on state-of-the-art, we have evaluated several algorithms [76, 110, 8, 117] on our dataset and relied on the same metrics used for evaluating algorithms on publicly available datasets [84].

Our approach covers the whole processing chain from the sensor to the target architecture. For this, we start with a study on the coupling sensor algorithm and its impact on localization accuracy in chapter 3. Indoor datasets are used for this study. Then, based on the HOOFR-SLAM algorithm, we develop an RGB-D version of HOOFR-SLAM in chapter 4, which is more robust and improved in terms of localization accuracy and performance in outdoor conditions. Finally, algorithm-architecture adequacy is applied to boost the performance of the algorithm in chapter 5. Algorithmic and hardware optimizations are used to achieve our goal of real-time processing on an embedded architecture.

Chapter 3

Sensor-Algorithm Parameters Coupling

3.1 Introduction

SLAM is a crucial perception functionality in a variety of applications, including robots and autonomous vehicles. RGB-D cameras are among the sensors typically employed by recent SLAM systems. Numerous RGB-D SLAM algorithms have been explored and assessed using publicly available datasets without taking into account sensor specifications or image capture modes that might increase or reduce localization accuracy. In this chapter, we discuss indoor localization while taking sensor specifications into account. In this context, we highlight the impact of sensor acquisition modalities on localization accuracy and suggest a parametric optimization strategy to improve localization accuracy in a given environment. This protocol is used to improve a depth-related SLAM algorithm parameter. Our own publicly available indoor dataset served as the basis for this analysis.

3.2 Related Works

Several works have been carried out to improve the visual SLAM, given its multiple advantages. Visual SLAM began by exploiting the images from a single camera called a monocular system [1, 3, 138] and evolved with stereo systems to solve the problem of scale drift [8, 139, 140]. Some of the stereo SLAM systems' contributions include R. Mur-Artal et al. [76]. They contributed with the stereo version of the ORB-SLAM that fixes the

problem of scale drift in their monocular version [1]. The stereo version applies the same approach of local bundle adjustment in a set of local keyframes so that the complexity is unaffected by the map's size, making it usable in large-scale environments. The algorithm was evaluated on KITTI [89], EuRoC [90], and TUM [84] datasets. HOOFR-SLAM [8], a recent algorithm with competitive performance, exploits the HOOFR extractor for feature detection and matching [57]. HOOFR-SLAM implements a processing structure that maximizes parallelism and avoids the need to optimize camera poses by applying bundle adjustments on keyframes or saving the history of map points by estimating the relative poses of the current input frame with a set of previous neighboring frames. The optimal pose is obtained by averaging the relative poses with weighted factors. Nguyen et al. evaluated the algorithm on KITTI [89], Oxford RobotCar [94], Malaga [95], MRT [96], St Lucia [97] and New College datasets [88]. PL-SLAM, presented by R. Gomez-Ojeda et al. [139], provides a solution for low-textured environments, combining points and line segments to operate robustly in a wider variety of scenarios, especially in those where point characteristics are rare or poorly distributed in the image. They also introduce a new bag of words that relies on combining the descriptive potential of the two types of features. PL-SLAM was evaluated on KITTI [89] and EuRoC MAV [90] datasets. Unlike previous work, Y. Liu [141] et al. designed a complete SLAM system, including the sensor. Y. Liu et al. [141] proposed a real-time stereo SLAM system based on the bionic eye inspired by the peripheral and central vision of the human eye. With the ability to mimic human eye movements, stereo cameras can improve the SLAM system's robustness in low-textured environments by actively searching for rich textured area. Most of the previous works are evaluated on large publicly available datasets. As a result, the evaluation of these algorithms lacks consideration of the sensor's characteristics (Field of view, shutter type, baseline, etc.) and their impact on the quality of the algorithm's output.

3.2.1 RGB-D SLAM

The emergence of RGB-D sensors allowed the evolution of 3D dense reconstruction. Many algorithms have exploited the RGB-D images to optimize performance and allow real-time execution, in addition to the embedding capability of these algorithms as

on mobiles [142]. A well-known and open-source RGB-D system by Endres et al.[143] includes the front-end part dedicated to compute frame-to-frame motion using feature matching and the Iterative Closest Point (ICP). The back-end part performs optimization of the pose-graph with loop closure constraints based on a heuristic search. R. Mur-Artal et al. [76] have upgraded the ORB-SLAM2 for a loosely coupled use of RGB-D input. ORB-SLAM2 uses depth information to synthesize stereo coordinates for the elements extracted from the image. In this way, the system is adaptable whether the input is stereo or RGB-D. M. Labbé et al. [110] have proposed an extension of the RTAB-Map library to implement SLAM with different sensor configurations and processing capabilities. Q. Fu et al.[144] proposed an RGB-D SLAM system using points and lines as features, which improved trajectory performance in low textured scenes. Despite the improvement brought by the cited works, their algorithms were evaluated on online datasets, which neglects the impact of the sensor-algorithm coupling. For example, the BAD SLAM [83] algorithm has shown that direct RGB-D SLAM algorithms are highly sensitive to cameras with rolling shutters, RGB and depth sensor synchronization, and calibration errors. So they evaluated their algorithm on their dataset acquired with synchronized global shutter RGB and depth cameras. Therefore, the online datasets provide only a partial picture of SLAM performance. Designing algorithms considering the sensor's properties could significantly improve the localization accuracy by exploring various sensors and their settings and identifying the algorithm's parameters directly related to the sensor used and how it impacts the algorithm's accuracy.

3.2.2 RGB-D Sensors Assessment

Few works have explored the characterization of RGB-D sensors. Notable works include M. Carfagni et al. [145], who have characterized the Intel SR300 depth sensor using it as a 3D scanner. This sensor's performance was evaluated by applying the German standard VDI/VDE 2634 on a raw dataset and a dataset with optimized parameters (Filters available on the cross-platform camera capture for Intel® RealSense™). Decoupling sensor raw data analysis from the optimized one allows an understanding of the worst performances of the device when used as a 3D scanner.

The VDI/VDE guideline presents a method of measuring a reference object (sphere, plane) used to define some essential characteristics of the analyzed optical system. Traceability of 3D measurements is warranted by acceptance test and re-verification test. The acceptance test involves the measurement of a calibrated artifact (sphere, a ball bar, and a plane). The acceptance test is accepted if the error lies inside the limits specified by the manufacturer, while the re-verification test is a repetition of the acceptance test over time. Three characteristics are estimated in this evaluation. The first is the Probing Error, measured using a sphere and defined as the characteristic error of the system within a small part of the measurement volume. The second one is Sphere Spacing Error (SS). SS is the difference between the acquired distance and the "true" distance between the centers of the two spheres (estimated from the point cloud of data using a best-fit sphere-fitting), the artifact used is a bar connecting two spheres (ball-bar). This characteristic shows the capability of the system to perform length measurements. Finally, the last characteristic estimated is Flatness Measurement Error F, which is the range of the signed distances of the measured points from the best-fit plane. This characteristic is measured using a rectangular parallelepiped.

Another study, conducted by the same authors, was carried out on the Intel D415 sensor based on the same German standard for 3D scans [146]. E. Lachat et al. [147] performed an evaluation and calibration of the Microsoft Kinect depth camera to reconstruct small 3D objects. To our knowledge, most of the work carried out for the characterization of depth sensors is related to 3D scanning and small 3D object reconstruction. This chapter proposes an approach to characterize the RGB-D sensor for SLAM applications. Our approach involves studying the different acquisition modalities and the impact of each modality's characteristics on the quality of the algorithm's output. A sensor-algorithm coupling is performed by identifying the algorithmic parameters directly correlated with the RGB-D sensors and by proceeding to analyze the system input and output quality.

3.3 Studied RGB-D SLAM approaches

3.3.1 ORB-SLAM2

ORB-SLAM2 is a feature-based method that computes the camera trajectory and a sparse 3D reconstruction [76]. It includes the three perception variants of the approach: monocular, stereo and RGB-D. ORB-SLAM is recognized for its ability to reuse the map, to close the loop and to perform re-localization. It is structured in three processing threads: The tracking thread, localizes the camera every frame, by finding feature matches in the local map and minimizes the re-projection errors by applying motion-only Bundle Adjustment (BA). The local mapping thread manages the local map and optimizes it by performing local BA. Moreover, the loop closing thread detects large loops and corrects the accumulated drifts by performing graph-pose optimization. A general overview of the system is shown in Figure 3.1. In this study, we focus on the pre-processing input module, Figure 3.2, in the Tracking thread.

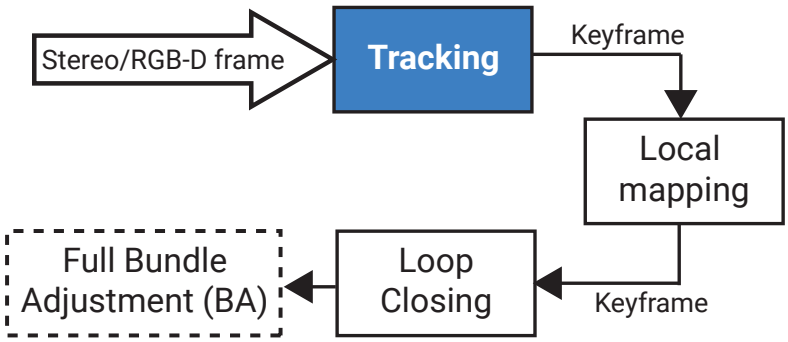

Figure 3.1: Overview of the ORB-SLAM2 thread system. We operate on the pre-processing input module inside the Tracking thread.

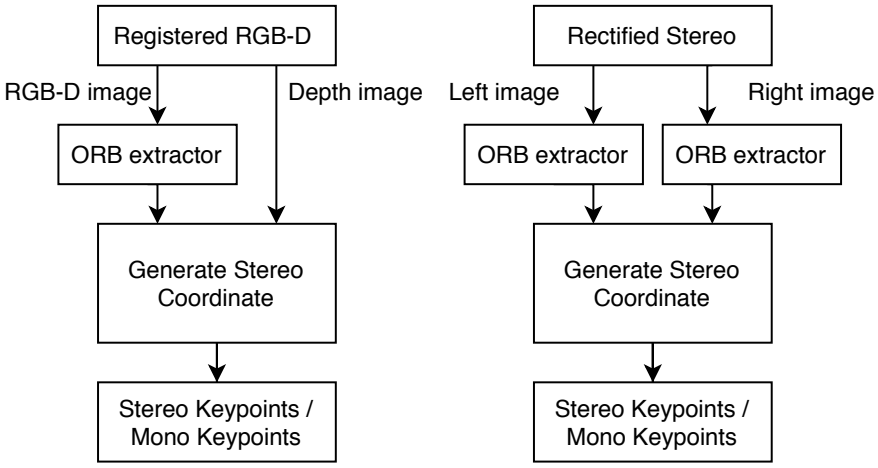

Figure 3.2: In the tracking thread, the stereo and RGB-D inputs are pre-processed to provide the same input data used throughout the system regardless of the input sensor [76]

The stereo ORB-SLAM is based on ORB extractor, which is a binary descriptor based on BRIEF [60]. It generates the stereo keypoints with ORB coordinates on the left and the horizontal coordinate of the right match, which are defined as follows: (u_l, v_l, u_r) where (u_l, v_l) are the coordinates in the left image, and u_r is the horizontal coordinate in the right image.

While for an RGB-D input, the Tracking thread extracts the features from the RGB image and for each feature with the coordinates (u_L, v_L), it transforms the depth value d into a virtual right coordinate u_r, as shown by Equ. 3.1.

$$u_r = u_l - f_x \cdot \frac{b}{d} \tag{3.1}$$

In addition to the camera's intrinsic parameters, the ORB-SLAM2 has three parameters that must be adjusted before it is launched. These parameters are the keypoints number, the FAST threshold, and the close/far threshold.

The number of keypoints defines the maximum number of corners to be detected in the entire image. Figure 3.3 shows how increasing the keypoints number leads to a significant rise in execution time. Furthermore, a higher number of keypoints can result in a larger error.

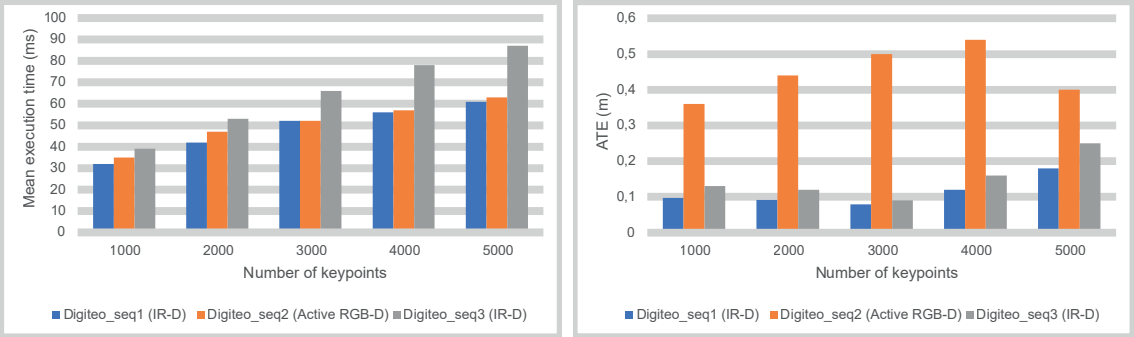

Figure 3.3: Impact of the number of keypoints on the execution time and on the Absolute Trajectory Error (ATE) measured on the three sequences: Digiteo_seq1 (in IR-D mode), Digiteo_seq2 (in RGB-D active-stereo mode), and Digiteo_seq3 (in IR-D mode), with the FAST and DepthThreshold parameters set at their default values.

The FAST threshold is used to test each pixel, whether it is a keypoint or not, based on the intensity of the neighboring pixels. The ORB-SLAM uses an adaptive threshold, so the min threshold is used if no point is detected in the cell with the initial threshold.

Finally, the close/far threshold is a coefficient which is multiplied by the baseline to establish a threshold distance over which classification of far and near keypoints is performed. This parameter is tied to the sensor since it depends first on the sensor baseline and secondly on the depth value of the keypoints. Paz et al. [148] proposed a simulated experiment to study the effect of the linearization in 3D and Inverse depth representation when a point is initialized using the stereo information. They consider a point between the two cameras at different distances and they plotted the uncertainty region of the 3-D representation. They found that for shorter distances (5m) the uncertainty covers the real distributions quite accurately and for longer distances uncertainty in closer region to the camera is overestimated and overconfident for far distances. Thus, based on [148] Mur-Artal et al. [76] classified keypoints as near if their associated depths are 40 times less than the stereo/RGB-D baseline. Otherwise, they are considered far. Near keypoints are devoted to compute scale, translation and rotation since their depth is accurately estimated. On the other hand, far points provide accurate information on rotation, but uncertain information on scale and translation. Far points are triangulated when multiple views support them.

Finally, the monocular keypoints are defined by two coordinates $x_m = (u_L, v_L)$ in the image on the left. These are the points for which a stereo match could not be found or which have an invalid depth value in the RGB-D case. These points are only triangulated from multiple views and used only to contribute to the estimation of rotation and translation [76].

3.3.2 RTAB-Map

RTAB-Map is a graph-based algorithm, fed with RGB-D or stereo input, odometry and extrinsics defining the position of the sensor in relation to the base of the robot. The inputs are then synchronized, and the Short-Term Memory (STM) creates a node that stores the odometry pose, raw sensor data. RTAB-Map has a memory management approach that limits the size of the graph in order to operate in large scale environments. RTAB-Map's memory consists of Working Memory (WM) and Long-Term Memory (LTM). When a node is transferred to LTM, it is no longer available for modules in WM. When RTAB-Map's update time exceeds the fixed time threshold, some nodes in WM are transferred to LTM to limit the size of WM and reduce the update time. The nodes that remain in WM are determined by a weighting mechanism to identify which locations are more reliable than the others. The outputs provided are Map Data which includes the latest added nodes with sensor data and graph, Map Graph, Corrected Odometry, 3D occupancy grid, Dense Point Cloud and 2D occupancy Grid. Figure 3.4 summarizes the main blocks of RTAB-Map [110].

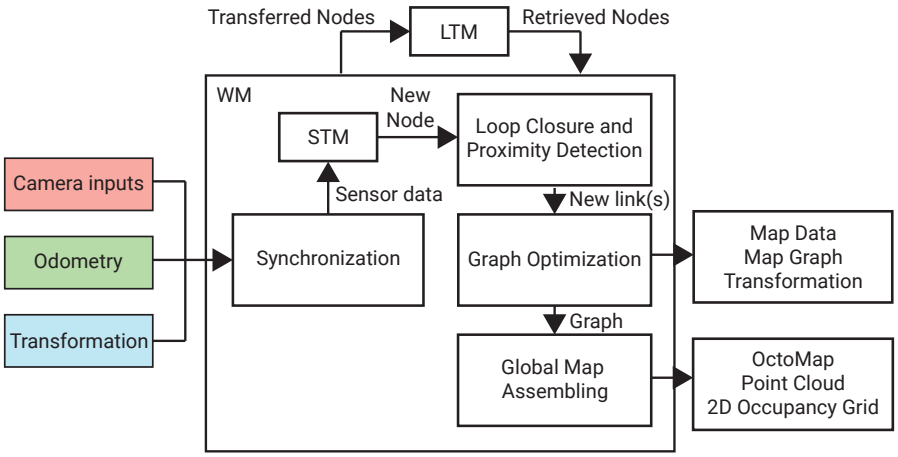

Figure 3.4: Block diagram of RTAB-Map: The algorithm gets required inputs that include the transformation that defines the sensors' position to the robot's base, camera inputs, and a chosen odometry from any source. After sensor synchronization, the short-term memory (STM) creates a node memorizing the odometry pose and the sensor's data. These inputs are fed to the graph-SLAM, and we get the Map Data containing the latest added node with sensor data and the graph as output. The 3D and 2D occupancy grids are optional [110].

Visual odometry was chosen as the input of the RTAB-Map which uses Stereo or RGB-D images as shown in Figure 3.5 from [110]. The Frame-To-Frame (F2F) approach is adopted which registers each new frame against the last keyframe. For feature detection, GoodFeaturesToTrack (GFTT) + ORB are used. For Stereo images, stereo matching is performed using the optical flow based on Lucas-Kanade's iterative method. Feature matching, applies the optical flow directly on the features without computing the descriptors allowing a faster matching. Motion prediction is a model for predicting the location of features in the current frame, based on previous transformations. This limits the search window when matching. This is useful in dynamic environments or with repetitive textures. After the matches are computed, the transformation is calculated by the RANSAC Perspective-n-Point (PnP) method. The resulting transformation is refined by the local bundle adjustment on the features of the last keyframe. Finally, if the number of inliers calculated during the motion estimation is below a fixed threshold, the keyframe is replaced by the current frame.

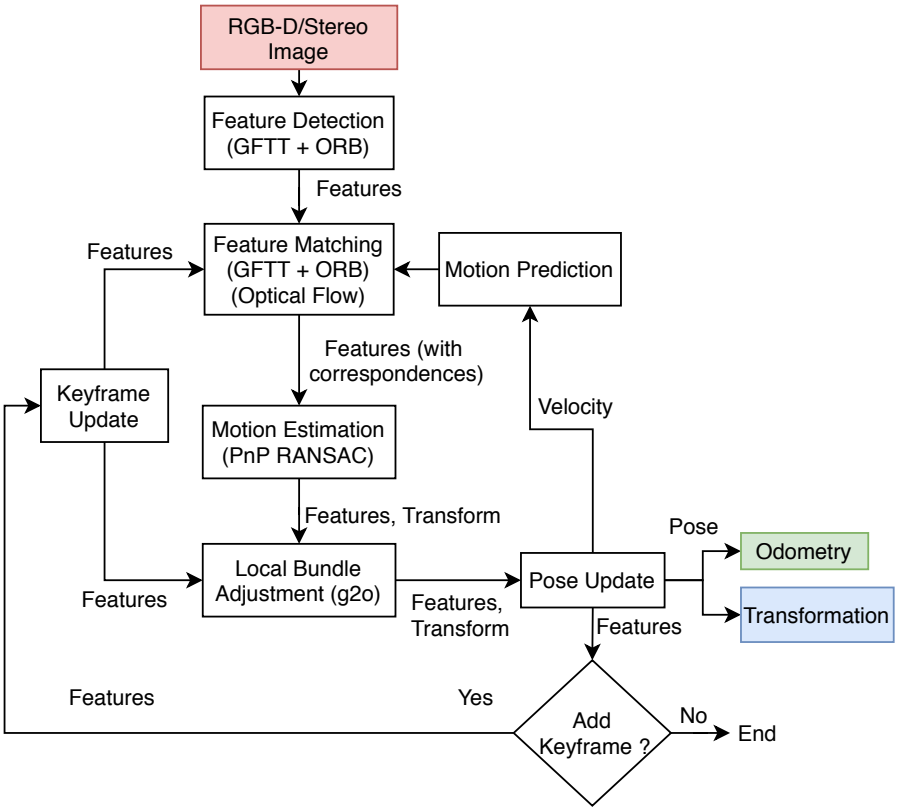

Figure 3.5: RTAB-Map's Visual Odometry block diagram of Frame-To-Frame (F2F) approach: The process starts with applying GoodFeaturesToTrack (GFTT) and ORB on the captured frame to detect features, then Features are matched using optical flow without having to compute the descriptors. Motion prediction predicts where features will lie in the current frame based on the previous motion transformation. Next, Perspective-n-Point (PnP) RANSAC used the computed correspondences to find the transformation of the current frame. The local bundle adjustment is used to refine the transformation, and finally, the pose is updated [110].

3.4 Study of the Sensor-Algorithm Parameters Coupling

Our parametric optimization protocol of sensor-algorithm coupling consists of evaluating the algorithm on different sequences with different acquisition modes. Then, the acquisition mode with the lowest error is used to globally tune the parameters of the algorithm. We focus on the optimization of the ORB-SLAM2 algorithm. ORB-SLAM2 has three main parameters related to the algorithm input, they are: the number of features, the FAST detector threshold and the depth threshold. The number of features relies on the FAST detector, which depends on the exposure [149]. Both parameters are not tied to the depth camera. We have identified a physically correlated parameter to the sensor:

the depth threshold [148]. This parameter allows the algorithm to classify near and far features. This parameter is tightly correlated to the error distribution of the RGB-D acquisition mode. We have evaluated the value of this parameter over a well-defined range. The various tests were carried out on a computing station, equipped with a 24-core Intel Xeon W-2265 processor running at 3.5GHz, 64GB RAM and an NVIDIA Quadro RTX 6000 graphics card with 4608 CUDA cores. We calculated the Euclidean trajectory error and the number of tracked points on the map for each frame. This protocol classifies input (Number of points tracked in the map seen by the current frame) and output (Euclidean distance between a current pose and the one in the referenced trajectory) conditions of the algorithm in order to qualify its performance. Inspired from [150, 151], we proposed a parametric optimization based on the following confusion matrix Equ. 3.2.

$$X = \begin{cases} TP & (E_i \leq s) \wedge (E_i \leq E_{i-1}) \wedge (M_i \geq M_{i-1}) \\ FP & (E_i \leq s) \wedge (E_i > E_{i-1}) \wedge (M_i < M_{i-1}) \wedge (M_i < \bar{M}) \\ TN & (E_i > s) \wedge (E_i > E_{i-1}) \wedge (M_i < M_{i-1}) \\ FN & (E_i > s) \wedge (E_i \leq E_{i-1}) \wedge (M_i \geq M_{i-1}) \wedge (M_i > \bar{M}) \end{cases} \tag{3.2}$$

X denotes the decision on a pose in the confusion matrix. TP stands for True Positive, FP for False Positive, TN for True Negative and FN for False Negative. E_i represents the Euclidean distance between a current pose and the one in the referenced trajectory. s is a given admissible positioning error. M_i represents the number of points tracked in the map seen by the current frame, and $\bar{M}$ represents the average number of points tracked in the map over the whole trajectory. A position is considered TP if its error is less than an admissible error and does not diverge (i.e., the current error is less than or equal to the previous error) and the input quality, represented in the number of tracked points, is not degraded. The FP positions have a reduced error but diverge, and the quality of the input is reduced. TN positions have a significant error that diverges, and the quality of the input is degraded. Finally, the FNs are defined by a significant converging error, and the input quality is good.

