A vision system based real-time SLAM applications

Dai-Duong Nguyen

université PARIS-SACLAY

SATIE école normale supérieure paris-saclay

UNIVERSITÉ PARIS SUD
Comprendre le monde, construire l'avenir

NNT: 2018SACLS518

A VISION SYSTEM BASED REAL-TIME SLAM APPLICATIONS

Thèse de doctorat de l'Université Paris-Saclay

préparée à l'Université Paris-Sud

Laboratoire des Systèmes et Applications des Technologies de l'Information et de l'Energie

École doctorale n°580
Sciences et Technologies de l'Information et de la Communication (STIC)

Specialité de doctorat : Robotique

Thèse présentée et soutenue à Gif-sur-Yvette
le 07/12/2018, par

M. Dai-Duong NGUYEN

Composition du jury:

M. Roland CHAPUIS Professeur, Institut Pascal - Université Clermont Auvergne	**Président du jury**
M. Michel DEVY Directeur de recherche, LAAS-CNRS Toulouse	**Rapporteur**
M. Jean-François NEZAN Professeur, INSA de Rennes	**Rapporteur**
Mme Anne VERROUST-BLONDET Chargée de recherche, INRIA Paris	**Examinatrice**
M. Samir BOUAZIZ Professeur, SATIE - Université Paris-Sud	**Examinateur**
M. Sergio RODRIGUEZ Maître de conférences, SATIE - Université Paris-Sud	**Examinateur**
M. Abdelhafid ELOUARDI Maître de conférences - HDR, SATIE - Université Paris-Sud	**Directeur de thèse**

Thèse de doctorat

A Vision System based real-time SLAM applications

December 2018

Abstract

SLAM (Simultaneous Localization And Mapping) has an important role in several applications such as autonomous robots, smart vehicles, unmanned aerial vehicles (UAVs) and others. Nowadays, real-time vision based SLAM applications becomes a subject of widespread interests in many researches. One of the solutions to solve the computational complexity of image processing algorithms, dedicated to SLAM applications, is to perform high or/and low level processing on co-processors in order to build a System on Chip. Heterogeneous architectures have demonstrated their ability to become potential candidates for a system on chip in a hardware software co-design approach. The aim of this thesis is to propose a vision system implementing a SLAM algorithm on a heterogeneous architecture (CPU-GPU or CPU-FPGA). The study will allow verifying if these types of heterogeneous architectures are advantageous, what elementary functions and/or operators should be added on chip and how to integrate image-processing and the SLAM Kernel on a heterogeneous architecture (i. e. How to map the vision SLAM on a System on Chip).

There are two parts in a visual SLAM system: Front-end (feature extraction, image processing) and Back-end (SLAM kernel). During this thesis, we studied several features detection and description algorithms for the Front-end part. We have developed our own algorithm denoted as HOOFR (Hessian ORB Overlapped FREAK) extractor which has a better compromise between precision and processing times compared to those of the state of the art. This algorithm is based on the modification of the ORB (Oriented FAST and rotated BRIEF) detector and the bio-inspired descriptor: FREAK (Fast Retina Keypoint). The improvements were validated using well-known real datasets. Consequently, we propose the HOOFR-SLAM Stereo algorithm for the Back-end part. This algorithm

uses images acquired by a stereo camera to perform simultaneous localization and mapping. The HOOFR SLAM performances were evaluated on different datasets (KITTI, New-College , Malaga, MRT, St-Lucia, ...).

Afterward, to reach a real-time system, we studied the algorithmic complexity of HOOFR SLAM as well as the current hardware architectures dedicated for embedded systems. We used a methodology based on the algorithm complexity and functional blocks partitioning. The processing time of each block is analyzed taking into account the constraints of the targeted architectures. We achieved an implementation of HOOFR SLAM on a massively parallel architecture based on CPU-GPU. The performances were evaluated on a powerful workstation and on architectures based embedded systems. In this study, we propose a system-level architecture and a design methodology to integrate a vision SLAM algorithm on a SoC. This system will highlight a compromise between versatility, parallelism, processing speed and localization results. A comparison related to conventional systems will be performed to evaluate the defined system architecture.

In order to reduce the energy consumption, we have studied the implementation of the Front-end part (image processing) on an FPGA based SoC system. The SLAM kernel is intended to run on a CPU processor. We proposed a parallelized architecture using HLS (High-level synthesis) method and OpenCL language programming. We validated our architecture for an Altera Arria 10 SoC. A comparison with systems in the state-of-the-art showed that the designed architecture presents better performances and a compromise between power consumption and processing times.

Résumé

SLAM (localisation et cartographie simultanées) joue un rôle important dans plusieurs applications telles que les robots autonomes, les véhicules intelligents, les véhicules aériens sans pilote (UAV) et autres. De nos jours, les applications SLAM basées sur la vision en temps réel deviennent un sujet d'intérêt général dans de nombreuses recherches. L'une des solutions pour résoudre la complexité de calcul des algorithmes de traitement d'image, dédiés aux applications SLAM, consiste à effectuer un traitement de haut ou de bas niveau sur les coprocesseurs afin de créer un système sur puce. Les architectures hétérogènes ont démontré leur capacité à devenir des candidats potentiels pour un système sur puce dans une approche de co-conception de logiciels matériels. L'objectif de cette thèse est de proposer un système de vision implémentant un algorithme SLAM sur une architecture hétérogène (CPU-GPU ou CPU-FPGA). L'étude permettra d'évaluer ce type d'architectures et contribuer à répondre aux questions relatives à la définition des fonctions et/ou opérateurs élémentaires qui devraient être implantés et comment intégrer des algorithmes de traitement de données tout en prenant en considération l'architecture cible (dans un contexte d'adéquation algorithme architecture).