After calculating the position in the confusion matrix of each point, the Receiver Operating Curve (ROC) curve is plotted for all parameter values. The ROC is a plot used to diagnostic the performance of classification model using two parameters: True Positive Rate (TPR) and False Positive Rate (FPR) defined as follow:

$$\begin{aligned} TPR &= \frac{TP}{TP+FN} \\ FPR &= \frac{FP}{FP+TN} \end{aligned} \tag{3.3}$$

The optimum parameter value is identified as the one with the highest True Positive Rate (TPR) and the lowest False Positive Rate (FPR).

3.5 Experimental results

This study was carried out using the Intel RealSense D435i camera. The ORB-SLAM2 and RTAB-Map are run on different datasets. Identification of the best-suited acquisition mode is first investigated. Next, a sensor-algorithm parameters coupling is carried out through a parametric optimization protocol. Then, the depth-based method is compared to stereo based method. The translation and rotation errors are evaluated for each dataset compared to the referenced trajectory. Finally, the effect of the projector on trajectory quality is investigated.

3.5.1 Comparison of Depth-based SLAM and Stereo-based SLAM algorithms

In this section, we compare Depth-based SLAM algorithms against their Stereo-based SLAM version. First, the depth-based SLAM using RGB images and IR images (with the projector off) is tested to find the optimal mode in each environment.

3.5.1.1 Passive IR-D SLAM vs Passive RGB-D SLAM

The D435i camera is equipped with an RGB camera and two IR cameras. The RGB camera is a rolling shutter type and has a Field of View (FOV) of 69.4° × 42.5°, while

the IR camera is a global shutter type and has a Field of View (FOV) of 86° × 57°. We examine the distinction between the RGB and IR images regarding the quality of the trajectory. Settings required to ensure proper operation of the algorithm on the dataset are shown in the Table 3.1 and 3.2. The number of features has been set to 1000 for a narrow environment (Digiteo_seq1 & Digiteo_seq2) and 2000 for a wide environment (Digiteo_seq3).

Parameter	RGB-D	IR-D
Number of features	1000/2000	1000/2000
Depth threshold coefficient	20	50
FAST initial threshold	20	20
FAST min threshold	7	7

Table 3.1: Passive RGB-D SLAM and Passive IR-D SLAM front-end parameters for ORB-SLAM2

Parameter	RGB-D & IR-D
Odometry strategy	Frame to Frame (F2F)
Feature detector	GFTT + ORB
Motion Prediction	0
Motion Estimation	20

Table 3.2: Passive RGB-D SLAM and Passive IR-D SLAM front-end parameters for RTAB-Map

Passive RGB-D ORB-SLAM2 and Passive IR-D ORB-SLAM2 trajectories are plotted on the same Figure 3.6. Same for the RTAB-Map in the Figure 3.7. The translational error and the rotational error for Passive RGB-D SLAM and Passive IR-D SLAM are computed with respect to the referenced trajectory. Table 3.3 summarizes the results.

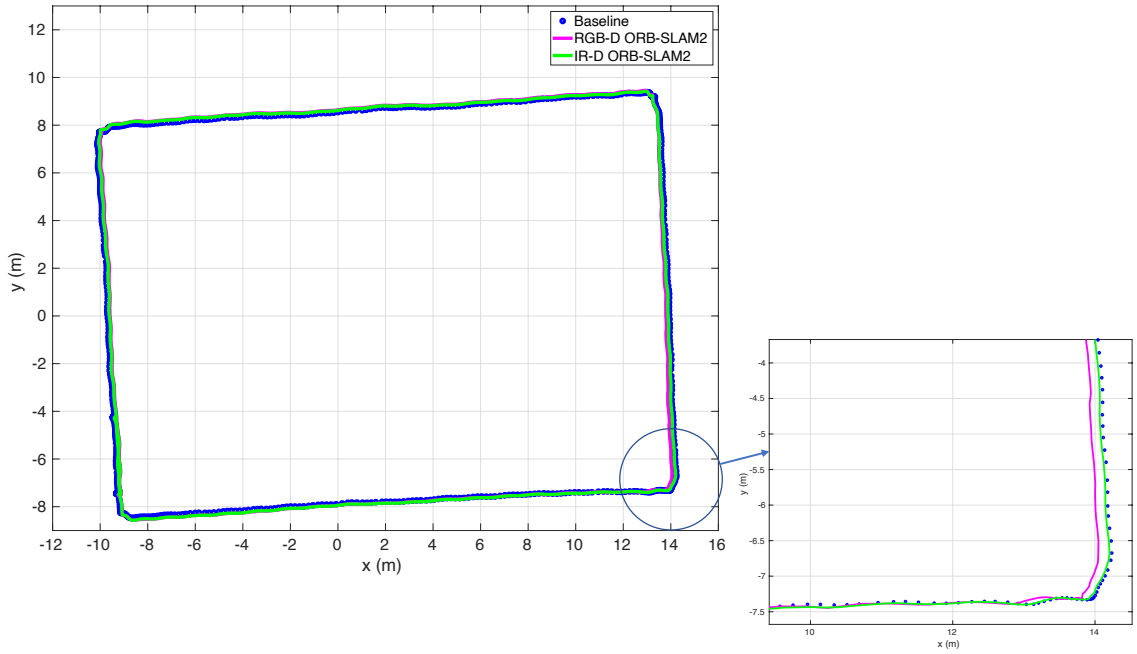

Figure 3.6: Passive RGB-D ORB-SLAM2 vs. Passive IR-D ORB-SLAM2 in Digiteo_seq1, the baseline is plotted in blue, RGB-D ORB-SLAM2 in magenta and IR-D ORB-SLAM2 in green. We can see in the zoom on the right that the IR-D follows the baseline perfectly, while the RGB-D drifts during a rotation.

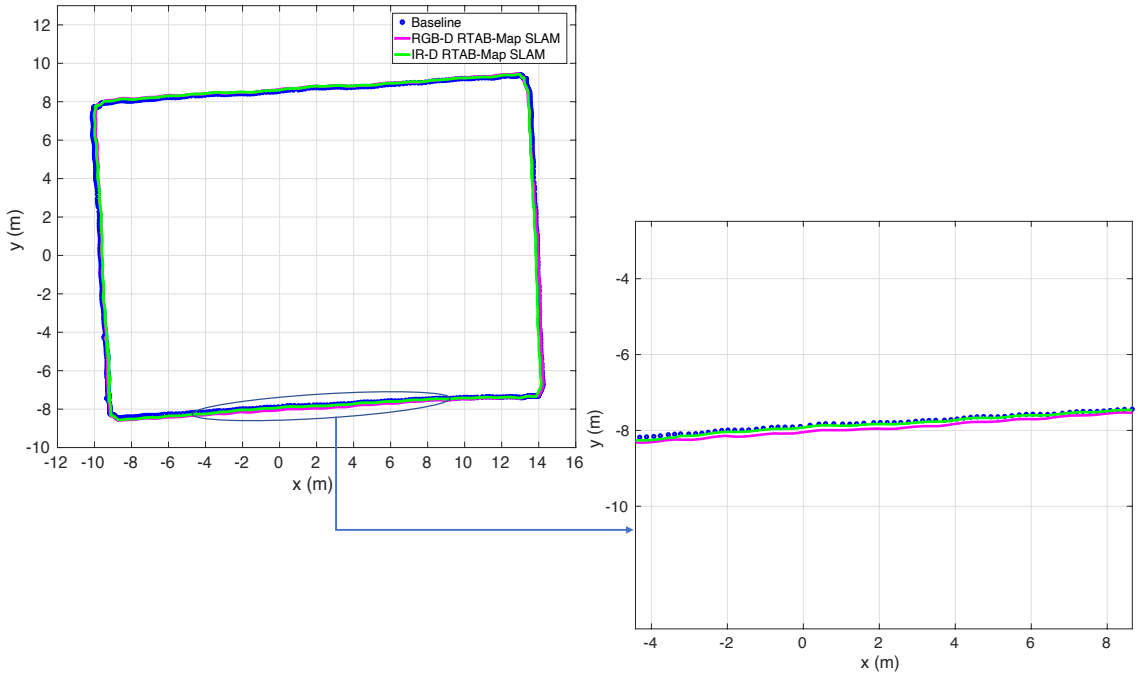

Figure 3.7: Passive RGB-D RTAB-Map SLAM vs. Passive IR-D RTAB-Map SLAM in Digiteo_seq1, the baseline is plotted in blue, RGB-D RTAB-Map in magenta, and IR-D RTAB-Map in green. The zoom in the right shows again that the IR-D follows the baseline perfectly, while the RGB-D is slightly offset.

		Passive RGB-D		Passive IR-D		RTAB error %	ORB error %
		RTAB	ORB	RTAB	ORB		
Digiteo_seq1	Tr (m)	0.13	0.09	0.1	0.08	23.08 ↓	11.11 ↓
	Rot (°)	29.6	15.31	27.98	11.1		
Digiteo_seq3	Tr (m)	0.21	0.28	0.21	0.16	0	42.86 ↓
	Rot (°)	19.46	9.48	19.91	9.39		

Table 3.3: Translational and Rotational Error of ORB-SLAM2 and RTAB-Map SLAM with Passive IR-D and Passive RGB-D

According to the results of Table 3.3, it is worth noting that with the IR-D mode, for the ORB-SLAM2, the error is minimized by 1cm for a narrow environment and 12cm for a wide environment scene. Accuracy improvement is mainly due to the characteristics of the sensor used in each situation. The IR-D sensor provides a wider field of view, allowing visual landmarks to be tracked over more extended periods, which should increase the accuracy of pose estimation and increase robustness since the visual overlap between successive images is greater [152]. In addition, the IR camera has a global shutter, unlike the RGB camera, which is a rolling shutter type. A rolling shutter camera exposes the lines sequentially with a delay, which causes significant distortion for fast-moving objects or those exposed to sudden brightness changes. Ignoring the rolling shutter can lead to significant drift in the estimated trajectory and inaccurate 3D reconstruction [153]. Another factor contributing to this difference in performance is that RGB-D alignment can be inaccurate due to uncertainties caused by hardware synchronization and digitization imperfections, unlike IR images which represent the advantage of being perfectly aligned, calibrated, and overlapped with the depth maps and are perfectly time-synchronized [154]. For the RTAB-Map, we can see no difference between the two modes in the basement parking environment, whereas the error is reduced by 3cm in laboratory corridors. As RTAB-Map has many parameters, it is not easy to find the proper parameters directly related to the sensor to visualize the impact of the sensor-algorithm coupling, unlike ORB-SLAM2, which has only a few parameters.

3.5.1.2 Depth threshold optimization

Since the IR-D mode gave better results in localization of at least 11%, it was retained for an in-depth study according to the depth threshold of the ORB-SLAM2. To this end,

we performed an automated test of the different values from 5 to 250, corresponding by multiplying by the baseline (0.05m) at an interval of 0.5m to 15m. We have empirically chosen a tolerable error s of 7cm and the average number of tracked points $\bar{M}$ over the trajectory at 500 since several scenes in the sequence lack textures. For each sequence, a ROC curve was established, as shown in Figure 3.8. The curves have been limited to a reduced scope containing the global minimum.

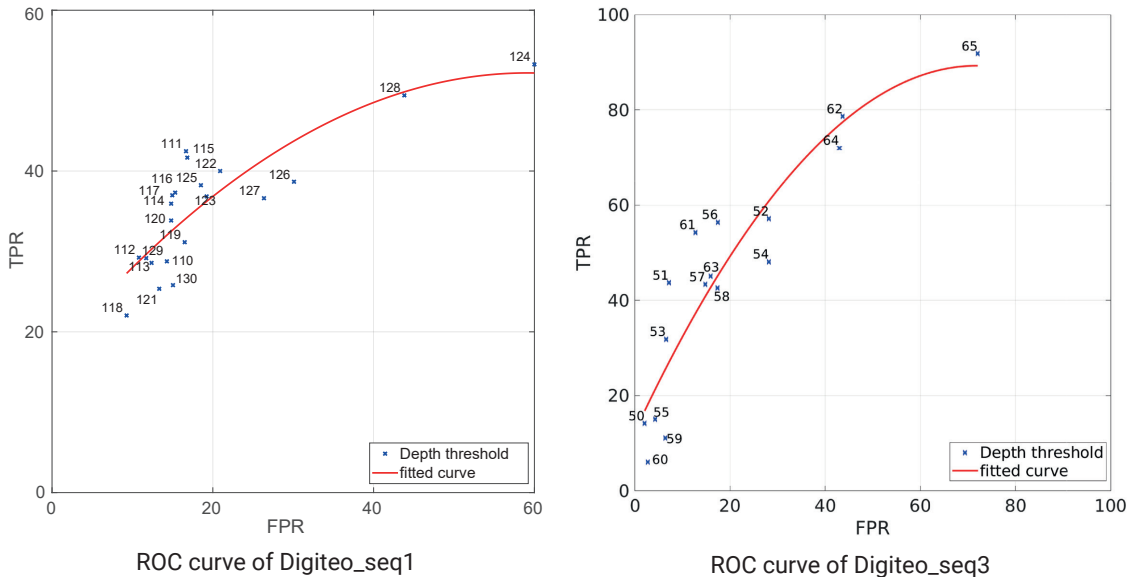

Figure 3.8: The ROC curve for different depth threshold coefficients. On the left, the ROC curve of the Digiteo_seq1, and on the right, the ROC curve of the Digiteo_seq3. The curves have been restricted to the interval containing the global minimum values.

Figure 3.9 shows the error as a function of the depth threshold coefficient in Digiteo_seq1 and Digiteo_seq3. The variation does not follow a specific pattern. Therefore, selecting an optimal value simply by tweaking the parameter directly is challenging. However, the suggested optimization method gives us an insight into the optimal value to select.

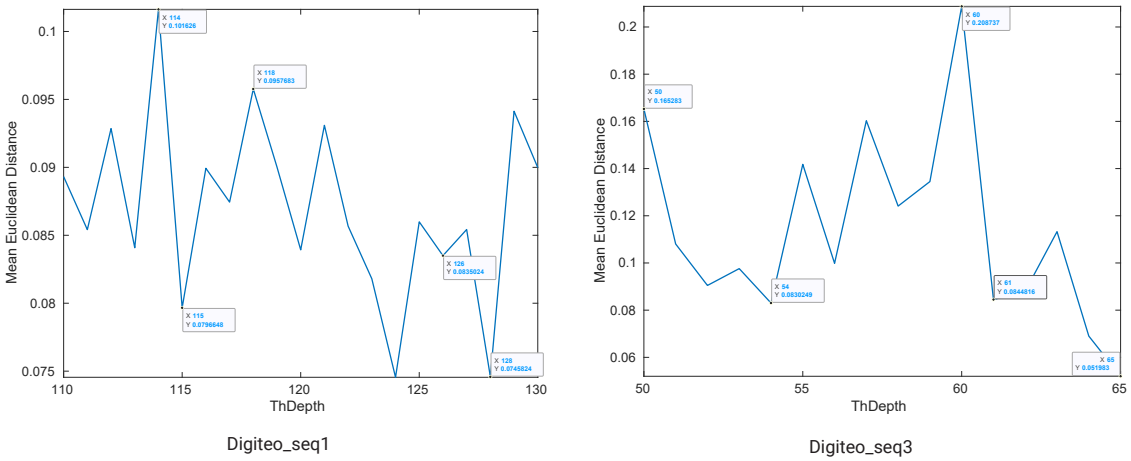

Figure 3.9: Variation of the MEE as a function of the depth threshold coefficient on the two sequences Digiteo_seq1 (right) and Digiteo_seq3 (left). The curve does not follow a specific trend, making it difficult to choose an optimal value.

Table 3.4 shows TPR, FPR, and the MEE values for different depth threshold values. The table shows that the values 115 and 128 allow a mean euclidean error (MEE) of 0.07m. As the mean of tracked points $\bar{M}$ is very high, we obtain many false positives for some parameter values. These points correspond to poses with an error lower than the admissible error but with a lower number of the mean tracked points $\bar{M}$, which means that the motion estimation is correct but unreliable. This kind of situation can happen in static environments. FPs are not tightly correlated to the average error of the trajectory. Indeed, FPs represent less reliable poses (better pose estimates with fewer tracked points) in contrast to TP (where the error is reduced and more points are tracked). Thus, we aim to get the optimal parameter value based on the maximum number of reliable poses.

Depth threshold coefficient	TPR	FPR	MEE(m)
114	0.35	0.14	0.1
115	0.41	0.16	0.07
118	0.22	0.9	0.09
126	0.38	0.3	0.08
128	**0.49**	**0.43**	**0.07**

(a) Digiteo_seq1

Depth threshold coefficient	TPR	FPR	MEE(m)
50	0.14	0.02	0.16
54	0.48	0.28	0.08
60	0.06	0.02	0.2
61	0.54	0.12	0.08
65	**0.91**	**0.72**	**0.05**

(b) Digiteo_seq3

Table 3.4: The values of TPR, FPR, and the MEE for different depth threshold values. The error variation as a function of the depth threshold does not follow a clear trend. However, we note that the coefficients with high TPR provide a reduced error.

The results of absolute trajectory error and relative pose error are shown in Table 3.5 for the selected optimal values.

	Optim. param. value	ORB IR-D (Tr/Rot)	ORB IR-D optim (Tr/Rot)	Error %
Digiteo_seq1	128	0.08m / 11.10°	0.07m / 11.11°	12.5 ↓
Digiteo_seq3	65	0.16m / 9.39°	0.05m / 9.4°	68.75 ↓

Table 3.5: IR-D errors in the ORB-SLAM2 after optimization

It should be noted that the depth threshold varies according to the environment. Also, the optimization has reduced the error by 68.75% in the Digiteo_seq3 dataset and 12.5% in the case of the Digiteo_seq1 dataset. This enhancement clearly shows how sensor-algorithm parameters coupling can significantly affect SLAM localization accuracy.

3.5.1.3 IR-D SLAM vs Stereo Vision SLAM

Based on the parameters found in Table 3.1,3.2, we compare IR-D ORB-SLAM2 and IR-D RTAB-Map to their stereo versions. The goal is to determine the rate of improvement in terms of accuracy of an RGB-D based algorithm versus a stereo based algorithm. The

IR-D SLAM and IR-Stereo Vision SLAM algorithms are compared with the referenced trajectory, where the translational and rotational errors are computed in Table 3.6.

		IR-Stereo		Passive IR-D		RTAB error %	ORB error %
		RTAB	ORB	RTAB	ORB(Optim)		
Digiteo_seq1	Tr (m)	0.14	0.14	0.1	0.07	28.57 ↓	50 ↓
	Rot (°)	29.11	11.08	27.98	11.11		
Digiteo_seq3	Tr (m)	0.21	0.23	0.21	0.05	0	78.26 ↓
	Rot (°)	20.23	9.39	19.91	9.4		

Table 3.6: Translational and Rotational Error of ORB-SLAM2 and RTAB-Map SLAM with Passive IR-D and Stereo

For the optimized ORB-SLAM2 IR-D, the error is reduced by 78.26% in the Digiteo_seq3 dataset and 50% in the Digiteo_seq1 dataset. For RTAB-Map, we have a similar result between IR-D and stereo for the Digiteo_seq3 sequence and a slight improvement in the Digiteo_seq1 sequence. These results confirm that the sensor's choice is not enough to improve the accuracy, but the algorithm parameters must also be optimized according to the perception system.

3.5.2 RGB-D SLAM: Active vs Passive

In this section, we compare the impact of the IR projector turned on and turned off on the trajectory for two sequences. The results were compared to the referenced trajectory, as shown in Figure 3.10. The ATE and RPE are calculated in the Table 3.7.

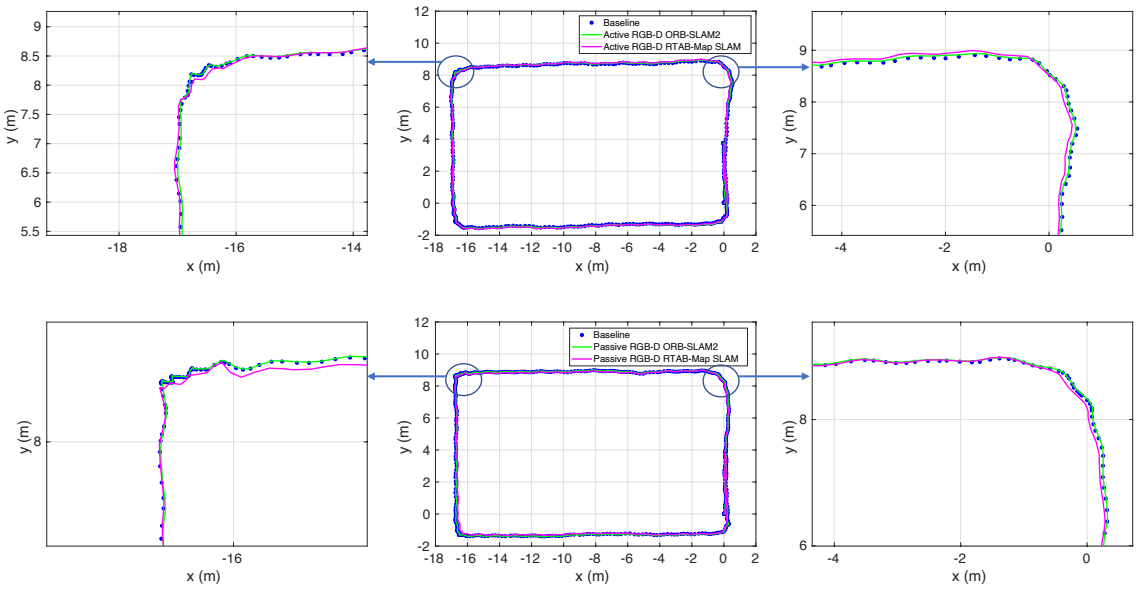

Figure 3.10: RGB-D ORB-SLAM2 (First row) and RGB-D RTAB-Map SLAM (Second row): Active vs Passive mode in Digiteo_seq2 The right and left images represent the zoom in the trajectories. They show how the active stereo approach (represented in green) sticks well to the reference (represented in blue) in the turns, unlike the passive stereo (represented in magenta), which drifts in and after the turns.

		Passive RGB-D		Active RGB-D		RTAB error %	ORB error %
		RTAB	ORB	RTAB	ORB (Optim)		
Digiteo_seq2	Tr (m)	0.11	0.06	0.07	0.04	36.36 ↓	33.33 ↓
	Rot (°)	37.76	13.32	36.87	15.49		
Digiteo_seq3	Tr (m)	0.21	0.28	0.43	0.25	104.76 ↑	10.71 ↓
	Rot (°)	19.46	9.48	33.81	8.46		

Table 3.7: Translational and Rotational Error of ORB-SLAM2 and RTAB-Map SLAM with Passive and Active RGB-D

By analysing the errors, we can see that the trajectory's accuracy based on active depth is slightly better than that based on passive depth. This slight difference is due to the depth maps, which are denser in the active case allowing more features with a valid depth value. Also, we can see that RTAB-Map, in the case of Digiteo_seq3, requires a study of its parameters to take advantage of the projector.

3.6 Conclusion

In this chapter, we presented the evaluation of various trajectories based on the camera's different modes of acquisition. Passive IR-D SLAM vs. Passive RGB-D SLAM was compared, and it was deduced that the depth threshold parameter for ORB-SLAM2 does not follow a specific trend. A method based on the ROC curve was established to find an optimal depth threshold value for the IR sensor. Using an IR camera compared to the RGB camera decreased ATE error by 23.08% for RTAB-Map in Digiteo_seq1 and 82.14% for optimized ORB-SLAM2 in Digiteo_seq3. The use of the D435's IR camera offers a significant advantage as it has a larger field of view for tracking features in blind spots, and the fact that the depth maps are also aligned with the left IR camera means that no further alignment processing is required. Based on the parameters found in the IR-D vs. Passive RGB-D SLAM comparison, the IR-D SLAM algorithm was compared with Stereo Vision SLAM, and we found a decrease in the translational error of 78.26% when using IR-D data for ORB-SLAM2 in Digiteo_seq3 and 28.57% for RTAB-Map in Digiteo_seq1. Finally, we compared the active and passive modes. We deduced that the active mode gives a more dense depth map; therefore, we get more accurate results since more features are used to calculate translation, rotation, and scaling. An RGBD-based SLAM system design must establish a strong coupling of the sensor's algorithm's parameters, especially those related to the field of view, depth threshold, and IR projector. Considering sensor characterization can increase the localization accuracy for robotics applications in indoor environments. In the next chapter 4, we will extend the HOOFR-SLAM toward an RGB-D sensor. Based on the results of this chapter, we will apply the optimization protocol to find the appropriate configuration to ensure high localization accuracy. We will also use further algorithmic optimization to improve the performance.

Chapter 4

RGB-D HOOFR-SLAM

4.1 Introduction

Simultaneous RGB-D localization and mapping (SLAM) have gained popularity due to the low cost and advantages of the RGB-D camera. Several efforts have been made to develop RGB-D SLAM. Unfortunately, these works have not been evaluated and extended to outdoor vehicle applications. In this chapter, we present an extension of HOOFR-SLAM to an enhanced RGB-D modality applied to an autonomous vehicle in a dynamic outdoor environment. We propose a feature filtering method based on depth maps to improve the algorithm's performance in dynamic environments. Additionally, algorithmic optimizations have been made to improve performance. In the previous chapter, we have seen how sensor-algorithm coupling is essential in enhancing localization accuracy in indoor environments. In this chapter, we rely on this optimization protocol to improve HOOFR-SLAM in outdoor environments. Finally, we use a hardware-in-the-loop (HIL) approach to validate the algorithm on an embedded architecture and a dataset collected by an instrumented vehicle of the laboratory.

4.2 Related Works

There are two types of visual SLAM: feature-based and direct methods. Because of their low complexity, feature-based algorithms are the most commonly utilized for embedded

real-time processing [8, 109]. However, this SLAM category is challenging in an outdoor environment with dynamic objects such as pedestrians, cars, and others. Dynamic objects are a significant annoyance while mapping and tracking. When the sky is cloudy, several outliers can be detected on the clouds, according to [155]. Several works have been conducted to handle this issue using semantic segmentation based on Deep Neural Networks (DNNs) or Convolutional Neural Networks (CNNs) [155, 156, 157, 158]. Filtering features improve localization accuracy. However, this pre-processing is an additional task that increases the computation time. Because of the processing time issue, embedding such systems in vehicles is not appropriate.

4.2.1 RGB-D SLAM

RGB-D sensors solved the problem of scale from which monocular SLAM suffers. They even outperformed the stereo sensors [159] by providing a dense depth map, allowing the 3D reconstruction of the environment with less complexity. The stereo computing is offloaded to the camera's built-in CPU, thus reducing the computation of the front-end part of the algorithm, which favors this type of sensor for embedded systems [142]. In this context, several algorithms have been developed using the RGB-D sensor. A well-recognized algorithm developed by R. Mur-Artal et al. is monocular ORB-SLAM [1], which has been extended with an RGB-D input [76]. ORB-SLAM2 uses depth information to generate the stereo coordinates, thus making the algorithm agnostic to the input type. RGB-D ORB-SLAM2 has only been evaluated on indoor datasets [84]. A geometric point and line constraint model (PL-GM) using an RGB-D camera has been proposed by C. Zhang [160]. This model uses the ORB extractor [161], Line segment detector (LSD) [162], and the depth map to retrieve the 3D points and lines; these are combined with the 2D points and lines to construct a geometric constraint model. The algorithm was evaluated only on two indoor public datasets [84, 92]. To sum up, the various RGB-D SLAM algorithms proposed in the literature have only been assessed on indoor datasets, which raises the question of how well they perform in an outdoor environment?

4.2.2 Feature filtering

To achieve more precise localization, several researchers are concentrating on developing solutions to the problems presented by visual SLAM in dynamic environments. M. Kaneko et al. [155] proposed a method that is based on deep learning semantic segmentation (DeepLab v2 [40]) to generate masks that filter out the detected features on mobile objects and the sky. This method solves the problem that when dynamic objects occupy a large part of the image, the ratio of outliers is dominant with respect to the features belonging to the carrier motion, which makes the RANSAC method inefficient. However, semantic segmentation is time-consuming and prevents the system from operating in real-time. J. Lee et al. [163] incorporated ORB-SLAM2 with semantic segmentation based on deep learning. Segmentation is applied to downsampled keyframes in parallel with the mapping thread to overcome the real-time execution problem. Another RGB-D SLAM algorithm by H.Wei et al. [5] uses the GMS (Grid-based Motion Statistics) feature point matching method with the K-means clustering algorithm to identify dynamic areas in the images and preserve the static information of dynamic environments. So the algorithm increases the number of reliable feature points while keeping the environment features. The algorithm was evaluated on the public indoor dataset [84] and showed improved performance compared to ORB-SLAM2. The DS-SLAM algorithm introduced by C. Yu et al. [157] represents an implementation of semantic segmentation using SegNet [164] combined with a motion consistency checking method to reject the features on moving objects. The algorithm showed significant improvement when compared to ORB-SLAM2 on the TUM RGB-D dataset [84]. Also, the segmentation thread is executed with a frame rate of 26 FPS, and the whole algorithm is 16 FPS. W. Xie et al. [165] used MaskRCNN to segment active moving objects (humans) and the mask inpainting method to repair the incomplete mask. They used motion detection based on Lucas–Kanade optical flow method for passive moving objects such as human-pushed chairs. This method includes motion and stillness recognition modules, making motion detection under a moving camera more reliable. The algorithm was evaluated in an indoor environment, and the process of semantic segmentation and mask inpainting is a cumbersome task for real-time embedded systems. Y. Liu et al. [166] used dynamic feature detection

using double K-means clustering and a static weight calculated using a static probability and the number of static observations to determine whether each feature is static or not. They also proposed a modified version of RANSAC based on static weights to improve its robustness in dynamic environments. The algorithm has been evaluated only on an indoor dataset and faces some issues in environments with high dynamics or in the case of high or low illumination. Barsan et al. [167] solved the problem of dynamic environments by simultaneously reconstructing the background, moving and potentially moving objects separately. They use an instance-aware semantic segmentation (Multi-task Network Cascades (MNC) [168]) to recognize dynamic and potentially dynamic objects from a single frame. The system runs at 2.5 FPS when evaluated on a PC. They identified that the instance-aware semantic segmentation was the primary bottleneck, making the algorithm unsuitable for real-time positioning tasks. Table 4.1 presents a summary of the SLAM systems presented above.

As our thesis aims to design a SLAM system for autonomous vehicles, the throughput rate must be 20 to 30 FPS to respect real-time constraints. The challenge is to find a trade-off between speed and localization accuracy in a dynamic environment. Therefore, we chose to implement a feature filtering based on depth consistency assumption, i.e., only the keypoints with a depth value consistent with the depth values of neighboring pixels are used. This approach demonstrated an accurate localization with real-time performance.

Algorithm authors	Dyn. env method.	Sensors	Hardware	Processing rate (FPS)	Dataset
J. Lee et al. [163]	Semantic segmentation applied to downsampled keyframes	Monocular	PC CPU/GPU	30	KITTI [89]
Y. Liu et al. [166]	Double K-means clustering	RGB-D	PC CPU	20.83	TUM [84]
H.Wei et al. [5]	Grid-based Motion Statistics + K-means	RGB-D	PC CPU	17	TUM [84]
C. Yu et al. [157]	Semantic segmentation	RGB-D	PC CPU	16	TUM [84]
M. Kaneko et al. [155]	Deep learning semantic segmentation	Monocular	PC CPU	7 [163]	CARLA driving simulator [169]
Barsan et al. [167]	Instance-aware semantic segmentation	Stereo	PC CPU	2.5	KITTI [89]
W. Xie et al. [165]	MaskRCNN + Mask inpainting	RGB-D	PC CPU	0.9	TUM [84] / ICL-NUIM [106]

Table 4.1: Summary of relevant properties of presented dynamic SLAM algorithms

4.3 RGB-D HOOFR-SLAM algorithm

4.3.1 Stereo HOOFR-SLAM algorithm overview

HOOFR-SLAM is a visual SLAM algorithm based on the Hessian ORB - Overlapped FREAK (HOOFR) bio-inspired extractor. This extractor, composed of the ORB detector with the Hessian score and the bio-inspired FREAK descriptor with enhanced overlap, has improved reliability and runtime during matching. HOOFR results showed competitive performance with SURF and SIFT with faster speed and low computational cost like ORB, but exceeding this latter in performance [57]. The Stereo HOOFR-SLAM consists of two main blocks, as shown in Figure 4.1. The first block is devoted to sensor data processing and ego-motion estimation, known as the front-end task. The second block represents the SLAM kernel and consists of the error graph optimization and loop closure tasks. The algorithm gets as an input stereo image. The left image is used to estimate the relative motion of the camera. The right image is used to compute the scale using stereo triangulation. For each frame, the HOOFR extractor is applied to detect and describe the features used for pose estimation and loop detection. The stereo matching provides the real scale by computing the ratio of the real distance of the landmarks and their triangulated distances. In the mapping thread, the features are matched with those of the previous left images. Each previous image with a successful transformation estimation is called the previous neighbor frame (PNF). The translation, rotation, and landmark positions are extracted from the essential matrix using triangulation. Due to its high processing cost, the bundle adjustment is substituted by windowed filtering to estimate the current camera position from a set of PNFs. Each predicted pose of the PNFs is associated with a confidence weight, so the optimal pose is the mean of all predictions by their respective weights [8]. The loop closure thread is executed in parallel with the mapping thread. Each left frame is queried in the set of keyframes to find the max likelihood. The current image is considered a new keyframe in a low-match score case and added to the pose graph. Potential loop closure is detected only when the pose of the keyframe with a high matching score is far from the current frame in the pose graph. The loop closure is then validated by computing the relative transformation between the current frame and the matched keyframe [8].

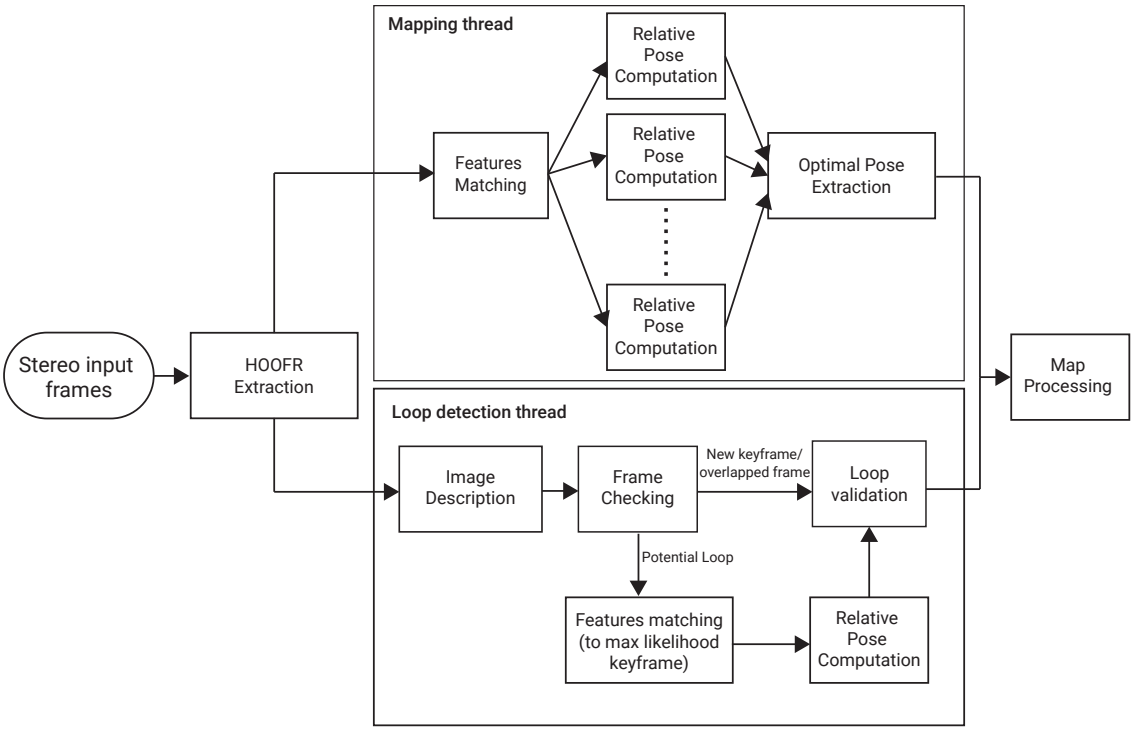

Figure 4.1: Functional blocks of the algorithm Stereo HOOFR-SLAM. The algorithm takes stereo images as input. The HOOFR extractor is applied on the left image to detect and describe keypoints. Two threads are launched in parallel: The mapping thread used for odometry and the loop detection thread used for loop closure. Finally, the pose graph is optimized in the map processing block [8].

4.3.2 RGB-D HOOFR-SLAM

The RGB-D HOOFR-SLAM takes as input RGB images and the corresponding depth maps. The RGB images must be pre-aligned to the depth maps. This step involves establishing the correspondence between the RGB pixels and the depth value, since the RGB and depth images are sampled from two different spaced cameras. The alignment is done by projecting the depth value from the IR camera plane to the RGB camera plane. From the depth map, we estimate the corresponding 3-D coordinate $(x_{IRworld}, y_{IRworld}, z_{IRworld})$ from each pixel location (x_{IR}, y_{IR}) as below:

$$\begin{aligned} x_{IRworld} &= \frac{z_{IRworld}}{f_{IR}} \cdot (x_{IR} - x_{IRc}) \\ y_{IRworld} &= \frac{z_{IRworld}}{f_{IR}} \cdot (y_{IR} - y_{IRc}) \end{aligned} \tag{4.1}$$

where (x_{IRc}, y_{IRc}) is the principal point location and f_{IR} is the focal length of the IR camera. Then, the 3D coordinates in the IR camera frame are transformed into the 3D coordinate system defined by the RGB camera using an affine transformation.

$$\begin{bmatrix} x_{RGBworld} \\ y_{RGBworld} \\ Z_{RGBworld} \end{bmatrix} = [R|T] \begin{bmatrix} x_{IRworld} \\ y_{IRworld} \\ Z_{IRworld} \end{bmatrix} \tag{4.2}$$

where $R \in \mathbb{R}^{3\times3}$ is the rotation matrix and $T \in \mathbb{R}^{3\times1}$ is the translation vector. Finally, we can find the corresponding pixel in the RGB image by projecting the 3D coordinates defined in the RGB camera frame to the RGB image plane as

$$\begin{bmatrix} x_{RGB} \\ y_{RGB} \\ 1 \end{bmatrix} = \frac{f_{RGB}}{Z_{RGBworld}} \begin{bmatrix} x_{RGBworld} \\ y_{RGBworld} \\ Z_{RGBworld} \end{bmatrix} \tag{4.3}$$

with $f_{RGB}$is the focal length of the RGB camera. RGB-D alignment can be inaccurate due to uncertainties caused by imperfections in hardware synchronization and digitization. Therefore, we use the IR camera images instead of the RGB camera. The IR images represent the advantage of being perfectly aligned, calibrated, and overlapped with the depth maps. They are perfectly time-synchronized. Ultimately, it saves us the additional computational overhead of aligning color-to-depth [154].

In the Stereo HOOFR-SLAM, the workflow starts with features detection and description, followed by stereo matching. Then the mapping thread is launched. In RGB-D HOOFR-SLAM, we propose a modified HOOFR extractor based on feature filtering using depth maps provided by the RGB-D sensor. This approach improves localization accuracy by eliminating features with unreliable depth, including features detected on moving vehicles, clouds and those with invalid depth values (zero), as shown in Figure 4.2. This way, only the relevant keypoints are kept and the number of keypoints to be described and matched is less, which reduces the processing time.

Figure 4.2: Many of features are detected on the clouds, which implies that most points are too far away to compute translation; we can also notice points detected on the bus, which confuses the tracking and mapping process.

The depth-based filtering method uses the Median Absolute Deviation (MAD) from the median applied on depth maps, as detailed on the Algorithm 4.1.

Algorithm 4.1 MAD applied to depth maps

```
//Convert depth image to a vector Z
For each pixel do
   if (z_pixel > 0 and z_pixel<65535)
      Z ← z_pixel;
   end if
end for
//Find the median of the depth vector by partial sorting elements
//And then taking the middle value (odd case)
//or the average of the middle two values (even case)
z̄ ← findMedian(Z);
//Calculate the absolute difference for each observation from the median
For each z_pixel in Z
   V ← abs( z_pixel - z̄);
end for
//Estimate the standard deviation using the median of V
s ← 1.4826*findMedian(V);
```

This approach identifies outlier points based on their depth value. The method is robust against outliers and also computationally inexpensive. In this method, we compute the median $\bar{z}$ of depth pixels (including non-zero and non-saturated values). Then, we calculate the absolute difference for each depth value from the median. $V = |Z - \bar{z}|$. Finally, we estimate the standard deviation $s = 1.4826 \cdot median(V)$, where the factor 1.4826 was

chosen so that the expected value of s is equal to the standard deviation for normally distributed data [170, 171]. A point is considered an outlier if : $d > t \cdot s$, where d is the depth value, and t is the decision threshold [171]. Experimentally, we found that the value $t = 3$ gives the best results.

We tested the algorithm with and without filter on a sequence of 100 images on a laptop equipped with an AMD Ryzen 9 4900HS processor and 24GB in memory. Table 4.2 shows the comparison in terms of the absolute trajectory error (ATE) and the program's execution time. Filtering keypoints during the detection phase allowed a speedup of 1.45 and an error reduction of 80%, as only the relevant features are kept as illustrated in the Figure 4.3, thus reducing the number of points processed by the further stages.

Without MAD Filter		With MAD Filter		Speedup	ATE Evolution
Exec. time (ms)	ATE (m)	Exec. time (ms)	ATE (m)		
64.47	0.7955	44.22	0.1592	1.45	-80%

Table 4.2: A comparison of the RGB-D HOOFR SLAM algorithm with and without the MAD filter on a sequence of 100 images.

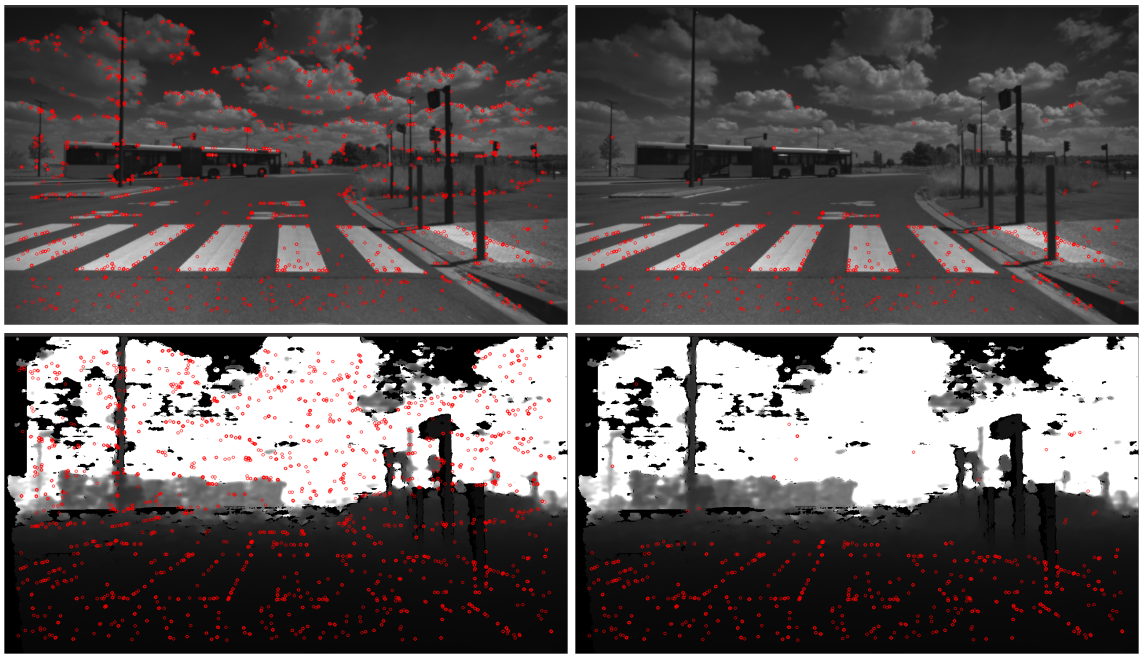

Figure 4.3: Keypoints distribution on the outdoor scene, before and after applying the filter.In the two images on the left, we see that most of the features are detected in the sky and on the bus. On the two images on the right, the keypoints are filtered.

4.3.3 PROSAC

After the filtering, description, and matching, comes the stage of the pose calculation. In the Stereo HOOFR SLAM version, the essential matrix was estimated by applying the RANSAC scheme with a sub-pixel measurement error (*me*) to have an optimized model without relying on optimization methods like the BA. Nguyen et al. found through experiments that a measurement error of inlier in the RANSAC scheme less than 0.4 allows a high localization accuracy. However, when applying a $me = 0.4$, the execution time dramatically increases. To solve this problem, Nguyen et al. proposed a solution to estimate the essential matrix twice, the first time with $me = 1$ and the second time using the inliers found previously with a $me = 0.4$. In the RGB-D HOOFR SLAM algorithm, since the number of interest points is reduced in the detection step, we estimate the essential matrix only once for a small value of *me*. In the worst cases, with few good points, RANSAC will take a long time to find the solution. Therefore, we have changed the RANSAC method to PROgressive Sample Consensus (PROSAC) as shown in the Algorithm 4.2. This method is based on a progressive sampling of the points starting with the top-ranked ones based on their quality factor, significantly saving computational time. In [172], the authors demonstrated that PROSAC was more than a hundred times faster than RANSAC, and in the worst case, they have identical behavior. We used Lowe's ratio test [173] to sort the matches. Lowe's ratio is computed from two distances, the best match distance, and the second best match distance. The best match is the one with the smallest distance, while the second best match is considered random noise. On this basis, if the good match cannot be distinguished from the noise at that moment, this good match must be rejected. So, the matches are ranked on this criterion. The closer the ratio is to 1, the lower the quality of this match.

Algorithm 4.2 Essential matrix estimation from matching set using PROSAC

1. Sort matches in ascending order based on Lowe's ratio $r = \frac{d_{bestmatch}}{d_{secondbestmatch}}$
2. Apply 5-points algorithm inside **PROSAC** scheme to the initial matching set with the measurement error equal to 0.4
3. Test the final optimal model on the whole initial matching set to select the inliers
4. Compute the mean of measurement errors returned by inliers from step 3. The inverse of this value represents the score of the estimated model.

Table 4.3 compares the essential matrix estimation (EME) execution time and the absolute trajectory error (ATE) of the two approaches, RANSAC and PROSAC, on a sequence of 100 images. By changing from RANSAC to PROSAC, we can see that the speedup is increased five times without losing too much accuracy.

RANSAC		PROSAC		Speedup
EME (ms)	ATE (m)	EME (ms)	ATE (m)	
13.61	0.1575	2.54	0.1595	$\times$**5.35**

Table 4.3: A comparison of the two methods RANSAC and PROSAC on a sequence of 100 images.

4.3.4 Measurement Error Optimization

To optimize the measurement error (*me*), we applied the optimization protocol presented in 3. We ran an automated script on an interval from 0.05 to 1. For a tolerable error s we choose 2.5m and the average number of tracked points $\bar{M}$ over the trajectory at 1800. The ROC curve shows that the optimal value is 0.7 with a True Positive Rate of 89%.

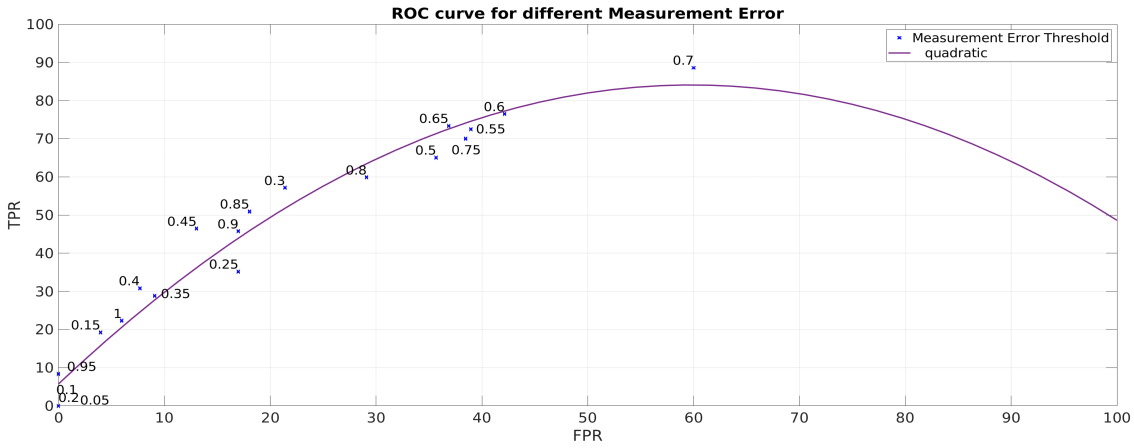

Figure 4.4: The ROC curve for different *me*. The ROC curve shows the distribution of several measurement error values as a function of TPR and FPR. The optimal value of the measurement error is 0.7, with a TPR of 89% and an FPR of 60%.

Figure 4.5 shows the evolution of the mean error against the measurement error, and Table 4.4 represents some measurement error values with their TPR, FPR, and associated error. We can see that the global minimum corresponds perfectly to the optimal value selected in the ROC curve, with the highest TPR. The *me* values with a zero TPR

and FPR indicate the parameter values that failed to meet the quality requirements set by the chosen thresholds.

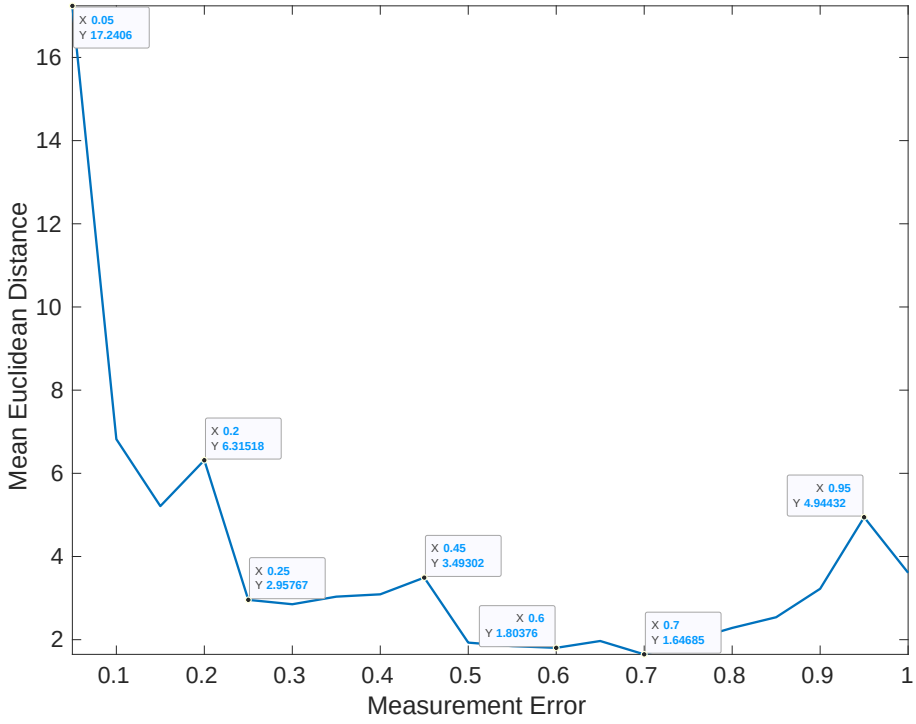

Figure 4.5: Evolution of the mean error against the measurement error computed on Sequence 1

Measurement errors	TPR	FPR	Error(m)
0.05	0	0	17.24
0.2	0	0	6.31
0.25	0.35	0.16	2.95
0.45	0.46	0.13	3.49
0.6	0.76	0.42	1.8
0.7	**0.89**	**0.6**	**1.64**
0.95	0.08	0	4.94

Table 4.4: The values of TPR, FPR, and the absolute error of the trajectory for different measurement error values. The optimal value is 0.7 with a True Positive Rate of 89% and a False Positive Rate of 60% .

4.4 Experimental results

The proposed RGB-D HOOFR-SLAM algorithm is evaluated on our sequences (Sequence 1 and 2 refer to Sec. 2.4.2). The accuracy of the visual SLAM system is measured

by absolute trajectory error in 2D, and we compared our method's accuracy and performance to that of Stereo HOOFR SLAM and other competitive algorithms. Finally, we used a HIL approach to evaluate the algorithm performance on an embedded architecture using our dataset and compared it with the performance on a PC.

4.4.1 RGB-D HOOFR SLAM vs Stereo algorithms

Comparing the RGB-D HOOFR SLAM to its Stereo version, as shown in Table 4.5, it can be seen that the Stereo HOOFR loses tracking at the beginning of the sequence; this is because of the strict conditions that have been set for the matching. If the positions of matches change slightly in images, the point is either too far from the camera or the camera is not moving much. These two cases do not provide a reasonable estimate, so these matches are rejected. In sequence 1, most points are detected on the clouds; therefore, the proportion of far points exceeds the near points leading to a rejection of the pose computation [8]. RGB-D HOOFR SLAM takes advantage of the depth map to overcome this problem and adapt the depth threshold to reject outlier points detected on the dynamic objects and in the sky. The RGB-D HOOFR SLAM outperformed the ORB-SLAM2 Stereo in localization accuracy, especially in FPS, with an improvement of 146.51%. In the ORB-SLAM2 Stereo, the ORB extractor occupies 50%, and the stereo matching occupies 31% of the tracking processing time [76]. The same goes for the HOOFR Stereo. 30% of the tracking processing time is saved when using an RGB-D camera. On the other hand, the ORB-SLAM2 does not perform upstream filtering, thus processing all the points and correcting the errors in the mapping phase using local BA, resulting in a significant processing time. Unlike the RGB-D HOOFR SLAM, the keypoints are filtered directly after detection, thus reducing the number of points to be processed and thus the execution time and outliers.

	RGB-D HOOFR		Stereo HOOFR		Stereo ORB		Evolution compared to "Stereo ORB"	
	RMSE (m)	FPS	RMSE (m)	FPS	RMSE (m)	FPS	RMSE (%)	FPS (%)
Seq1 (L: 978.57m)	**1.47**	**27.54**	Tracking lost	-	2.59	11.62	-43.24	+137.01
Seq2 (L: 189.81m)	0.93	**28.57**	12.31	10.05	**0.71**	11.59	+30.99	+146.51

Table 4.5: The Root Mean Square Error (RMSE) in meter and Frame rate Per Second (FPS) of the RGB-D HOOFR SLAM compared to Stereo SLAM algorithms

4.4.2 RGB-D HOOFR SLAM vs state-of-the-art RGB-D algorithms

This section compares the RGB-D HOOFR SLAM algorithm with other state-of-the-art RGB-D algorithms. For this comparison, we have selected two algorithms that are well-known for robustness and real-time operating and that have been evaluated in outdoor environments, namely ORB-SLAM2 [76] and RTAB-Map SLAM [110]. Figure 4.6 shows the trajectories of the three algorithms on the two sequences, the trajectories follow closely the ground truth and the RGB-D HOOFR SLAM represents a slight improvement in localization accuracy as shown in the Table 4.6 . The RGB-D HOOFR SLAM shows competitive results in localization accuracy and performance over the ORB-SLAM2 and RTAB-Map algorithms, mostly for sequence 1.

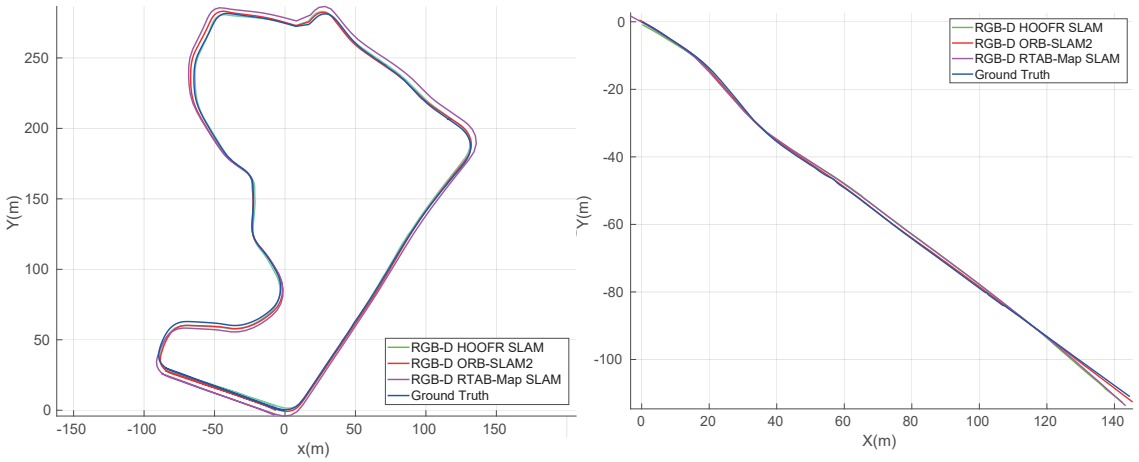

Figure 4.6: Sequence 1 and Sequence 2 trajectories plots: Trajectory plots of the three algorithms RGB-D HOOFR SLAM (in green), RGB-D ORB-SLAM2 (in red), and RGB-D RTAB-Map SLAM (in magenta) against the ground truth (in blue) over the two sequences

	RGB-D HOOFR		RGB-D ORB		Error evolution		RGB-D RTAB-Map		Error evolution	
	RMSE (m)	FPS	RMSE (m)	FPS	RMSE (%)	FPS (%)	RMSE (m)	FPS	RMSE (%)	FPS (%)
Seq1 (L: 978.57m)	**1.47**	**27.54**	2.27	17.11	-35.24	+60.96	4.39	13.77	-66.51	+100
Seq2 (L: 189.81m)	**0.93**	**28.57**	1.18	18.02	-21.19	+58.55	3.18	16.41	-70.75	+74.1

Table 4.6: The Root Mean Square Error (RMSE) in meter and Frame rate Per Second (FPS) of the RGB-D HOOFR SLAM compared to state-of-the-art SLAM algorithms

4.4.3 Evaluation of RGB-D HOOFR-SLAM on an Embedded Architecture

Following the improvement of the localization accuracy, an evaluation is performed with the matching block implemented on GPU [8] on PC and the Nvidia AGX Xavier embedded card. Results presented in Table 4.7 show the mean execution time decreased by shifting from CPU to CPU-GPU for the PC and the embedded board by about 1.28 speedup factor on the PC and 1.19 speedup factor on the embedded card.

	PC CPU (ms)	PC CPU-GPU (ms)	PC SpeedUp	AGX CPU (ms)	AGX CPU-GPU (ms)	AGX SpeedUp
Seq1 (L: 978.57m)	36.31	27.17	**1.33**	50.82	41.85	**1.21**
Seq2 (L: 189.81m)	35.08	28.10	**1.24**	52.53	44.73	**1.17**

Table 4.7: Average execution time of the algorithm in milliseconds on the two sequences on PC and the Nvidia AGX Xavier embedded board.

Table 4.8 represents the speedup of the block matching from CPU to GPU.

	PC CPU (ms)	PC GPU (ms)	PC SpeedUp	AGX CPU (ms)	AGX GPU (ms)	AGX SpeedUp
Seq1 (L: 978.57m)	9.79	2.73	**3.58**	7.71	3.01	**2.56**
Seq2 (L: 189.81m)	8.48	2.77	**3.06**	6.17	3.21	**1.92**

Table 4.8: Mean execution time of the matching block in milliseconds on the two sequences on PC and the Nvidia AGX Xavier embedded card.

We can see that switching from RANSAC to PROSAC in Table 4.3 allows greater speed up compared to GPU acceleration. In fact, in the RANSAC method, the number of iterations increases exponentially for small proportions of inliers. This problem has been solved with the PROSAC algorithm, which takes as input matches ordered by quality and gives them priority during sampling under the assumption that these matches have a high probability of being inliers, which leads to this significant speedup. As a result, by combining algorithmic optimizations and implementation on a heterogeneous architecture, we arrive at a trade-off between localization accuracy and real-time execution.

4.5 Conclusion

This chapter presented the RGB-D HOOFR SLAM algorithm, an enhanced algorithm of the Stereo HOOFR SLAM based on an RGB-D camera [8]. Filtering based on the

MAD has been envisaged in the outdoor environment. Afterward, algorithmic optimizations were performed on the pose estimation block by changing RANSAC to PROSAC. Next, the measurement error parameter was optimized using the protocol proposed in the chapter 3. The algorithm was evaluated on two sequences recorded with an instrumented laboratory vehicle. These datasets are the first RGB-D outdoor vehicle application datasets to our knowledge.

We compared the RGB-D HOOFR SLAM to its stereo version. Besides, the RGB-D HOOFR SLAM has been compared to other state-of-the-art algorithms regarding its performance and proved to be a good trade-off of execution time and localization accuracy. Our algorithm can operate in real-time at 27 FPS on a PC CPU and 19 FPS on an embedded processor without accelerating the processing on the GPU. The next chapter 5 will involve architecture-algorithm adequacy that will allow real-time processing on-the-fly of the RGB-D HOOFR SLAM algorithm on an embedded architecture.

Chapter 5

Hardware-Software codesign: Toward an FPGA Architecture Based Front-End Processing

5.1 Introduction

Feature-based SLAM systems are becoming increasingly popular due to their performance and robustness. Several feature extractors are used in various SLAM systems such as ORB [60], SIFT [173], and SURF [58]. Although these extractors yield good matching results, the computational complexity of feature extraction and matching is a significant hurdle when embedding such SLAM algorithms on low-power architectures. Recently, Nguyen et al. [10] proposed an FPGA implementation of the HOOFR extractor while maintaining the same accuracy. However, the matching task remains the most time-consuming task in the processing flow. The design of an accelerated architecture for this functional block is mandatory to achieve on-the-fly processing on a system-on-chip. Our challenge is to boost the algorithm's performance on low-power architectures to ensure on-the-fly processing. FPGAs are considered the best choice for stream processing. Contrary to GPUs, which only provides parallelism for data processing and acceleration, FPGAs can provide data, task, and pipeline parallelism, which makes them more suitable for stream processing [174, 175] especially for embedded systems.

We achieve our objective through an algorithm-architecture mapping applied to CPU-FPGA architectures. In practice, an algorithm is broken down into functional blocks, and each block is assigned to the appropriate processing unit, ensuring optimal performance. In this chapter, we evaluate the performance of the matching block on different architectures since it is the bottleneck of performance, concluding with the proposition of an optimal CPU-FPGA mapping for the RGB-D HOOFR-SLAM front-end.

5.2 Related Works

Several works have recently focused on implementing visual SLAM algorithms on embedded architectures. In particular, indirect-based algorithms are gaining attention due to their low complexity and ease of parallelization. Much effort has been devoted to accelerating the most time-consuming part of the algorithm, including feature extraction and matching. Fang et al. [176] proposed a design of ORB feature extractor since it is the bottleneck of performance and energy consumption. They implemented the extractor on an Altera Stratix V FPGA and achieved 67 frames per second using VGA-resolution images. The design runs at 203MHz frequency, reducing the energy consumption. Liu et al. [9] presented eSLAM, an implementation end-to-end ORB-based SLAM system on a SoC. The authors accelerated the extraction part and extended the hardware implementation to the matching block. First, the descriptor was reformulated into a rotationally symmetric pattern to simplify the hardware implementation. Then, the extractor and the matching blocks were implemented as a pipeline operating in two modes. The extractor and matching are launched in parallel with the pose estimation and optimization in normal frames mode. In the keyframes mode, feature extraction runs in parallel with pose estimation and optimization. The implementation has been conducted on a Xilinx Zynq XCZ7045 SoC architecture. The algorithm was evaluated on the TUM dataset [84], achieving a frame rate of 55 FPS for normal frames and 31 FPS for keyframes with a power consumption of 1.936W. Vemulapati et al. [177] proposed a SoC-based ORB-SLAM by accelerating the feature extraction and matching on FPGA since they consume 60-70% of the runtime of the algorithm. The rest of the algorithm is run on the ARM cores. In order to reduce resource utilization, they reduce the bit-depth of each pixel at different stage of the ORB

pipeline and quantize the pixels to 6 bits per pixel with a 25% reduction of Flip-Flop resources and a drop of 3.9% in accuracy. Compared to eSLAM, the loss of accuracy is much lower. The algorithm was implemented on a Xilinx ZU3EG FPGA-SoC on-board with a quad-core ARM and evaluated on the TUM dataset [84]. The algorithm achieved 62 FPS with an energy consumption of 4.6W. Nguyen et al. [10] proposed a HOOFR feature extraction architecture with a maintained complexity of the HOOFR algorithm to ensure a similar detection result on hardware as on software. The algorithm uses bucketing to ensure the homogeneous distribution of keypoints intended for SLAM applications. The algorithm was implemented on Arria 10 SoC-FPGA and achieved a frame rate of 26 FPS at 1280×720 pixels.

Following Nguyen et al.'s work [10], we propose an architecture combining the HOOFR extractor and the matching blocks since the high computational intensity of feature extraction and matching makes running the algorithm on low-power embedded platforms very challenging. The HOOFR-SLAM algorithm has been decomposed into blocks and evaluated on different architectures to achieve an optimized partitionning (to assign each block to the appropriate processing unit), considering data transfer constraints between processing units. Our ultimate goal is to ensure processing at a adequate rate for autonomous vehicle applications (30 FPS) at a high-definition resolution.

5.3 HOOFR-SLAM Front-end Overview

The HOOFR-SLAM algorithm is composed of two threads running in parallel: Mapping thread and Loop detection thread. The front-end includes HOOFR extraction and the mapping thread as shown in Figure 5.1. By measuring the processing time of each functional block on the CPU, as shown in Table 5.1, we notice that the most computationally expensive blocks are detection (50% of front-end execution time) and matching (33% of front-end execution time).

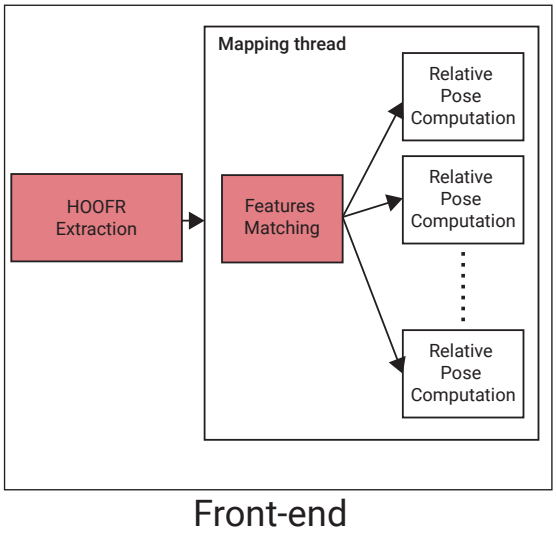

Figure 5.1: HOOFR-SLAM Front-end diagram, tasks that run on the front-end are the HOOFR extraction (including detection, Hessian score computing, filtering, and description), Features Matching, and Relative Pose Computation. In red, the most computationally expensive blocks are detection (50% of front-end execution time) and matching (33% of front-end execution time).

	Functional block	Runtime on CPU (ms)
HOOFR Extraction	Feature detection (FAST/Hessian/Filtering)	**14.29**
	Description	1.68
Mapping thread	Matching	**9.69**
	Pose estimation	3.5
Back-end	Loop detection	5.22
	Map processing	0.126

Table 5.1: Processing time of each functional block on CPU.

5.3.1 Bucketing-based HOOFR extractor

Nguyen et al. [10] proposed the implementation of a bucketing-based HOOFR extractor on FPGA. The bucketing technique divides the image into a grid, as shown in Figure 5.2. The number of cells in the grid depends on the image's resolution. For each cell, keypoints are detected by the FAST detector. Then, the Hessian score is calculated for each point, keeping only the points with the highest score. After filtering the points, the description of each point is computed. The bucketing technique has the benefit of ensuring a homogeneous distribution of keypoints over the entire image, which improves the localization accuracy of the SLAM [76, 64] and allows a pipeline operation at the cell level, as

illustrated in Figure 5.2, meaning that when a kernel finishes its task on the cell, the next kernel starts processing immediately on this cell.

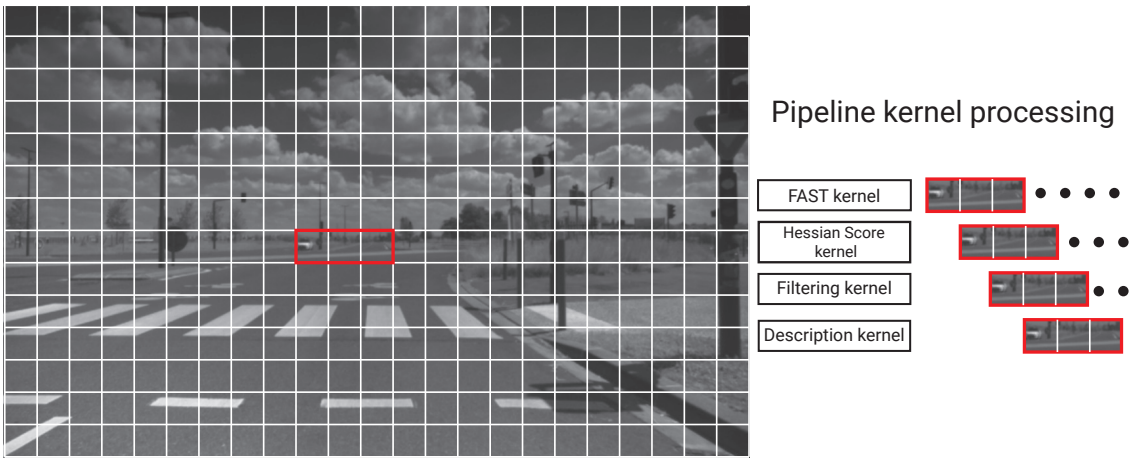

Figure 5.2: Image split into a grid. Image cells are processed in the pipeline. When a kernel finishes processing a cell, the following kernel takes over.

The architecture of the HOOFR extractor on FPGA is illustrated in Figure 5.3. The system includes four functional blocks (FAST, Hessian score, Filtering, and Description). These kernels are launched in parallel. At each image acquisition, the CPU copies the image in the global memory. Then, it consecutively launches the kernels for detection (FAST, Hessian score, and Filtering). The description block is launched once the integral of the image is computed and ready in the global memory. Finally, synchronization is performed when all the kernels are active, and the CPU waits to get the results. The FAST kernel and the Hessian score calculation have been duplicated because they are the bottlenecks of the algorithm flow. Testing the pixels of the whole image in high resolutions (for example, 921,600 pixels in the case of HD resolution) makes the FAST kernel very cost-intensive. Moreover, the latter returns several keypoints, which makes computing the Hessian score very costly. The communication between these different kernels is done using the Intel channel extension. Intel channels provide a mechanism for direct data transfer and synchronization between kernels via FIFO buffers without interaction with the host processor.

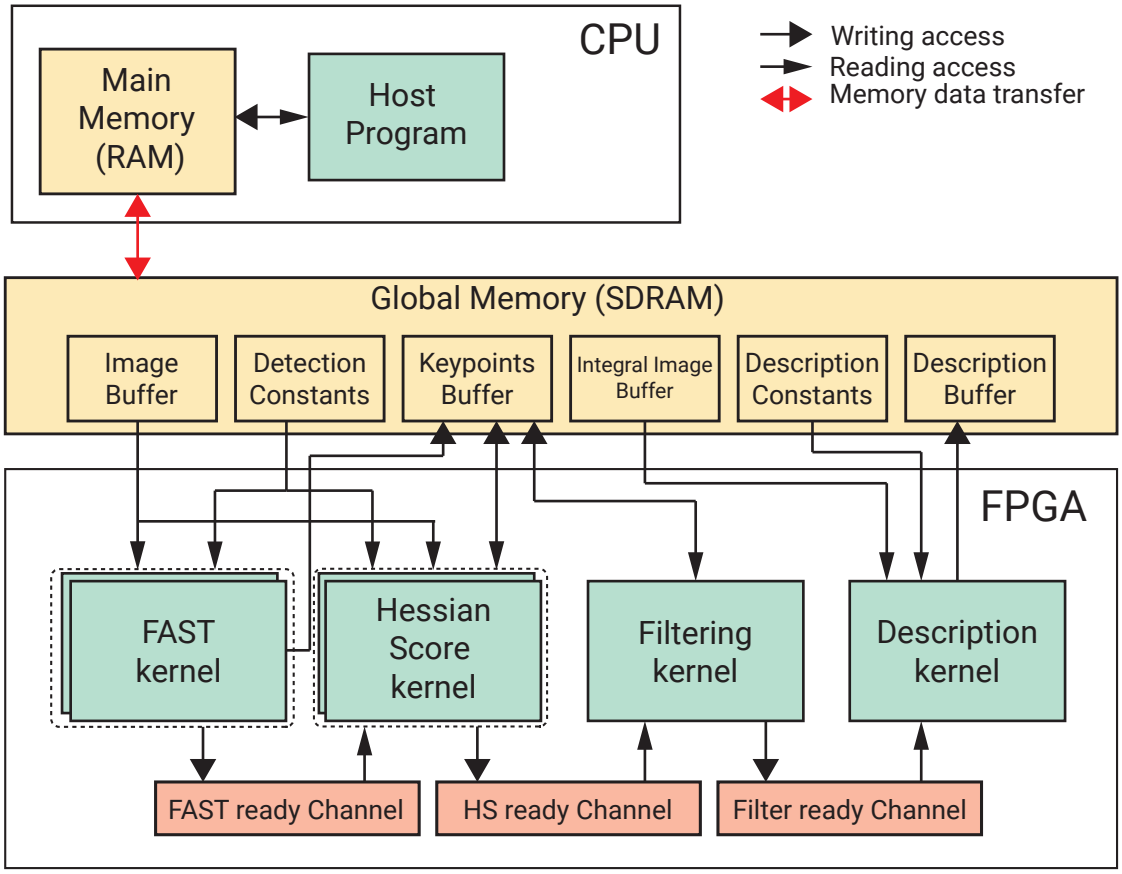

Figure 5.3: HOOFR extractor architecture comprises duplicated FAST kernel, duplicated Hessian score computing kernel, a filtering kernel, and the description kernel. Kernels communicate using the Intel channel extension without interacting with the host processor.

5.3.2 Features Matching

In [8], the matching block was implemented to run on GPU. The algorithm matches the PNF (Previous Neighbor Frame) 's keypoints to the current frame. In the PNF, the cell index and 256-bit description are required for each keypoint. Four matrices are transferred to GPU global memory: Pnf_{Dess}, Pnf_{Cels}, Cur_{Dess} and $Curr_{Distributions}$.

The Pnf_{Dess} is the description of keypoints in previous neighbor frames, and is organized as an unsigned char matrix with a dimension of ($pnf_{np} \times 32$), where pnf_{np} is the total number of keypoints in PNFs. The Pnf_{Cels} is a ($pnf_{np} \times 1$) matrix, where each row represents the cell number where each keypoint is located. For the current frame, Cur_{Dess} is a current description matrix which is created same as Pnf_{Dess} with a dimension of

($cur_{np} \times 32$), where the cur_{np} is the total number of keypoints in the current frame. The keypoints of the current frame are organized by the order of image cell. To keep track of the current keypoints located in each cell, a structure denoted Points_Distribution is used. This structure is composed of two elements: the first element ref is the position where the first keypoint of the cell is located in the whole set, the second element nb is the number of keypoints of the cell. $Curr_{Distributions}$ is a matrix with the dimension of $(N_{CELLS} \times 2)$ for the N cells built using Points_Distribution structure.

For each keypoint in the PNFs, the correspondence is searched in the current frame at the same cell and neighbor cells, as shown in Figure 5.4. This process is so fast since each cell contains a small number of keypoints suitable for handling by one work item. The GPU uses 9 work items in a work group to find the matches in 9 neighbor cells of the current frame. The 9 results are stored in the local memory and are synchronized by the barrier function. After synchronizing, the final matching is extracted by one of the nine work items. Then, the match is validated by computing the Lowe's ratio [173].

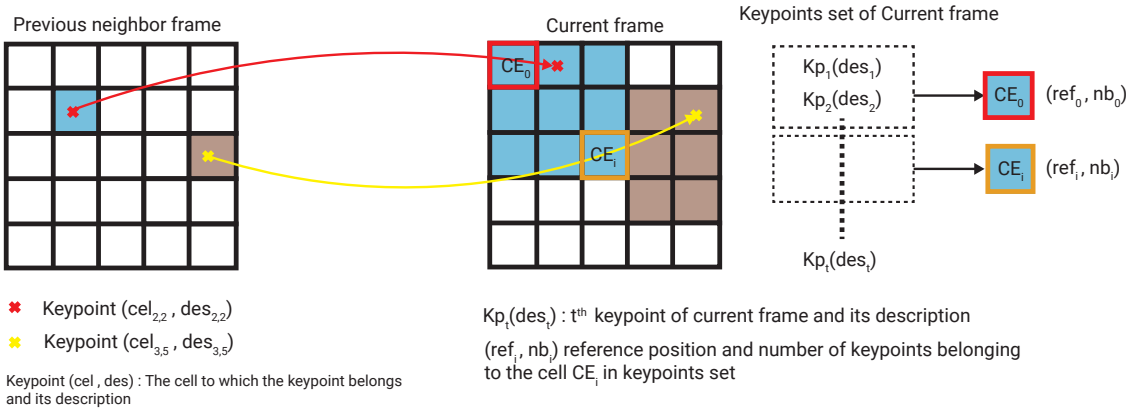

Figure 5.4: Features Matching strategy. The concept consists in searching for the corresponding keypoint of the previous neighbor images in the same cell and the neighboring cells in the current image.

5.4 Algorithm-Architecture mapping

Starting from the Bucketing-based HOOFR extractor [10], we aim to embed the front end of the HOOFR SLAM on FPGA in the context of pushing the processing as close as possible to the sensor. For this purpose, we implemented the matching algorithm on

the FPGA since it is the second most computationally heavy task. The implementation of matching on GPU [8] has shown outstanding performance. However, considering the power consumption, GPUs are highly power-consuming, making thermal management more challenging in embedded systems, especially for applications where energy autonomy is a crucial asset. Although GPUs (i.e., Jetson AGX Xavier™) begin getting close to FPGAs in performance-per-watt, that does not mean they are the best solution for all applications. FPGAs represent an essential feature, which is any-to-any I/O connection, allowing connection to any device, network, or storage without needing a host CPU. The FPGA is well suited for front-end processing in general-purpose processing since it can be directly connected to high-speed sensors and offer very high bandwidth.

In our study, we first evaluated the GPU implementation of matching block on FPGA since the algorithm was implemented in OpenCL [8] and thus can be run on both GPU and FPGA. When we ran the algorithm using NDRange on two images of a resolution of 1280 × 720 with 2000 keypoints, we got a very large runtime of **813.01 ms**. This execution time is explained by the data dependency represented in waiting for the results of the work-items to select the match with the minimum distance. The impact of synchronization is not seen on GPU, as the processing frequency is high and the work-items on GPU are launched simultaneously, unlike in FPGA, where the work-items are launched in parallel in a pipelined way. Also, FPGAs are flexible in terms of programming architecture and are able to provide performance for operations that contain conditionals and/or branches. These architectural differences have a significant impact on performance. For this reason, we used a single-work-item implementation, also called task kernel.

In Table 5.2, we see that with a single task kernel, we obtain a speedup factor of $\times 96$ compared to the NDRange kernel. To reach similar performances as on GPU, we create four matching kernels since they don't take a lot of hardware resources. With four matching kernels, we get a speedup of $\times 254$.

Matching NDRange kernel runtime (ms)	813.01			
Matching task kernel duplication	1	2	3	4
Matching task runtime (ms)	8.39	5.02	3.99	**3.19**

Table 5.2: Comparison of NDRange and single-work-item execution time in milliseconds with different numbers of kernel task duplication.

Figure 5.5 represents the CPU-FPGA mapping for the HOOFR extractor and the matching block. Once the description is done, the results are stored in the global memory, and the four matching blocks are launched concurrently. The description and cell matrices of PNFs are partitioned and distributed over the four kernels, the description of the current frame keypoint and their distributions on cells are accessed by the four kernels as shown in Figure 5.6.

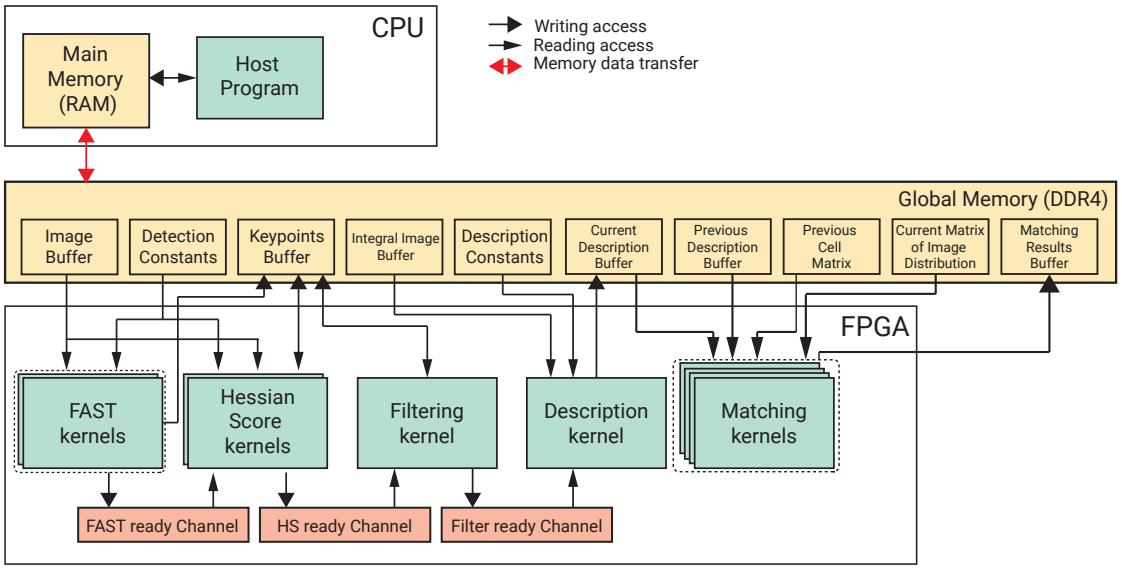

Figure 5.5: CPU-FPGA mapping of the HOOFR extractor and the matching block, the matching block is duplicated four times. The keypoints of the previous neighbor images are distributed on the four kernels.

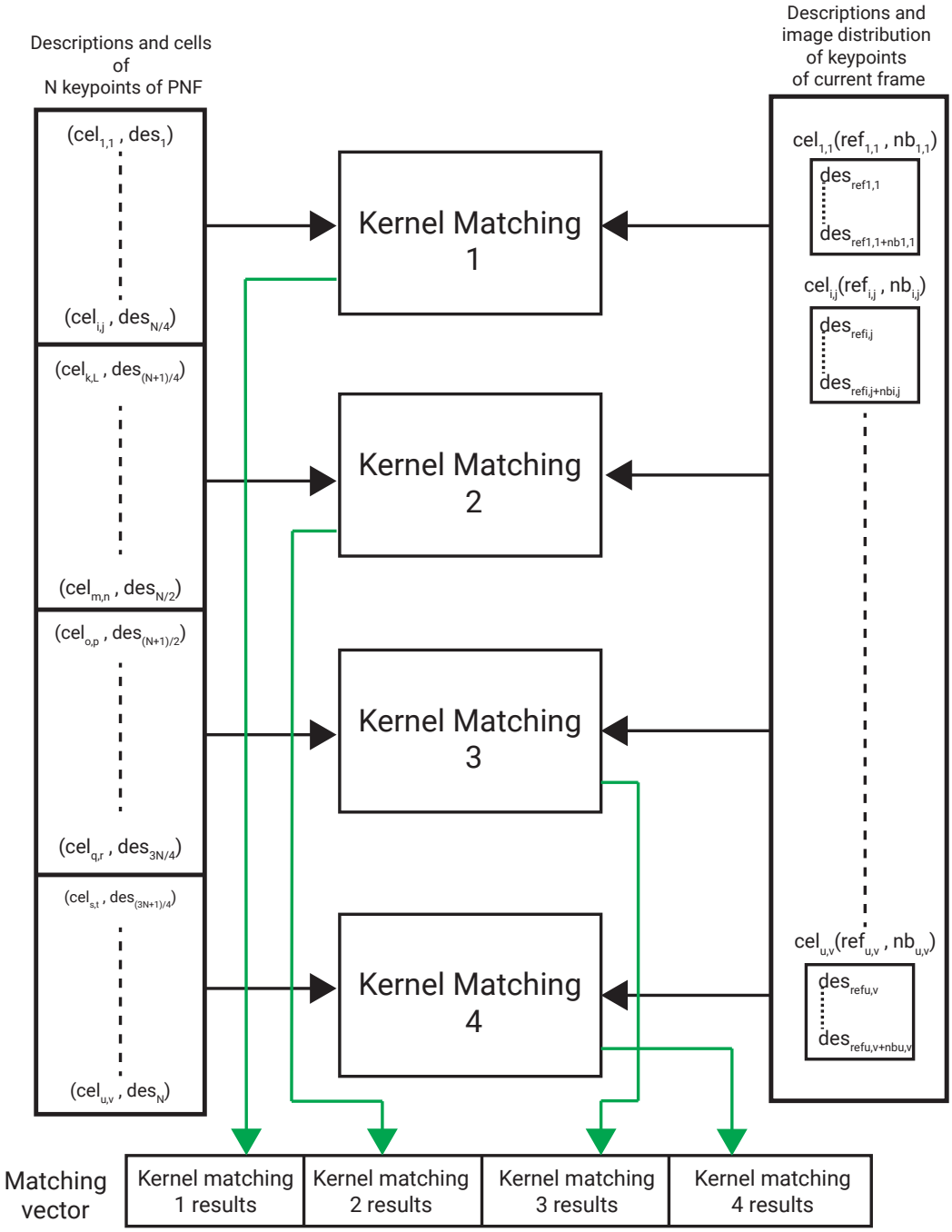

Figure 5.6: Matching kernels duplication diagram. The indexes of the cells and the descriptors of the previous neighboring images are divided into four chunks and distributed over the four kernels. Each kernel finds the match in the current and neighboring cell from the descriptor matrix of the current image. Finally, each kernel stores the found matches into a vector.

Algorithm 5.1 represents one of the four duplicated kernels on FPGA.

Algorithm 5.1 OpenCL Matching Kernel 1/4 on FPGA

```
//OpenCL Matching Kernel on FPGA
declare global arrays: Pnf_Dess, Pnf_Cels, Curr_Distributions, Correspondence, Curr_desc;
function Kernel: Matching
  //Iteration on one quarter of the Pnf keypoints
  For i from 0 to num_ktps/4 do
    //Get the cell coordinates in the grid (NX,NY) to which the Pnf keypoint belongs.
    NX ←Pnf_Cels[i].x;
    NY ←Pnf_Cels[i].y;
    point_pnf_des ←Get_Keypoint_Descriptor(Pnf_Dess[32∗i]);
      //Iteration over the neighbor cells
      //NUM_SEARCH_X, NUM_SEARCH_Y represent the horizontal and vertical search radius respectively.
      For kx from 0 to 2∗NUM_SEARCH_X+1
        For ky from 0 to 2∗NUM_SEARCH_Y+1
          //Compute the neighbor cell coordinates (nx,ny)
          nx ←NX + kx - NUM_SEARCH_X;
          ny ←NY + ky - NUM_SEARCH_Y;
          If (nx,ny) not exceeding Grid boundaries
            //Get the keypoints indexes belonging to the current cell (nx,ny) from Curr_Distributions
            (idx_start,idx_end) = Get_from_Img_Distrib(Curr_Distributions, nx, ny);
             For j from idx_star to idx_end
                point_curr_des ←Get_Keypoint_Descriptor(Curr_desc[32∗j]); //current keypoint descriptor
                Correspondence[i] ←Find_the_best_and_the_second_matches(point_pnf_des, point_curr_des);
```

5.5 Experimental Results

We implement the HOOFR-SLAM front-end on a DE5a-Net DDR4 Arria 10 FPGA PCIe board connected to a desktop via PCIe Gen 3x8. Its CPU is an Intel Xeon Silver 4108 processor with eight cores with a base frequency of 1.8GHz. The PC used for comparison is a high-performance laptop equipped with an 8-core AMD Ryzen 9 CPU with a base frequency of 3 GHz. The frames used for the evaluation are outdoor sequences recorded with the Intel RealSense D455 camera with a frame rate of 30 FPS and a resolution of 1280x720. The images are resized with the resize function of OpenCV.

5.5.1 Resource usage

Our accelerator is implemented on a DE5a-Net DDR4 Arria 10 FPGA Development Kit (operating at 50 MHz) with 1150K LEs. The resource utilization of the proposed architecture is shown in Table 5.3. We followed some performance-improvement rules proposed by Intel® FPGA SDK for OpenCL™ Pro Edition Best Practices Guide to save resources. By reducing the number of unrolled loops and the compute units, we decreased the resource usage and required memory bandwidth. Unrolling the outer loops of a nested-loop structure also increases resource utilization significantly. Therefore, we unrolled only the inner loops. By saving resources, we could incorporate the matching block into our architecture.

Kernel	ALUTs	FFs	RAMs	DSPs
FAST Detection kernel 1	13,155 (2%)	19,048 (1%)	166.2 (6%)	0 (0%)
FAST Detection kernel 2	13,155 (2%)	19,048 (1%)	166.2 (6%)	0 (0%)
Hessian Score kernel 1	7,320 (1%)	12,121 (1%)	62 (2%)	6 (0%)
Hessian Score kernel 2	7,321 (1%)	12,256 (1%)	62 (2%)	6 (0%)
Filtering kernel	5,901 (1%)	11,557 (1%)	75 (3%)	0 (0%)
Description kernel	24,397 (3%)	35,413 (2%)	236 (9%)	8.5 (1%)
Matching kernel 1	13,903 (2%)	19,962 (1%)	190 (7%)	3.5 (0%)
Matching kernel 2	14,282 (2%)	24,427 (1%)	206 (8%)	3.5 (0%)
Matching kernel 3	14,312 (2%)	24,419 (1%)	206 (8%)	3.5 (0%)
Matching kernel 4	14,290 (2%)	24,358 (1%)	206 (8%)	3.5 (0%)
Kernel Subtotal	**128,036 (15%)**	**202,609 (12%)**	**1,575 (59%)**	**34.5 (3%)**
Pipe and channel resources	55 (0%)	558 (0%)	5 (0%)	0 (0%)
Available	**854,400**	**1,708,800**	**2,713**	**1,518**

Table 5.3: FPGA resource utilization. ALUTSs Adaptive Look-Up Tables, FFs Flip Flops, RAMs Random Access Memory blocks, DSPs Digital Signal Processing blocks. Between parenthesis, the usage percentage of the total available.

5.5.2 Matching block timings analysis

The matching block performance has been evaluated on high-performance PC CPU (8-core AMD Ryzen 9 CPU @ 3 GHz) and FPGA architectures for a different number of features. To quantify the performance of each architecture, regardless of the frequency, we have computed the number of cycles per match (CPM) using the following equation:

$$CPM = \frac{t \cdot f}{N} \tag{5.1}$$

Where t is execution time, f is the frequency of the architecture and N is the number of features. We ran the matching block on the PC CPU, and on the DE5a-Net DDR4 Arria 10 FPGA. Our system is evaluated on images taken by an Intel RealSense D455 camera at a resolution of 1280×720. Table 5.4 shows that the FPGA outperforms the PC CPU in CPM and in execution time. This hardware implementation allows a speedup of up to $\times 15$.

Keypoints	PC CPU (CPM)	FPGA (CPM)
1000	49953	**88**
1500	72813	**78**
2000	88925	**108**
2500	110675	**127**

Table 5.4: The number of cycles per match (CPM) of the matching block on PC CPU vs FPGA.

Keypoints	PC CPU (ms)	FPGA (ms)	Speedup
1000	16.65	1.76	**9.46**
1500	36.40	2.33	**15.62**
2000	59.28	4.32	**13.72**
2500	92.22	6.37	**14.47**

Table 5.5: The runtime of the matching block on PC CPU vs FPGA.

5.5.3 Matching block accuracy analysis

In order to ensure that the accuracy of matching is maintained, we fix the FAST detection threshold at 7 and the hamming distance threshold at 20 and we compute the recall (number of correct matches/number of correspondences) and the 1-precision (number of false matches/number of matches) [57, 178, 179]. Both quantities are compared in software and hardware implementation. The results are summarized in the Table 5.6 while the Figure 5.7 shows the matching result on FPGA with images in a resolution of 640×480 pixels. These results show that the matching result on hardware is slightly similar to that on software.

	PC (CPU)		FPGA	
	Recall (%)	1-precision (%)	Recall (%)	1-precision (%)
480×270	64	36	64.94	35.05
640 × 480	58.4	41.6	57	42.98
848×480	56.8	43.2	56.65	43.34
1280×720	57.35	42.65	56.94	43.05

Table 5.6: Comparison of the matching recall-precision between PC and FPGA

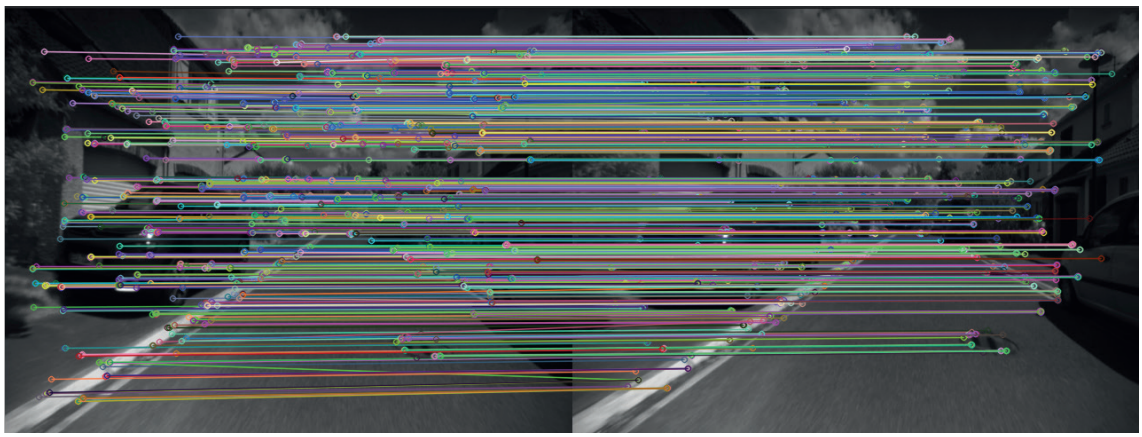

Figure 5.7: Matching two successive images from outdoor sequence 1 on FPGA.

5.5.4 Overall Performance Evaluation

In this section, we evaluate the performance of the entire algorithm on different resolutions supported by the Intel RealSense D455 camera. The image is divided into a grid. NX and NY represent the number of horizontal and vertical cells. Each cell return a maximum number of keypoints (with the highest Hessian score) limited to POINTS_PER_CELL. As shown in Table 5.7, our design reaches a frequency of **30** FPS at a resolution of **1280**×**720** pixels, generating and matching **2009** keypoints which is well suited to achieve good localization accuracy, as we have seen in the previous chapter. At **640** × **480** pixels resolution , our design reaches **38** FPS, which is sufficient to perform on-the-fly processing. The HD resolution can be helpful if an object detection task is needed since object detection at more considerable distances becomes feasible [124].

Resolution (px)	Number of Keypoints	NX×NY	Runtime (ms)	FPS
480×270	2228	24×14	24.46	40.88
640×480	2158	24×14	26.21	38.15
848×480	2111	24×14	27.09	36.91
1280×720	**2009**	**24×14**	**33**	**30.3**

Table 5.7: Timing performance on different resolutions (FAST_threshold = 7, POINTS_PER_CELL = 15)

5.6 Conclusion

This chapter presents a design of FPGA-based near-sensor processing (matching incorporated with the Bucketing-based HOOFR extractor). A matching algorithm has been adapted to FPGA programming paradigms to achieve real-time processing tailored to the Intel RealSense camera frame rate. Also, we have shown that the algorithm's complexity has been preserved to guarantee the same quality of results in software as in hardware. Our design was implemented on DE5a-Net DDR4 Arria 10 FPGA architecture. The FPGA implementation shows that the OpenCL design is 9x to 14x faster than the C++ implementation running on a high-performance CPU. The achieved throughput is 30 FPS at 1280×720 pixels resolution and 38 FPS at 640×480 pixels resolution, which is sufficient to perform on-the-fly processing. The proposed architecture, as shown in the Figure 5.8, involves acquiring the images from the sensor on the CPU, launching the keypoint extraction and matching on FPGA, and finally launching the pose estimation on the CPU. The loop closure detection is launched on a parallel thread on the CPU.

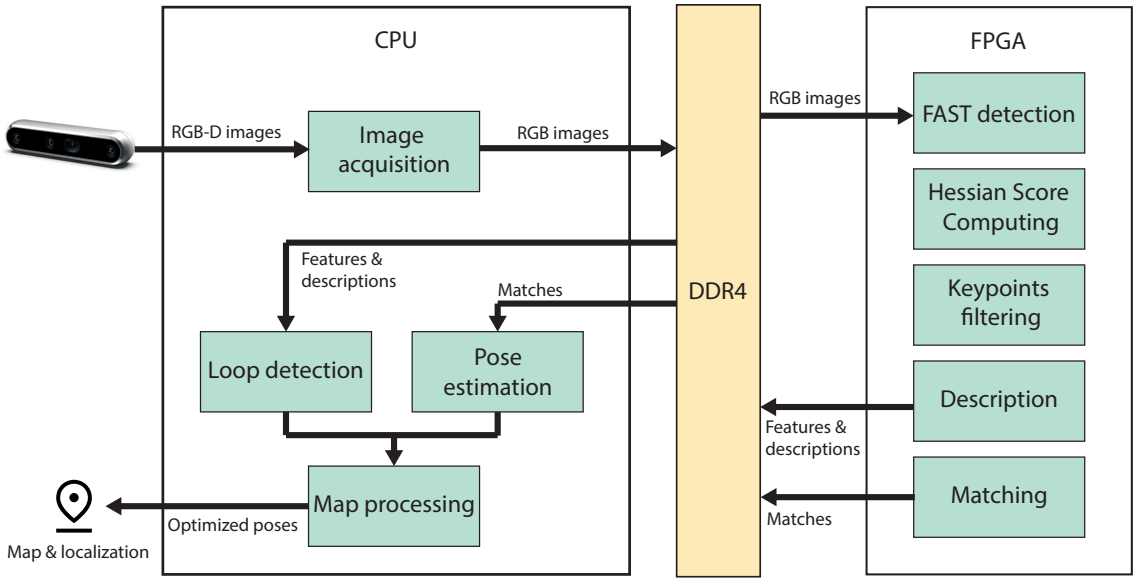

Figure 5.8: HOOFR SLAM functional block partitioning on a heterogeneous CPU-FPGA architecture

Conclusion and Future Works

Conclusion

In this thesis, we adopted a methodology considering the processing chain of a SLAM system (from the sensor to the embedded architecture). The focus of our study was on the front-end part of SLAM algorithms. We have started by investigating the characteristics of an RGB-D sensor and its impact on localization accuracy. In this context, we proposed a sensor-algorithm coupling methodology, which consists of identifying the parameters correlated to the sensors or significantly impacting the localization (i.e. camera pose). Then, the optimization protocol is applied to these parameters to determine their optimal values.

After the characterization of the RGB-D sensor, we proceeded to the extension of the HOOFR-SLAM towards using RGB-D sensors for autonomous vehicle applications. To ensure the consistency and robustness of the algorithm in dynamic environments, we have implemented a keypoint filtering mechanism based on depth maps. This filter allows us to efficiently enhance the localization's accuracy by keeping only reliable keypoints.

Afterward, algorithmic optimizations were introduced to the algorithm. When analyzing the data flow, we noticed that the amount of input data highly drives processing time. As more keypoints are processed, the complexity grows in several functional blocks (e.g. description, matching, pose estimation). Unlike other works [165, 155, 167], which use keypoint filtering only to improve the algorithm's accuracy in dynamic environments, our implementation serves two goals. First, improving the accuracy, and second, reducing the amount of data fed to functional blocks, thus reducing the processing time. For this

purpose, the keypoint filtering was performed directly after features detection, reducing the number of points to be described, matched, and used for the pose estimation.

On the other hand, in pose estimation, the RANSAC algorithm processes all matches in the same way and draws random samples uniformly from the complete set. At low proportions of inliers, the number of iterations increases exponentially. This problem was addressed using the PROSAC algorithm, which takes quality-ordered matches in input, prioritizing these matches when sampling under the assumption that these matches have a high probability of being inliers, which leads to faster convergence. When comparing the performance between RANSAC and PROSAC, our algorithm achieves x5 of speedup while maintaining the same accuracy. Our SLAM system can run at 27 FPS on a 3 GHz 8-core CPU and 19 FPS on a 2.26GHz 8-core embedded processor without accelerating processing on their associated GPUs. This processing rate generally allows a good localization accuracy with an approximate error bound of 1m in an urban environment for a vehicle speed of up to 40Km/h. However, for on-the-fly processing, the algorithm must be able to run at the same rate as the sensor or even more (from 30 FPS onwards).

For this purpose, we boost the performance of our algorithm and take advantage of FPGA-based architectures known for their low power consumption. A study was conducted on optimizing the partitioning of the different functional blocks on the computing units. This partitioning considers the data flow, the functional blocks' dependency, and workloads. An FPGA architecture based on an OpenCL implementation for HOOFR feature matching has been designed. The complexity of the matching algorithm has been respected to guarantee the same matching performance in software as in hardware implementations. This matching system has been incorporated with the bucketing-based HOOFR extractor to embed 88% of the front-end on FPGA. The FPGA implementation shows that the OpenCL design is up to 14x faster than the CPU-based implementation. The achieved throughput is 30 FPS at 1280×720 pixels resolution and 38 FPS at 640×480 pixels resolution. The results demonstrated the capacity of FPGA architecture intending to handle an on-the-fly processing of the HOOFR SLAM front-end. However, for implementation on a System on Chip (SoC), we are limited by the amount of data to process. For the description of the features, we need 500 MB to store the description of 2000 keypoints, which limits us to using off-chip memories, making part of the performance dependent on

the memory bandwidth. The obtained results and the defined architecture model demonstrated the possibility of performing complex algorithms such as SLAM on heterogeneous CPU-FPGA architectures-based embedded systems. The technological evolution allows the design of CPU-FPGA architectures. These architectures will also make it possible to bring the data processing as close as possible to the sensor and thus design intelligent Visual SLAM systems.

Future works

The work presented in this thesis presents multiple contributions toward embedded SLAM for autonomous vehicles. Several leads are considered to extend this work on different levels (sensor, algorithm, and architecture). At the sensor level, we intend to extend our RGB-D dataset with more challenging scenarios under a wide range of conditions (including weather and illumination changes) for a more in-depth study. The datasets studied were acquired at moderate speeds (20Km/h to 40Km/h), allowing a good performance at 30FPS. Scenarios at higher speeds on highways are planned to complete sensor characterization to maintain or improve accuracy. In addition, datasets with various elevations are intended. As for now, the algorithm can operate on flat ground. The integration of inertial measurement units (IMU) will be an asset to ensure the algorithm's consistency in an environment with altitude variations. From an architecture point of view, pose computing will be a subject of study for a fully embedded front-end system on FPGA. Finally, the FPGA and camera interfacing should be studied thoroughly to realize the on-the-fly processing, designing hence a smart visual SLAM sensor. This latter provides a high-performance processor for back-end processing and an FPGA handling the processing and interpretation of data from the visual sensors, an architecture model can be proposed as shown in Figure 5.9. This architecture consists of interfacing the camera to the FPGA memory. This allows the image pixels to directly flow to the FPGA. The on-chip architecture handles the pre-processing operations offloaded from the CPU, thus improving the overall system throughput, and resolves the bottleneck between the camera and the CPU.

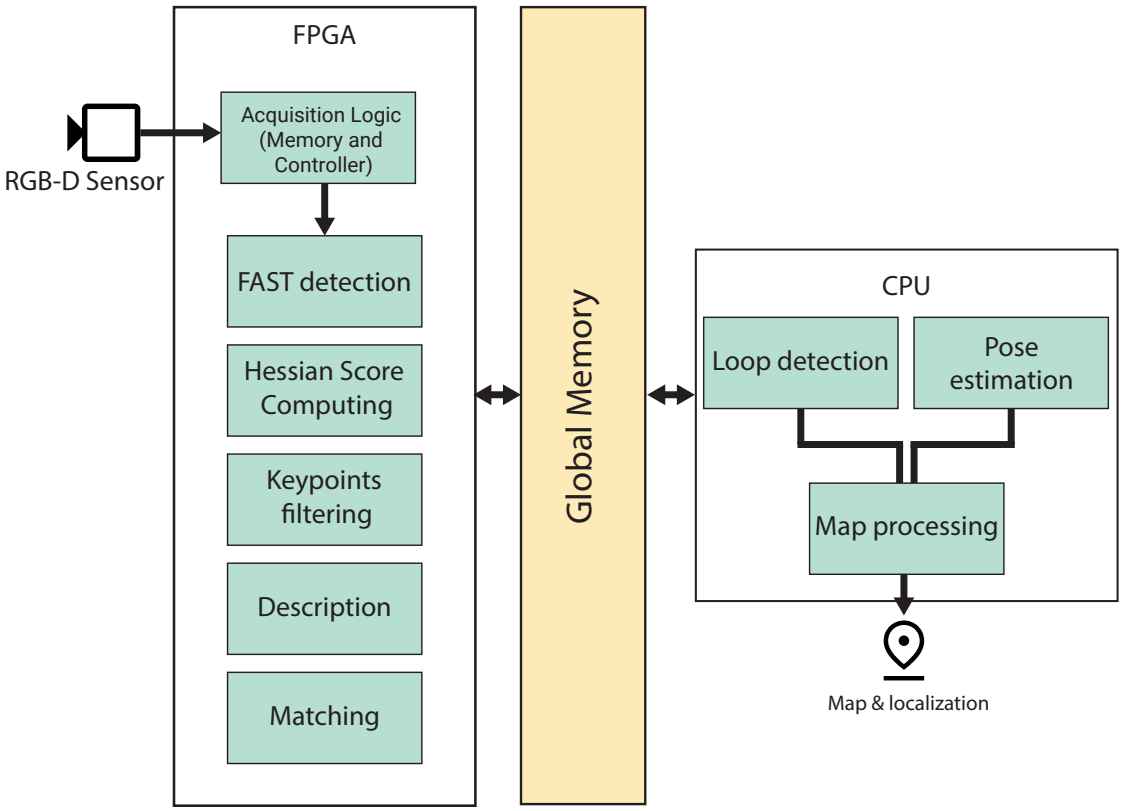

Figure 5.9: Online FPGA SLAM Preprocessing

Appendix

Résumé de la thèse

Introduction

Le domaine des véhicules autonomes est actuellement une tendance dans de nombreux travaux de recherche. Plusieurs approches ont été adoptées pour résoudre le problème du SLAM pour les véhicules autonomes. Le système mondial de navigation par satellite (GNSS) est une technologie couramment utilisée pour la localisation. Néanmoins, ce système a été considéré comme limité en raison de la dégradation du signal dans les zones urbaines denses et des scénarios avec effets d'ombre. Le système avancé d'aide à la conduite (ADAS) est une méthode alternative qui a été évaluée et qui avait pour but d'aider à la localisation du véhicule. Cependant, la disponibilité de toutes les informations routières, telles que le marquage des voies et les bords de route, n'est pas assurée sur toutes les routes, ce qui rend cette méthode inefficace. D'autre part, les systèmes cellulaires 4G/5G, la bande ultra large (UWB), le réseau local sans fil (WLAN), le réseau de capteurs sans fil (WSN) et Bluetooth restent limités en termes de coût, de précision, de sécurité, de complexité et d'évolutivité. Par conséquent, les méthodologies SLAM basées sur la vision, appelées SLAM visuel, sont devenues le courant principal de la recherche actuelle. Les algorithmes de SLAM visuel doivent être plus robustes pour faire face aux paramètres complexes et dynamiques de l'environnement urbain. Contrairement aux robots mobiles sur lesquels la plupart des méthodes SLAM développées ont été évaluées, les véhicules autonomes ont des paramètres plus difficiles à prendre en compte si la conduite autonome est souhaitée. Ces défis incluent la taille de l'environnement, la

fermeture de la boucle et l'association de données, qui font partie des paramètres problématiques qui apparaissent dans des environnements plus dynamiques, tels que ceux que l'on trouve dans les espaces urbains [26]. À cet effet, le processus de conception doit prendre en compte plusieurs paramètres en partant des capteurs jusqu'à l'architecture embarquée. De plus, l'émergence d'architectures embarquées hétérogènes de faible puissance offre une grande opportunité d'explorer le potentiel de pousser le traitement plus près du capteur et d'assurer un traitement à la volée.

Couplage des paramètres capteur-algorithme

Le SLAM est une fonctionnalité de perception cruciale dans une variété d'applications, notamment les robots et les véhicules autonomes. Les caméras RGB-D font partie des capteurs généralement utilisés par les systèmes SLAM récents. De nombreux algorithmes SLAM RGB-D ont été explorés et évalués à l'aide d'ensembles de données accessibles au public sans tenir compte des spécifications du capteur ou des modes de capture d'image qui pourraient augmenter ou réduire la précision de la localisation. Dans ce chapitre, nous abordons la localisation en intérieur en tenant compte des spécifications des capteurs. Dans ce contexte, nous soulignons l'impact des modalités d'acquisition des capteurs sur la précision de la localisation et nous proposons une stratégie d'optimisation paramétrique pour améliorer la précision de la localisation dans un environnement donné. Ce protocole est utilisé pour améliorer un paramètre de l'algorithme SLAM lié à la profondeur. Notre propre jeu de données d'intérieur disponible publiquement a servi de base à cette analyse. Notre protocole d'optimisation paramétrique du couplage capteur-algorithme consiste à évaluer l'algorithme sur différentes séquences avec différents modes d'acquisition. Ensuite, le mode d'acquisition présentant l'erreur la plus faible est utilisé pour ajuster globalement les paramètres de l'algorithme. Nous nous concentrons sur l'optimisation de l'algorithme ORB-SLAM2. ORB-SLAM2 possède trois paramètres principaux liés à l'entrée de l'algorithme, à savoir : le nombre des primitives, le seuil du détecteur FAST et le seuil de profondeur. Le nombre des primitives repose sur le détecteur FAST, qui dépend de l'exposition [149]. Ces deux paramètres ne sont pas liés à la caméra de profondeur. Nous avons identifié un paramètre physiquement corrélé au capteur : le seuil de

profondeur [148]. Ce paramètre permet à l'algorithme de classer les primitives proches et lointaines. Ce paramètre est étroitement corrélé à la distribution d'erreur du mode d'acquisition RGB-D. Nous avons évalué la valeur de ce paramètre en fonction de la distribution d'erreur. Nous avons évalué la valeur de ce paramètre sur une plage bien définie. Les différents tests ont été réalisés sur une station de calcul, équipée d'un processeur Intel Xeon W-2265 de 24 cœurs tournant à 3,5 GHz, de 64 Go de RAM et d'une carte graphique NVIDIA Quadro RTX 6000 avec 4608 cœurs CUDA. Nous avons calculé l'erreur de trajectoire euclidienne et le nombre de points suivis sur la carte pour chaque image. Ce protocole permet de classer les conditions d'entrée (nombre de points suivis sur la carte vus par l'image courante) et de sortie (distance euclidienne entre une pose courante et celle de la trajectoire référencée) de l'algorithme afin de qualifier ses performances. Inspiré de [150, 151], nous avons proposé une optimisation paramétrique basée sur la matrice de confusion suivante. Nous avons évalué les algorithmes sur diverses trajectoires basées sur les différents modes d'acquisition de la caméra. La comparaison entre le SLAM IR-D passif et le SLAM RGB-D passif a permis de déduire que le paramètre de seuil de profondeur pour ORB-SLAM2 ne suit pas une tendance spécifique. Une méthode basée sur la courbe ROC a été établie pour trouver une valeur de seuil de profondeur optimale pour le capteur IR. L'utilisation d'une caméra IR par rapport à la caméra RGB a permis de réduire l'erreur ATE de 23,08% pour RTAB-Map dans Digiteo_seq1 et de 82,14% pour ORB-SLAM2 optimisé dans Digiteo_seq3. L'utilisation de la caméra IR du D435 offre un avantage significatif car elle possède un champ de vision plus large pour le suivi des primitives dans les angles morts, et le fait que les cartes de profondeur soient également alignées avec la caméra IR gauche signifie qu'aucun traitement d'alignement supplémentaire n'est nécessaire. Sur la base des paramètres trouvés dans la comparaison du SLAM IR-D et du SLAM RGB-D passif, l'algorithme SLAM IR-D a été comparé au SLAM Stereo Vision, et nous avons trouvé une diminution de l'erreur de translation de 78,26% en utilisant les données IR-D pour ORB-SLAM2 dans Digiteo_seq3 et de 28,57% pour RTAB-Map dans Digiteo_seq1. Enfin, nous avons comparé les modes actif et passif. Nous avons déduit que le mode actif donne une carte de profondeur plus dense ; par conséquent, nous obtenons des résultats plus précis puisque davantage de primitives sont utilisées pour calculer la translation, la rotation et la mise à

l'échelle. La conception d'un système SLAM basé sur RGB-D doit prévoir un couplage fort des paramètres de l'algorithme du capteur, en particulier ceux liés au champ de vision, au seuil de profondeur et au projecteur IR. La prise en compte de la caractérisation du capteur peut augmenter la précision de la localisation pour les applications robotiques dans les environnements intérieurs.

RGB-D HOOFR-SLAM

La localisation et la cartographie simultanées RGB-D (SLAM) sont devenues de plus en plus populaires en raison de leur faible coût et des avantages de la caméra RGB-D. Plusieurs efforts ont été déployés pour développer le SLAM RGB-D. Malheureusement, ces travaux n'ont pas été évalués et étendus aux applications de véhicules extérieurs. Dans ce chapitre, nous présentons une extension de HOOFR-SLAM à une modalité RGB-D améliorée appliquée à un véhicule autonome dans un environnement extérieur dynamique. Nous proposons une méthode de filtrage des primitives basée sur les cartes de profondeur pour améliorer les performances de l'algorithme dans les environnements dynamiques. De plus, des optimisations algorithmiques ont été effectuées pour améliorer les performances. Dans le chapitre précédent, nous avons vu comment le couplage capteur-algorithme est essentiel pour améliorer la précision de la localisation dans les environnements intérieurs. Dans ce chapitre, nous nous appuyons sur ce protocole d'optimisation pour améliorer HOOFR-SLAM dans les environnements extérieurs. Enfin, nous utilisons une approche hardware-in-the-loop (HIL) pour valider l'algorithme sur une architecture embarquée et un jeu de données collecté par un véhicule instrumenté du laboratoire. HOOFR-SLAM est un algorithme de SLAM visuel basé sur l'extracteur bio-inspiré Hessian ORB - Overlapped FREAK (HOOFR). Cet extracteur, composé du détecteur ORB avec le score hessien et du descripteur FREAK bio-inspiré avec un chevauchement amélioré, présente une fiabilité et un temps d'exécution améliorés pendant la mise en correspondance. Les résultats de l'HOOFR ont montré une performance compétitive avec SURF et SIFT avec une vitesse plus rapide et un faible coût de calcul comme ORB, mais dépassant ce dernier en performance [57]. Le Stereo HOOFR-SLAM se compose de deux blocs principaux. Le premier bloc est consacré au traitement des données du capteur et à

l'estimation de l'ego-motion, appelé front-end. Le second bloc représente le noyau SLAM et se compose de l'optimisation du graphe de pose et des tâches de fermeture de boucle. L'algorithme reçoit en entrée une image stéréo. L'image de gauche est utilisée pour estimer le mouvement relatif de la caméra. L'image de droite est utilisée pour calculer l'échelle en utilisant la triangulation stéréo. Pour chaque image, l'extracteur HOOFR est appliqué pour détecter et décrire les primitives utilisées pour l'estimation de la pose et la détection des boucles. La mise en correspondance stéréo fournit l'échelle réelle en calculant le rapport entre la distance réelle des points de repère et leurs distances triangulées. Dans le processus de mise en correspondance, les primitives sont mises en correspondance avec celles des images précédentes de gauche. Chaque image précédente avec une estimation de transformation réussie est appelée image voisine précédente (PNF). La translation, la rotation et les positions des points de repère sont extraites de la matrice essentielle en utilisant la triangulation. En raison de son coût de traitement élevé, l'ajustement du faisceau est remplacé par un filtrage fenêtré pour estimer la position actuelle de la caméra à partir d'un ensemble de PNF. Chaque pose prédite des PNF est associée à un poids de confiance, ainsi la pose optimale est la moyenne de toutes les prédictions par leurs poids respectifs [8]. Le thread de fermeture de boucle est exécuté en parallèle avec le thread de mapping. Chaque image gauche est interrogée dans l'ensemble des images clés pour trouver la vraisemblance maximale. L'image actuelle est considérée comme une nouvelle image clé dans le cas d'un score de correspondance faible et est ajoutée au graphe de pose. Une fermeture de boucle potentielle n'est détectée que lorsque la pose de l'image clé ayant un score de correspondance élevé est éloignée de l'image actuelle dans le graphe de pose. La fermeture de la boucle est alors validée par le calcul de la transformation relative entre l'image courante et l'image clé correspondante [8]. Dans le HOOFR-SLAM stéréo, le processus commence par la détection et la description des primitives, suivies de la mise en correspondance stéréo. Ensuite, le processus de mise en correspondance temporelle est lancé. Dans la méthode RGB-D HOOFR-SLAM, nous proposons un extracteur HOOFR modifié basé sur le filtrage des primitives à l'aide des cartes de profondeur fournies par le capteur RGB-D. Cette approche améliore la précision de la localisation en éliminant les primitives dont la profondeur n'est pas fiable, notamment les primitives détectées sur

des véhicules en mouvement, les nuages et celles dont les valeurs de profondeur sont invalides (zéro). De cette façon, seuls les points clés pertinents sont conservés et le nombre de points clés à décrire et à faire correspondre est réduit, ce qui diminue le temps de traitement. Après le filtrage, la description et la mise en correspondance, vient l'étape du calcul de la pose. Dans la version Stereo HOOFR SLAM, la matrice essentielle a été estimée en appliquant le schéma RANSAC avec une erreur de mesure (me) inférieure au pixel afin d'obtenir un modèle optimisé sans avoir recours à des méthodes d'optimisation comme le BA. Nguyen et al. ont trouvé, au cours d'expériences, qu'une erreur de mesure de l'inlier dans le schéma RANSAC inférieure à 0.4 permet une précision de localisation élevée. Cependant, lorsqu'on applique un me=0.4, le temps d'exécution augmente considérablement. Pour résoudre ce problème, Nguyen et al. ont proposé une solution consistant à estimer la matrice essentielle deux fois, la première fois avec me=1 et la seconde fois en utilisant les inliers trouvés précédemment avec un me=0,4. Dans l'algorithme RGB-D HOOFR SLAM, puisque le nombre de points d'intérêt est réduit dans l'étape de détection, nous estimons la matrice essentielle une seule fois pour une petite valeur de me. Dans le pire des cas, avec peu de bons points, RANSAC prendra beaucoup de temps pour trouver la solution. Par conséquent, nous avons changé la méthode RANSAC en PROSAC (PROgressive Sample Consensus). Cette méthode est basée sur un échantillonnage progressif des points en commençant par ceux qui sont les mieux classés en fonction de leur facteur de qualité, ce qui permet d'économiser considérablement le temps de calcul. Dans [172], les auteurs ont démontré que PROSAC était plus de cent fois plus rapide que RANSAC, et dans le pire des cas, ils ont un comportement identique. Nous avons utilisé le test du ratio de Lowe [173] pour trier les correspondances. Nous avons comparé le SLAM RGB-D HOOFR à sa version stéréo. En outre, le RGB-D HOOFR SLAM a été comparé à d'autres algorithmes de pointe en ce qui concerne ses performances et s'est avéré être un bon compromis entre le temps d'exécution et la précision de localisation. Notre algorithme peut fonctionner en temps réel à 27 FPS sur un CPU de PC et à 19 FPS sur un processeur embarqué sans accélérer le traitement sur le GPU.

Co-conception matériel-logiciel : Vers un traitement front-end basé sur une architecture FPGA

Les systèmes SLAM basés sur les primitives deviennent de plus en plus populaires en raison de leurs performances et de leur robustesse. Plusieurs extracteurs de caractéristiques sont utilisés dans divers systèmes SLAM tels que ORB [60], SIFT [173], et SURF [58]. Bien que ces extracteurs offrent de bons résultats de correspondance, la complexité de calcul de l'extraction et de la mise en correspondance des primitives représente un défi important lors de l'intégration de tels algorithmes SLAM dans des architectures embarqués. Récemment, Nguyen et al. [10] ont proposé une implémentation FPGA de l'extracteur HOOFR tout en conservant la même précision. Cependant, la tâche d'appariement reste la plus longue dans le flux de traitement. La conception d'une architecture accélérée pour ce bloc fonctionnel est obligatoire pour réaliser un traitement à la volée sur un système sur puce. Notre défi consiste à améliorer les performances de l'algorithme sur des architectures à faible consommation afin de garantir le traitement à la volée. Les FPGA sont considérés comme le meilleur choix pour le traitement de flux. Contrairement aux GPU, qui ne fournissent un parallélisme que pour le traitement et l'accélération des données, les FPGA peuvent fournir un parallélisme des données, des tâches et des pipelines, ce qui les rend plus adaptés au traitement par flux: [174, 175] notamment pour les systèmes embarqués. Nous atteignons notre objectif grâce à une adéquation algorithme-architecture appliquée aux architectures CPU-FPGA. En pratique, un algorithme est décomposé en blocs fonctionnels, et chaque bloc est affecté à l'unité de traitement appropriée, ce qui garantit des performances optimales. Dans ce chapitre, nous évaluons les performances du bloc de correspondance sur différentes architectures, puisqu'il s'agit du goulot d'étranglement des performances. Nous concluons en proposant une répartition optimale CPU-FPGA pour le front-end RGB-D HOOFR-SLAM. En partant de l'extracteur HOOFR basé sur le Bucketing: [10], nous visons à embarquer le front-end du HOOFR SLAM sur FPGA dans le contexte de pousser le traitement aussi près que possible du capteur. À cette fin, nous avons implémenté l'algorithme d'appariement sur le FPGA car il s'agit de la deuxième tâche la plus lourde en termes de calcul. L'implémentation de l'appariement sur GPU : [8] a montré des performances remarquables. Cependant, si l'on considère la consommation

d'énergie, les GPU sont très gourmands en énergie, ce qui rend la gestion thermique plus difficile dans les systèmes embarqués, en particulier pour les applications où l'autonomie énergétique est un atout crucial. Bien que les GPU (c'est-à-dire les Jetson AGX Xavier™) commencent à se rapprocher des FPGA en termes de performance par watt, cela ne signifie pas qu'ils constituent la meilleure solution pour toutes les applications. Les FPGA représentent une caractéristique essentielle, qui est la connexion d'E/S any-to-any, permettant la connexion à n'importe quel dispositif, réseau ou stockage sans avoir besoin d'un CPU hôte. Le FPGA est bien adapté au traitement frontal dans le traitement polyvalent, car il peut être directement connecté à des capteurs à haute vitesse et offrir une bande passante très élevée. Dans notre étude, nous avons d'abord évalué l'implémentation GPU du bloc de correspondance sur FPGA puisque l'algorithme a été implémenté en OpenCL: [8] et peut donc être exécuté à la fois sur GPU et FPGA. Lorsque nous avons exécuté l'algorithme en utilisant NDRange sur deux images d'une résolution de 1280×720 avec 2000 points clés, nous avons obtenu un temps d'exécution très important de 813.01ms. Ce temps d'exécution s'explique par la dépendance des données représentée par l'attente des résultats des work-items pour sélectionner la correspondance avec la distance minimale. L'impact de la synchronisation n'est pas visible sur GPU, car la fréquence de traitement est élevée et les work-items sur GPU sont lancés simultanément, contrairement à FPGA, où les work-items sont lancés en parallèle de manière pipelinée. De plus, les FPGA sont flexibles en termes d'architecture de programmation et sont capables de fournir des performances pour les opérations qui contiennent des opérations conditionnelles et/ou des branches. Ces différences architecturales ont un impact significatif sur les performances. Pour cette raison, nous avons utilisé une implémentation à élément de travail unique, également appelée noyau de tâche.

Dans l'évaluation on a trouvé qu'avec un noyau à tâche unique, nous obtenons un facteur d'accélération de $\times 96$ par rapport au noyau NDRange. Pour atteindre des performances similaires à celles du GPU, nous créons quatre noyaux correspondants car ils ne nécessitent pas beaucoup de ressources matérielles. Avec quatre noyaux correspondants, nous obtenons un gain de vitesse de $\times 254$. L'algorithme de mise en correspondance a été adapté aux paradigmes de programmation des FPGA afin de réaliser un traitement en temps réel adapté à la fréquence d'images de la caméra Intel RealSense. Nous avons

également montré que la complexité de l'algorithme a été préservée pour garantir la même qualité de résultats en logiciel qu'en matériel. Notre conception a été implémentée sur l'architecture FPGA DE5a-Net DDR4 Arria 10. L'implémentation FPGA montre que la conception OpenCL est $\times 9$ à $\times 14$ plus rapide que l'implémentation C++ fonctionnant sur un CPU haute performance. Le débit obtenu est de 30 FPS à une résolution de 1280×720 pixels et de 38 FPS à une résolution de 640×480 pixels, ce qui est suffisant pour effectuer un traitement à la volée. L'architecture proposée, implique l'acquisition des images du capteur sur le CPU, le lancement de l'extraction et de la mise en correspondance des points clés sur le FPGA, et enfin le lancement de l'estimation de la pose sur le CPU. La détection de fermeture de boucle est lancée sur un thread parallèle sur le CPU.

Conclusion et perspectives

Dans cette thèse, nous avons adopté une méthodologie considérant la chaîne de traitement d'un système SLAM (du capteur à l'architecture embarquée). Notre étude s'est concentrée sur la partie frontale des algorithmes SLAM. Nous avons commencé par étudier les caractéristiques d'un capteur RGB-D et son impact sur la précision de la localisation. Dans ce contexte, nous avons proposé une méthodologie de couplage capteur-algorithme, qui consiste à identifier les paramètres corrélés aux capteurs ou ayant un impact significatif sur la localisation (i.e. la pose de la caméra). Ensuite, le protocole d'optimisation est appliqué à ces paramètres pour déterminer leurs valeurs optimales. Après la caractérisation du capteur RGB-D, nous avons procédé à l'extension du HOOFR-SLAM vers l'utilisation de capteurs RGB-D pour des applications de véhicules autonomes. Pour assurer la cohérence et la robustesse de l'algorithme dans des environnements dynamiques, nous avons implémenté un mécanisme de filtrage des points clés basé sur des cartes de profondeur. Ce filtre nous permet d'améliorer efficacement la précision de la localisation en ne conservant que les points clés fiables. Ensuite, des optimisations algorithmiques ont été introduites dans l'algorithme. En analysant le flux de données, nous avons remarqué que la quantité de données d'entrée influence fortement le temps de traitement. Plus le nombre de points clés traités est important, plus la complexité augmente dans plusieurs blocs fonctionnels (par exemple, la description, la correspondance, l'estimation

de la pose). Contrairement à d'autres travaux : [165, 155, 167], qui utilisent le filtrage des points clés uniquement pour améliorer la précision de l'algorithme dans des environnements dynamiques, notre implémentation sert deux objectifs. Premièrement, améliorer la précision, et deuxièmement, réduire la quantité de données fournies aux blocs fonctionnels, réduisant ainsi le temps de traitement. Dans ce but, le filtrage des points clés a été effectué directement après la détection des caractéristiques, réduisant ainsi le nombre de points à décrire, à faire correspondre et à utiliser pour l'estimation de la pose. D'autre part, dans l'estimation de la pose, l'algorithme RANSAC traite toutes les correspondances de la même manière et tire des échantillons aléatoires uniformément de l'ensemble complet. Lorsque la proportion d'observations aberrantes est faible, le nombre d'itérations augmente de manière exponentielle. Ce problème a été résolu à l'aide de l'algorithme PROSAC, qui prend en entrée des correspondances ordonnées par qualité et leur donne la priorité lors de l'échantillonnage, en partant du principe que ces correspondances ont une probabilité élevée d'être des valeurs aberrantes, ce qui conduit à une convergence plus rapide. En comparant les performances de RANSAC et de PROSAC, notre algorithme atteint une accélération de $\times 5$ tout en conservant la même précision. Notre système SLAM peut fonctionner à 27 FPS sur un CPU 8-core à 3 GHz et à 19 FPS sur un processeur embarqué 8-core à 2.26GHz sans accélérer le traitement sur leurs GPUs associés. Cependant, pour un traitement à la volée, l'algorithme doit pouvoir fonctionner à la même vitesse que le capteur, voire plus (à partir de 30 FPS). Pour cela, nous avons amélioré les performances de notre algorithme en profitant des architectures à base de FPGA connues pour leur faible consommation énergetique. Une étude a été menée sur l'optimisation du partitionnement des différents blocs fonctionnels sur les unités de calcul. Ce partitionnement prend en compte le flux de données, la dépendance des blocs fonctionnels et les charges de travail. Une architecture FPGA basée sur une implémentation OpenCL pour la mise en correspondance des primitives HOOFR a été conçue. La complexité de l'algorithme d'appariement a été respectée pour garantir les mêmes performances d'appariement dans le logiciel que dans les implémentations matérielles. Ce système de mise en correspondance a été incorporé à l'extracteur HOOFR basé sur le "bucketing" afin d'intégrer 88% du frontal sur FPGA. L'implémentation FPGA montre que la conception OpenCL est jusqu'à 14 fois plus rapide que l'implémentation basée sur le CPU. Le débit obtenu est

de 30 FPS à une résolution de 1280 pixels et de 38 FPS à une résolution de 640 pixels. Les résultats ont démontré la capacité de l'architecture FPGA à gérer un traitement à la volée de l'interface SLAM HOOFR. Le travail présenté dans cette thèse présente de multiples contributions vers le SLAM embarqué pour les véhicules autonomes. Plusieurs pistes sont envisagées pour étendre ce travail à différents niveaux (capteur, algorithme, et architecture). Au niveau du capteur, nous avons l'intention d'étendre notre jeu de données RGB-D avec des scénarios plus difficiles dans une large gamme de conditions (y compris les changements de météo et d'illumination) pour une étude plus approfondie. Les jeux de données étudiés ont été acquis à des vitesses modérées (20Km/h à 40Km/h), permettant une bonne performance à 30FPS. Des scénarios à des vitesses plus élevées sur des autoroutes sont prévus pour compléter la caractérisation du capteur afin de maintenir ou d'améliorer la précision. De plus, des ensembles de données avec différentes élévations sont prévus. Pour l'instant, l'algorithme peut fonctionner sur un terrain plat. L'intégration d'unités de mesure inertielle (IMU) sera un atout pour assurer la cohérence de l'algorithme dans un environnement présentant des variations d'altitude. Du point de vue de l'architecture, le calcul de pose sera un sujet d'étude pour un système frontal entièrement embarqué sur FPGA. Enfin, l'interface FPGA et caméra devra être étudiée de manière approfondie pour réaliser le traitement à la volée, concevant ainsi un capteur SLAM visuel intelligent.

Bibliography

[1] Raul Mur-Artal, J. M.M. Montiel, and Juan D. Tardos. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 2015. xxiii, 25, 28, 29, 68, 69, 91

[2] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. *IEEE International Conference on Robotics and Automation*, 2014. xxiii, 29, 30

[3] Jakob Engel, Thomas Schops, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. *Computer Vision - ECCV*, 2014. xxiii, 25, 30, 31, 68

[4] R. A. Newcombe et al. Kinectfusion: Real-time dense surface mapping and tracking. *10th IEEE International Symposium on Mixed and Augmented Reality*, 2011. xxiii, 33

[5] Hongyu Wei, Tao Zhang, and Liang Zhang. Gmsk-slam: a new rgb-d slam method with dynamic areas detection towards dynamic environments. *Multimedia Tools and Applications*, 2021. xxiii, 34, 35, 92, 94

[6] Stefano Aldegheri, Nicola Bombieri, Domenico D. Bloisi, and Alessandro Farinelli. Data flow orb-slam for real-time performance on embedded gpu boards. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2019. xxiii, 37, 38

[7] Tianji Ma, Nanyang Bai, Wentao Shi, Xi Wu, Lutao Wang, Tao Wu, and Changming Zhao. Research on the application of visual slam in embedded gpu. *Wireless Communications and Mobile Computing*, 2021. xxiii, 38, 39

[8] Dai-Duong Nguyen, Abdelhafid Elouardi, Sergio A. Rodriguez Florez, and Samir Bouaziz. Hoofr slam system: An embedded vision slam algorithm and its hardware-software mapping-based intelligent vehicles applications. *IEEE Transactions on Intelligent Transportation Systems*, 2019. xxiii, 32, 37, 39, 40, 42, 59, 67, 68, 69, 91, 95, 96, 103, 105, 113, 115, 132, 134, 135

[9] Runze Liu, Jianlei Yang, Yiran Chen, and Weisheng Zhao. Eslam: An energy-efficient accelerator for real-time orb-slam on fpga platform. *Proceedings - Design Automation Conference*, 2019. xxiii, 40, 41, 109

[10] Dai Duong Nguyen, Abdelhafid El Ouardi, Sergio Rodriguez, and Samir Bouaziz. Fpga implementation of hoofr bucketing extractor-based real-time embedded slam applications. *Journal of Real-Time Image Processing*, 2021. xxiii, 24, 37, 42, 43, 108, 110, 111, 114, 134

[11] NVIDIA Technical Blog. https://developer.nvidia.com/blog/nvidia-jetson-agx-xavier-32-teraops-ai-robotics/, Aug 2022. xxiv, 61

[12] Hasitha Muthumala Waidyasooriya, Masanori Hariyama, and Kunio Uchiyama. *Design of FPGA-Based Computing Systems with OpenCL*. Springer International Publishing, 2018. xxiv, 66, 67

[13] D. Hahnel, W. Burgard, D. Fox, and S. Thrun. An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements. *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003. 10

[14] Dirk Holz and Sven Behnke. Mapping with micro aerial vehicles by registration of sparse 3d laser scans. *(eds) Intelligent Autonomous Systems 13. Advances in Intelligent Systems and Computing, vol 302. Springer, Cham*, 2016. 10

[15] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2007. 10, 17, 28

[16] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. *6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, 2007. 10, 17, 23, 28, 30

[17] Richard A. Newcombe, Steven J. Lovegrove, and Andrew J. Davison. Dtam: Dense tracking and mapping in real-time. *Proceedings of the IEEE International Conference on Computer Vision*, 2011. 10, 29, 37

[18] Michael Zollhöfer, Patrick Stotko, Andreas Görlitz, Christian Theobalt, Matthias Nießner, Reinhard Klein, and Andreas Kolb. State of the art on 3d reconstruction with rgb-d cameras. *Computer graphics forum, Wiley Online Library*, 2018. 10

[19] Misha Urooj Khan, Syed Azhar Ali Zaidi, Arslan Ishtiaq, Syeda Ume Rubab Bukhari, Sana Samer, and Ayesha Farman. A comparative survey of lidar-slam and lidar based sensor technologies. *Mohammad Ali Jinnah University International Conference on Computing (MAJICC)*, 2021. 10

[20] Boyu Gao, Haoxiang Lang, and Jing Ren. Stereo visual slam for autonomous vehicles: A review. *IEEE International Conference on Systems, Man, and Cybernetics*, 2020. 10, 17

[21] Kai Berger. A state of the art report on multiple rgb-d sensor research and on publicly available rgb-d datasets. *Computer Vision and Machine Learning with RGB-D Sensors*, 2014. 10

[22] Gayan Brahmanage and Henry Leung. Outdoor rgb-d mapping using intel-realsense. *IEEE SENSORS*, 2019. 11

[23] Soenke Michalik, Soeren Michalik, Jamin Naghmouchi, and Mladen Berekovic. Real-time smart stereo camera based on fpga-soc. *IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, 2017. 11

[24] Gabriel J. Garcia, Carlos A. Jara, Jorge Pomares, Aiman Alabdo, Lucas M. Poggi, and Fernando Torres. A survey on fpga-based sensor systems: Towards intelligent and reconfigurable low-power sensors for computer vision, control and signal processing. *Sensors*, 2014. 11

[25] Konstantinos Boikos and Christos-Savvas Bouganis. A scalable fpga-based architecture for depth estimation in slam. *International Symposium on Applied Reconfigurable Computing*, 2019. 11

[26] Talha Takleh Omar Takleh, Nordin Abu Bakar, Shuzlina Abdul Rahman, Raseeda Hamzah, and Zalilah Abd Aziz. A brief survey on slam methods in autonomous vehicle. *International Journal of Engineering & Technology*, 2018. 12, 129

[27] Patrik Schmuck. Multi-uav collaborative monocular slam. *Proceedings - IEEE International Conference on Robotics and Automation*, 2017. 17

[28] Rachid Latif and Amine Saddik. Slam algorithms implementation in a uav, based on a heterogeneous system: A survey. *4th World Conference on Complex Systems*, 2019. 17, 37

[29] Sebastian Hening, Corey A. Ippolito, Kalmanje S. Krishnakumar, Vahram Stepanyan, and Mircea Teodorescu. 3d lidar slam integration with gps/ins for uavs in urban gps-degraded environments. *AIAA Information Systems-AIAA Infotech at Aerospace*, 2017. 17, 20

[30] Denis Chekhlov, Andrew P. Gee, Andrew Calway, and Walterio Mayol-Cuevas. Ninja on a plane: Automatic discovery of physical planes for augmented reality using visual slam. *6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, 2007. 17

[31] Henning Lategahn, Andreas Geiger, and Bernd Kitt. Visual slam for autonomous ground vehicles. *Proceedings - IEEE International Conference on Robotics and Automation*, 2011. 17

[32] Abdelhafid El Ouardi Abdelouahed Tajer Mounir Amraoui, Rachid Latif. Feature extractors evaluation based v-slam for autonomous vehicles. *Advances in Science, Technology and Engineering Systems Journal*, 2020. 17

[33] Randall C. Smith and Peter C. Cheeseman. On the representation and estimation of spatial uncertainty. *The International Journal of Robotics Research*, 5:56 – 68, 1986. 17

[34] H.F. Durrant-Whyte. Uncertain geometry in robotics. *IEEE Journal on Robotics and Automation*, 1988. 17

[35] J.J. Leonard and H.F. Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. pages 1442–1447 vol.3, 1991. 17

[36] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. *International Joint Conference on Artificial Intelligence*, 2003. 17

[37] Mohammed Chghaf, Sergio Rodriguez, and Abdelhafid El Ouardi. Camera, lidar and multi-modal slam systems for autonomous ground vehicles: a survey. *Journal of Intelligent & Robotic Systems*, 2022. 17, 18, 22

[38] Andrea Macario Barros, Maugan Michel, Yoann Moline, Gwenole Corre, and Frederick Carrel. A comprehensive survey of visual slam algorithms. *Robotics*, 2022. 18

[39] Myriam Servieres, Valerie Renaudin, Alexis Dupuis, and Nicolas Antigny. Visual and visual-inertial slam: State of the art, classification, and experimental benchmarking. *Journal of Sensors*, 2021. 19, 24, 25

[40] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 2018. 19, 92

[41] Stephan Weiss, Markus W. Achtelik, Simon Lynen, Margarita Chli, and Roland Siegwart. Real-time onboard visual-inertial state estimation and self-calibration of mavs in unknown environments. *IEEE International Conference on Robotics and Automation*, 2012. 19

[42] Igor Cvisic, Josip Cesic, Ivan Markovic, and Ivan Petrovic. Soft-slam: Computationally efficient stereo visual simultaneous localization and mapping for autonomous unmanned aerial vehicles. *Journal of Field Robotics*, 2018. 19

[43] Raul Mur-Artal and Juan D. Tardos. Visual-inertial monocular slam with map reuse. *IEEE Robotics and Automation Letters*, 2017. 19

[44] Carlos Campos Martinez, Richard Elvira, Juan J. Gomez Rodriguez, Jose M.M. Montiel, and Juan D. Tardos. Orb-slam3: An accurate open-source library for visual, visual-inertial and multi-map slam, 2020. 19, 30

[45] David Schubert, Thore Goll, Nikolaus Demmel, Vladyslav Usenko, Jorg Stuckler, and Daniel Cremers. The tum vi benchmark for evaluating visual-inertial odometry. *International Conference on Intelligent Robots and Systems*, 2018. 19

[46] Ruyu Liu, Jianhua Zhang, Shengyong Chen, and Clemens Arth. Towards slam-based outdoor localization using poor gps and 2.5d building models. *IEEE International Symposium on Mixed and Augmented Reality*, 2019. 20

[47] Ziyang Hong, Yvan Petillot, and Sen Wang. Radarslam: Radar based large-scale slam in all weathers. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2020. 20

[48] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. *Robotics: Science and Systems*, 2014. 21

[49] Tixiao Shan and Brendan Englot. Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2018. 21

[50] Jianhao Jiao, Huaiyang Huang, Liang Li, Zhijian He, Yilong Zhu, and Ming Liu. Comparing representations in tracking for event camera-based slam. *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2021. 22

[51] Guillermo Gallego, Tobi Delbruck, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew Davison, Joerg Conradt, Kostas Daniilidis, and Davide Scaramuzza. Event-based vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022. 22

[52] Mubariz Zaffar, Shoaib Ehsan, Rustam Stolkin, and Klaus McDonald Maier. Sensors, slam and long-term autonomy: A review. *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2018. 22

[53] Thomas Schops, Torsten Sattler, and Marc Pollefeys. Bad slam: Bundle adjusted direct rgb-d slam. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019. 22

[54] Alexander Andreopoulos and John K. Tsotsos. On sensor bias in experimental methods for comparing interest-point, saliency, and recognition algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2012. 22

[55] Yulong Wu and John Tsotsos. Active control of camera parameters for object detection algorithms. *arXiv preprint arXiv:1705.05685*, 2017. 22

[56] Dorian Galvez-Lopez and Juan D. Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 2012. 23

[57] Dai-Duong Nguyen, Abdelhafid El Ouardi, Emanuel Aldea, and Samir Bouaziz. Hoofr: An enhanced bio-inspired feature extractor. *23rd International Conference on Pattern Recognition*, 2016. 24, 32, 69, 95, 120, 131

[58] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. *Computer Vision – ECCV 2006*, 2006. 24, 108, 134

[59] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 2004. 24

[60] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. *International Conference on Computer Vision*, 2011. 24, 73, 108, 134

[61] Chiang-Heng Chien, Chiang-Ju Chien, and Chen-Chien Hsu. Hw/sw co-design and fpga acceleration of a feature-based visual odometry. *4th International Conference on Robotics and Automation Engineering*, 2019. 24

[62] Chiang-Heng Chien, Chiang-Ju Chien, and Chen-Chien Hsu. Hardware-software co-design of an image feature extraction and matching algorithm. *2nd International Conference on Intelligent Autonomous Systems*, 2019. 24, 37

[63] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2004. 25, 52

[64] Taihu Pire, Thomas Fischer, Javier Civera, Pablo De Cristoforis, and Julio Jacobo Berlles. Stereo parallel tracking and mapping for robot localization. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2015. 26, 111

[65] Alvaro Parra Bustos, Tat Jun Chin, Anders Eriksson, and Ian Reid. Visual slam: Why bundle adjust? *Proceedings - IEEE International Conference on Robotics and Automation*, 2019. 26

[66] Guillaume Bresson, Zayed Alsayed, Li Yu, and Sebastien Glaser. Simultaneous localization and mapping: A survey of current trends in autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 2017. 26, 27

[67] Giorgio Grisetti, Rainer Kummerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2010. 27

[68] Bastien Vincke, Abdelhafid Elouardi, Alain Lambert, and Alain Merigot. Efficient implementation of ekf-slam on a multi-core embedded system. *IECON - 38th Annual Conference on IEEE Industrial Electronics Society*, 2012. 28, 59

[69] Bastien Vincke, Abdelhafid Elouardi, and Alain Lambert. Design and evaluation of an embedded system based slam applications. *IEEE/SICE International Symposium on System Integration*, 2010. 28, 59

[70] Albert A. Jimenez Serrata, Shufan Yang, and Renfa Li. An intelligible implementation of fastslam2.0 on a low-power embedded architecture. *EURASIP Journal on Embedded Systems*, 2017. 28

[71] J. Engel, V. Koltun, and D. Cremers. Direct sparse odometry. *IEEE Transactions On Pattern Analysis And Machine Intelligence*, 2018. 31

[72] X. Gao, R. Wang, N. Demmel, and D. Cremers. Ldso: Direct sparse odometry with loop closure. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2018. 31

[73] Jakob Engel, Jorg Stuckler, and Daniel Cremers. Large-scale direct slam with stereo cameras. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2015. 31

[74] R. Wang, M. Schwörer, and D. Cremers. Stereo dso: Large-scale direct sparse visual odometry with stereo cameras. *International Conference on Computer Vision*, 2017. 31

[75] Taihu Pire, Thomas Fischer, Gaston Castro, Pablo De Cristoforis, Javier Civera, and Julio Jacobo Berlles. S-ptam: Stereo parallel tracking and mapping. *Robotics and Autonomous Systems*, 2017. 32

[76] Raul Mur-Artal and Juan D. Tardos. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 2017. 32, 33, 34, 67, 68, 70, 72, 73, 74, 75, 91, 103, 104, 111

[77] Christos Ttofis, Christos Kyrkou, and Theocharis Theocharides. A low-cost real-time embedded stereo vision system for accurate disparity estimation based on guided image filtering. *IEEE Transactions on Computers*, 2016. 33

[78] Jinglin Zhang, Jean-Francois Nezan, Jean-Gabriel Cousin, and Erwan Raffin. Implementation of stereo matching using a high level compiler for parallel computing acceleration. *Proceedings of the 27th Conference on Image and Vision Computing New Zealand*, 2012. 33

[79] Qiongyao Jin, Yungang Liu, Yongchao Man, and Fengzhong Li. Visual slam with rgb-d cameras. *2019 Chinese Control Conference*, 2019. 33

[80] C. Kerl, J. Sturm, and D. Cremers. Dense visual slam for rgb-d cameras. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013. 33

[81] Rainer Kummerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. G2o: A general framework for graph optimization. *IEEE International Conference on Robotics and Automation*, 2011. 33

[82] Thomas Whelan, Renato F Salas-Moreno, Ben Glocker, Andrew J Davison, and Stefan Leutenegger. Elasticfusion: Real-time dense slam and light source estimation. *The International Journal of Robotics Research*, 2016. 33

[83] Thomas Schops and Torsten Sattler. Bad slam : Bundle adjusted direct rgb-d slam. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019. 34, 36, 70

[84] Jrgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012. 35, 36, 44, 58, 67, 69, 91, 92, 94, 109, 110

[85] A. Handa, T. Whelan, J.B. McDonald, and A.J. Davison. A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM. *IEEE Intl. Conf. on Robotics and Automation*, 2014. 35

[86] Ankur Handa, Richard A. Newcombe, Adrien Angeli, and Andrew J. Davison. Real-time camera tracking: When is high frame-rate best? *ECCV'12: Proceedings of the 12th European conference on Computer Vision*, 2012. 36

[87] Stephan Weiss, Markus W. Achtelik, Simon Lynen, Michael C. Achtelik, Laurent Kneip, Margarita Chli, and Roland Siegwart. Monocular vision for long-term micro aerial vehicle state estimation: A compendium. *Journal of Field Robotics*, 2013. 36

[88] 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*, 2009. 36, 69

[89] A Geiger, P Lenz, C Stiller, and R Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 2013. 36, 39, 56, 69, 94

[90] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research*, 2016. 36, 69

[91] Jakob Engel, Vladyslav Usenko, and Daniel Cremers. A photometrically calibrated benchmark for monocular visual odometry. *arXiv preprint arXiv:1607.02555*, 2016. 36

[92] Ankur Handa, Thomas Whelan, John McDonald, and Andrew J. Davison. A benchmark for rgb-d visual odometry, 3d reconstruction and slam. *IEEE International Conference on Robotics and Automation*, 2014. 36, 44, 91

[93] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. *IEEE Conference on Computer Vision and Pattern Recognition*, 2016. 36

[94] Will Maddern, Geoffrey Pascoe, Chris Linegar, and Paul Newman. 1 year, 1000 km: The oxford robotcar dataset. *The International Journal of Robotics Research*, 2017. 36, 69

[95] Jose-Luis Blanco-Claraco, Francisco-Angel Moreno-Duenas, and Javier Gonzalez-Jimenez. The malaga urban dataset: High-rate stereo and lidar in a realistic urban scenario. *The International Journal of Robotics Research*, 2014. 36, 69

[96] Frank Moosmann and Christoph Stiller. Velodyne SLAM. *Proceedings of the IEEE Intelligent Vehicles Symposium*, 2011. 36, 69

[97] Michael Warren, D. McKinnon, Hu He, and Ben Upcroft. Unaided stereo vision based pose estimation. *Australasian Conference on Robotics and Automation*, 2010. 36, 69

[98] Abdelhamid Dine, Abdelhafid Elouardi, Bastien Vincke, and Samir Bouaziz. Graph-based slam embedded implementation on low-cost architectures: A practical approach. *IEEE International Conference on Robotics and Automation*, 2015. 37

[99] Peter Ondrúška, Pushmeet Kohli, and Shahram Izadi. Mobilefusion: Real-time volumetric surface reconstruction and dense tracking on mobile phones. *IEEE transactions on visualization and computer graphics*, 2015. 37

[100] Min Liu and Tobi Delbruck. Edflow: Event driven optical flow camera with keypoint detection and adaptive block matching. *IEEE Transactions on Circuits and Systems for Video Technology*, 2022. 40

[101] Yakun Wu, Li Luo, Shujuan Yin, Mengqi Yu, Fei Qiao, Hongzhi Huang, Xuesong Shi, Qi Wei, and Xinjun Liu. An fpga based energy efficient ds-slam accelerator for mobile robots in dynamic environment. *Applied Sciences*, 2021. 41, 63

[102] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017. 41

[103] Khronos Group. Opencl - open standard for parallel programming of heterogeneous systems, accessed on August 9, 2022. 41

[104] Keisuke Sugiura and Hiroki Matsutani. A universal lidar slam accelerator system on low-cost fpga. *IEEE Access*, 2022. 43

[105] Edwin B. Olson. Real-time correlative scan matching. *IEEE International Conference on Robotics and Automation*, 2009. 43

[106] Thomas Whelan, Michael Kaess, Hordur Johannsson, Maurice Fallon, John J Leonard, and John McDonald. Real-time large-scale dense rgb-d slam with volumetric fusion. *The International Journal of Robotics Research*, 2015. 44, 94

[107] E. Palazzolo, J. Behley, P. Lottes, P. Giguère, and C. Stachniss. ReFusion: 3D Reconstruction in Dynamic Environments for RGB-D Cameras Exploiting Residuals. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2019. 44

[108] Luca Ulrich, Enrico Vezzetti, Sandro Moos, and Federica Marcolin. Analysis of rgb-d camera technologies for supporting different facial usage scenarios. *Multimedia Tools and Applications*, 2020. 47

[109] Tao Peng, Dingnan Zhang, Ruixu Liu, Vijayan K. Asari, and John S. Loomis. Evaluating the power efficiency of visual slam on embedded gpu systems. *Proceedings of the IEEE National Aerospace Electronics Conference*, 2019. 49, 91

[110] Mathieu Labbe and Francois Michaud. Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of Field Robotics*, 2019. 49, 67, 70, 75, 76, 77, 104

[111] Imad El Bouazzaoui, Sergio Rodriguez Florez, and Abdelhafid El Ouardi. Digiteo_seq1. *Mendeley Data*, page V1, 2021. 51

[112] Imad El Bouazzaoui, Sergio Rodriguez Florez, and Abdelhafid El Ouardi. Digiteo_seq2. *Mendeley Data*, page V1, 2021. 51

[113] Imad El Bouazzaoui, Sergio Rodriguez Florez, and Abdelhafid El Ouardi. Digiteo_seq3_ir-d. *Mendeley Data*, page V1, 2021. 51

[114] Imad El Bouazzaoui, Sergio Rodriguez Florez, and Abdelhafid El Ouardi. Digiteo_seq3_passive-stereo_rgb-d. *Mendeley Data*, page V1, 2021. 51

[115] Imad El Bouazzaoui, Sergio Rodriguez Florez, and Abdelhafid El Ouardi. Digiteo_seq3_stereo. *Mendeley Data*, page V1, 2021. 51

[116] Imad El Bouazzaoui, Sergio Rodriguez Florez, and Abdelhafid El Ouardi. Digiteo_seq3_active-stereo_rgb-d. *Mendeley Data*, page V1, 2021. 51

[117] Johannes L. Schonberger and Jan Michael Frahm. Structure-from-motion revisited. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016. 51, 67

[118] Johannes L. Schonberger, Enliang Zheng, Jan Michael Frahm, and Marc Pollefeys. Pixelwise view selection for unstructured multi-view stereo. *Computer Vision - ECCV 2016*, 2016. 51

[119] Bryan Schauer. Multicore processors - a necessity. *ProQuest discovery guides*, 2008. 59

[120] B. Vincke, A. Elouardi, and A. Lambert. Multiprocessing improvements on a low-cost system based simultaneous localization and mapping. *International Conference on Multimedia Computing and Systems*, 2011. 59

[121] Micaela Verucchi, Luca Bartoli, Fabio Bagni, Francesco Gatti, Paolo Burgio, and Marko Bertogna. Real-time clustering and lidar-camera fusion on embedded platforms for self-driving cars. *Fourth IEEE International Conference on Robotic Computing*, 2020. 59

[122] Sabir Hossain and Deok-jin Lee. Deep learning-based real-time multiple-object detection and tracking from aerial imagery via a flying robot with gpu-based embedded devices. *Sensors*, 2019. 59

[123] Anthony Cowley, Ian D. Miller, and Camillo Jose Taylor. Upslam: Union of panoramas slam. *IEEE International Conference on Robotics and Automation*, 2021. 59, 60

[124] Fynn Schwiegelshohn, Lars Gierke, and Michael Hubner. Fpga based traffic sign detection for automotive camera systems. *10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip*, 2015. 59, 121

[125] C. T. Johnston, K. T. Gribbon, and D. G. Bailey. Implementing image processing algorithms on fpgas. *Proceedings of the Eleventh Electronics New Zealand Conference, ENZCon'04, Palmerston North*, 2004. 59

[126] Konstantinos Boikos and Christos-Savvas Bouganis. A scalable fpga-based architecture for depth estimation in SLAM. *Applied Reconfigurable Computing, Springer*, 2019. 59

[127] G.P. Stein, E. Rushinek, G. Hayun, and A. Shashua. A computer vision system on a chip: a case study from the automotive domain. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Workshops*, 2005. 59, 60

[128] Antonio De la Piedra, An Braeken, and Abdellah Touhafi. Sensor systems based on fpgas and their applications: A survey. *Sensors*, 2012. 60

[129] Mihai Bujanca, Xuesong Shi, Matthew Spear, Pengpeng Zhao, Barry Lennox, and Mikel Lujan. Robust slam systems: Are we there yet? *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2021. 60

[130] Chengying Cai, Jichao Jiao, Wei Xu, Min Pang, and Jianye Dong. Hard-lite slam: A hybrid detector based real-time slam system. *6th International Conference on Innovation in Artificial Intelligence*, 2022. 60

[131] Zeyad Assi Obaid Nasri Sulaiman, MH Marhaban, and MN Hamidon. Design and implementation of fpga-based systems-a review. *Australian Journal of Basic and Applied Sciences*, 2009. 62

[132] M.B.I. Reaz, F. Mohd-Yasin, S.L. Tan, H.Y. Tan, and M.I. Ibrahimy. Partial encryption of compressed images employing fpga. *IEEE International Symposium on Circuits and Systems*, 2005. 63

[133] Zhilin Xu, Jincheng Yu, Chao Yu, Hao Shen, Yu Wang, and Huazhong Yang. Cnn-based feature-point extraction for real-time visual slam on embedded fpga. *IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines*, 2020. 63

[134] Miguel Arias-Estrada and César Torres-Huitzil. Real-time field programmable gate array architecture for computer vision. *Journal of Electronic Imaging*, 2001. 63

[135] Abdelhamid Dine. Localisation et cartographie simultanées par optimisation de graphe sur architectures hétérogènes pour l'embarqué. (embedded graph-based simultaneous localization and mapping on heterogeneous architectures), 2016. 63

[136] Fahad Bin Muslim, Liang Ma, Mehdi Roozmeh, and Luciano Lavagno. Efficient fpga implementation of opencl high-performance computing applications via high-level synthesis. *IEEE Access*, 2017. 63

[137] FPGA Intel. Intel fpga sdk for opencl pro edition (2020), 2020. 66

[138] Chao Sheng, Shuguo Pan, Wang Gao, Yong Tan, and Tao Zhao. Dynamic-dso: Direct sparse odometry using objects semantic information for dynamic environments. *Applied Sciences*, 2020. 68

[139] Ruben Gomez-Ojeda, Francisco Angel Moreno, David Zuniga-Noel, Davide Scaramuzza, and Javier Gonzalez-Jimenez. Pl-slam: A stereo slam system through the combination of points and line segments. *IEEE Transactions on Robotics*, 2019. 68, 69

[140] Christopher Mei, Gabe Sibley, Mark Cummins, Paul Newman, and Ian Reid. Rslam: A system for large-scale mapping in constant-time using stereo. *International Journal of Computer Vision*, 2011. 68

[141] Yanqing Liu, Dongchen Zhu, Jingquan Peng, Xianshun Wang, Lei Wang, Lili Chen, Jiamao Li, and Xiaolin Zhang. Real-time robust stereo visual slam system based on bionic eyes. *IEEE Transactions on Medical Robotics and Bionics*, 2020. 69

[142] Vincent Angladon, Simone Gasparini, Vincent Charvillat, Tomislav Pribanic, Tomislav Petkovic, Matea Donlic, Benjamin Ahsan, and Frederic Bruel. An evaluation of real-time rgb-d visual odometry algorithms on mobile devices. *Journal of Real-Time Image Processing*, 2019. 70, 91

[143] Felix Endres, Jurgen Hess, Jurgen Sturm, Daniel Cremers, and Wolfram Burgard. 3-d mapping with an rgb-d camera. *IEEE Transactions on Robotics*, 2014. 70

[144] Qiang Fu, Hongshan Yu, Lihai Lai, Jingwen Wang, Xia Peng, Wei Sun, and Mingui Sun. A robust rgb-d slam system with points and lines for low texture indoor environments. *IEEE Sensors Journal*, 2019. 70

[145] Monica Carfagni, Rocco Furferi, Lapo Governi, Michaela Servi, Francesca Uccheddu, and Yary Volpe. On the performance of the intel sr300 depth camera: Metrological and critical characterization. *IEEE Sensors Journal*, 2017. 70

[146] Monica Carfagni, Rocco Furferi, Lapo Governi, Chiara Santarelli, Michaela Servi, Francesca Uccheddu, and Yary Volpe. Metrological and critical characterization of the intel d415 stereo depth camera. *Sensors (Switzerland)*, 2019. 71

[147] Elise Lachat, Helene Macher, Tania Landes, and Pierre Grussenmeyer. Assessment and calibration of a rgb-d camera (kinect v2 sensor) towards a potential use for close-range 3d modeling. *Remote Sensing*, 2015. 71

[148] Lina M. Paz, Pedro Pinies, Juan D. Tardos, and Jose Neira. Large-scale 6-dof slam with stereo-in-hand. *IEEE Transactions on Robotics*, 2008. 74, 78, 130

[149] Gaspard Florentz and Emanuel Aldea. Superfast: Model-based adaptive corner detection for scalable robotic vision. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014. 77, 129

[150] Michael J. Milford and Gordon. F. Wyeth. Seqslam: Visual route-based navigation for sunny summer days and stormy winter nights. *IEEE International Conference on Robotics and Automation*, 2012. 78, 130

[151] Mark Cummins and Paul Newman. Appearance-only slam at large scale with fab-map 2.0. *The International Journal of Robotics Research*, 2011. 78, 130

[152] Zichao Zhang, Henri Rebecq, Christian Forster, and Davide Scaramuzza. Benefit of large field-of-view cameras for visual odometry. *IEEE International Conference on Robotics and Automation*, 2016. 82

[153] David Schubert, Nikolaus Demmel, Vladyslav Usenko, Jörg Stückler, and Daniel Cremers. Direct sparse odometry with rolling shutter. *European Conference on Computer Vision (ECCV)*, 2018. 82

[154] Anders Grunnet-Jepsen, John N Sweetser, and John Woodfill. Best-known-methods for tuning intel® realsense d400 depth cameras for best performance. *Intel Corporation: Satan Clara, CA, USA*, 2018. 82, 97

[155] Masaya Kaneko, Kazuya Iwami, Torn Ogawa, Toshihiko Yamasaki, and Kiyoharu Aizawa. Mask-slam: Robust feature-based monocular slam by masking using semantic segmentation. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2018. 91, 92, 94, 124, 137

[156] Jianfang Chang, Na Dong, and Donghui Li. A real-time dynamic object segmentation framework for slam system in dynamic scenes. *IEEE Transactions on Instrumentation and Measurement*, 2021. 91

[157] Chao Yu, Zuxin Liu, Xin Jun Liu, Fugui Xie, Yi Yang, Qi Wei, and Qiao Fei. Ds-slam: A semantic visual slam towards dynamic environments. *IEEE International Conference on Intelligent Robots and Systems*, 2018. 91, 92, 94

[158] Nikolas Brasch, Aljaz Bozic, Joe Lallemand, and Federico Tombari. Semantic monocular slam for highly dynamic environments. *IEEE International Conference on Intelligent Robots and Systems*, 2018. 91

[159] Imad El Bouazzaoui, Sergio Rodriguez Florez, and Abdelhafid El Ouardi. Enhancing rgb-d slam performances considering sensor specifications for indoor localization. *IEEE Sensors Journal*, 2021. 91

[160] Chenyang Zhang. Pl-gm:rgb-d slam with a novel 2d and 3d geometric constraint model of point and line features. *IEEE Access*, 2021. 91

[161] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. *Proceedings of the IEEE International Conference on Computer Vision*, 2011. 91

[162] Rafael Grompone von Gioi, Jeremie Jakubowicz, Jean-Michel Morel, and Gregory Randall. Lsd: a line segment detector. *Image Processing On Line*, 2012. 91

[163] Jinkyu Lee, Muhyun Back, Sung Soo Hwang, and Il Yong Chun. Improved real-time monocular slam using semantic segmentation on selective frames. *arXiv*, 2021. 92, 94

[164] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017. 92

[165] Wanfang Xie, Peter Xiaoping Liu, and Minhua Zheng. Moving object segmentation and detection for robust rgbd-slam in dynamic environments. *IEEE Transactions on Instrumentation and Measurement*, 2021. 92, 94, 124, 137

[166] Yu Liu, Yilin Wu, and Wenzhao Pan. Dynamic rgb-d slam based on static probability and observation number. *IEEE Transactions on Instrumentation and Measurement*, 2021. 92, 94

[167] Ioan Andrei Barsan, Peidong Liu, Marc Pollefeys, and Andreas Geiger. Robust dense mapping for large-scale dynamic environments. *IEEE International Conference on Robotics and Automation*, 2019. 93, 94, 124, 137

[168] Jifeng Dai, Kaiming He, and Jian Sun. Instance-aware semantic segmentation via multi-task network cascades. *IEEE Conference on Computer Vision and Pattern Recognition*, 2016. 93

[169] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. *Conference on robot learning*, 2017. 94

[170] Ronald K Pearson. Outliers in process modeling and identification. *IEEE Transactions on control systems technology*, 2002. 99

[171] Jani Posio, Kauko Leiviskä, Jari Ruuska, and Paavo Ruha. Outlier detection for 2d temperature data. *IFAC proceedings volumes*, 2008. 99

[172] O. Chum and J. Matas. Matching with prosac - progressive sample consensus. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2005. 100, 133

[173] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 2004. 100, 108, 114, 133, 134

[174] Song Wu, Die Hu, Shadi Ibrahim, Hai Jin, Jiang Xiao, Fei Chen, and Haikun Liu. When fpga-accelerator meets stream data processing in the edge. *IEEE 39th International Conference on Distributed Computing Systems*, 2019. 108, 134

[175] Kohei Nakamura, Ami Hayashi, and Hiroki Matsutani. An fpga-based low-latency network processing for spark streaming. *IEEE International Conference on Big Data (Big Data)*, 2016. 108, 134

[176] Weikang Fang, Yanjun Zhang, Bo Yu, and Shaoshan Liu. Fpga-based orb feature extraction for real-time visual slam. *International Conference on Field Programmable Technology*, 2017. 109

[177] Vibhakar Vemulapati and Deming Chen. Orb-based slam accelerator on soc fpga. *arXiv preprint arXiv:2207.08405*, 2022. 109

[178] A. Alahi, R. Ortiz, and P. Vandergheynst. Freak: Fast retina keypoint. *IEEE Conference on Computer Vision and Pattern Recognition*, 2012. 120

[179] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2005. 120