Il y a deux parties dans un système SLAM visuel : Front-end (extraction des points d'intéret) et Back-end (coeur de SLAM). Au cours de la thèse, concernant la partie Front-end, nous avons étudié plusieurs algorithmes de détection et description des primitives dans l'image. Nous avons développé notre propre algorithme intitulé HOOFR (Hessian ORB Overlapped FREAK) qui possède une meilleure performance par rapport à ceux de l'état de l'art. Cet algorithme est basé sur la modification du détecteur ORB et du descripteur bio-inspiré FREAK. Les résultats de l'amélioration ont été validés en utilisant des jeux de données réel connus. Ensuite, nous avons proposé l'algorithme HOOFR-SLAM Stereo pour la partie Back-end. Cette algorithme utilise les images acquises par une paire

de caméras pour réaliser la localisation et cartographie simultanées. La validation a été faite sur plusieurs jeux de données (KITTI, New_College, Malaga, MRT, St_lucia, ...).

Par la suite, pour atteindre un système temps réel, nous avons étudié la complexité algorithmique de HOOFR SLAM ainsi que les architectures matérielles actuelles dédiées aux systèmes embarqués. Nous avons utilisé une méthodologie basée sur la complexité de l'algorithme et le partitionnement des blocs fonctionnels. Le temps de traitement de chaque bloc est analysé en tenant compte des contraintes des architectures ciblées. Nous avons réalisé une implémentation de HOOFR SLAM sur une architecture massivement parallèle basée sur CPU-GPU. Les performances ont été évaluées sur un poste de travail puissant et sur des systèmes embarqués basés sur des architectures. Dans cette étude, nous proposons une architecture au niveau du système et une méthodologie de conception pour intégrer un algorithme de vision SLAM sur un SoC. Ce système mettra en évidence un compromis entre polyvalence, parallélisme, vitesse de traitement et résultats de localisation. Une comparaison avec les systèmes conventionnels sera effectuée pour évaluer l'architecture du système définie.

Vue de la consommation d'énergie, nous avons étudié l'implémentation la partie Front-end sur l'architecture configurable type soc-FPGA. Le SLAM kernel est destiné à être exécuté sur un processeur. Nous avons proposé une architecture par la méthode HLS (High-level synthesis) en utilisant langage OpenCL. Nous avons validé notre architecture sur la carte Altera Arria 10 soc. Une comparaison avec les systèmes les plus récents montre que l'architecture conçue présente de meilleures performances et un compromis entre la consommation d'énergie et les temps de traitement.

Publications

- *Journal:*

1. Dai Duong NGUYEN, Abdelhafid ELOUARDI, Sergio RODRIGUEZ and Samir BOUAZIZ, "*HOOFR SLAM System: an Embedded Vision SLAM Algorithm and its Hardware-Software Mapping Based Intelligent Vehicles Applications*". **IEEE Transaction on Intelligent Transportation Systems (IEEE ITS), 2018**.

- *Conferences:*

1. Dai-Duong Nguyen, Abdelhafid Elouardi, Emanuel Aldea, Samir Bouaziz, "HOOFR: An Enhanced Bio-Inspired Feature Extractor". The 23rd International Conference on Pattern Recognition, **ICPR 2016**.

2. Dai Duong Nguyen, Abdelhafid Elouardi and Samir Bouaziz, "Enhanced Bio-Inspired Feature Extraction for Embedded Application". The 14th IEEE International Conference on Control, Automation, Robotics and Vision, **ICARCV 2016**.

3. Dai Duong Nguyen, Mohamed Abouzahir, Abdelhafid Elouardi, Bruno Larnaudie and Samir Bouaziz, "*GPU Accelerated Robust-Laser based Fast Simultaneous Localization and Mapping*". The 16th IEEE International Conference on Scalable Computing and Communications, **SCALCOM 2016**.

Acknowledgement

First and foremost, I would like to thank Pr. Roland CHAPUIS for being the chairman of my thesis jury and for his role as an examiner. I also want to thank Pr. Michel DEVY and Pr. Jean-François NEZAN for their work as reviewers, and Mrs Anne VERROUST-BLONDET for her role as an examier in the jury.

I want to thank especially my PhD supervisor Mr. Abdelhafid ELOUARDI who followed me throughout these three years and to have framed, guided, directed, helped and supported me during the whole of my thesis. I thank the MOSS Digiteo Labs group of the SATIE laboratory, with the leader Mr. Roger Reynaud, for allowing me to to carry out my PhD work within the group and for the pleasant work environment that they offered me throughout these three years. My sincere thanks go to the entire embedded systems team of MOSS Digiteo Labs for the warm welcome that I received and to all the people with whom I shared my knowledge and my passion for robotics.

I would also like to thank Mr Samir BOUAZIZ, Mr Emanuel ALDEA and Mr Sergio RODRGUEZ for their useful advices and discussions. I'm grateful to the Paris Sud University, Paris Saclay, to have me granted funding during these three years of thesis. I will be particularly grateful for this generous contribution that has given me a breath to continue my thesis work.

Finally, I want to thank my lovely wife and my lovely son who have always supported and encouraged me throughout these years of schooling and my parents without whom I will not have realized my education. I will be eternally grateful to you for encouraging me in difficult times.

Contents

List of Tables

List of Figures

Introduction

Simultaneous Localization and Mapping

There are three main areas in the issue of autonomous navigation of mobile robots: localization, reconstruction and path planning [1]. Localization is the determination of the current robot pose in an environment. Reconstruction integrates the partial observations of surrounding objects into a single consistent model and path planning determines an appropriate path in the map to navigate through the environment. In the literature, reconstruction is also called as mapping. At the beginning, localization and mapping were studied independently, however researchers recognized then that they are dependent. It means that, having a good localization in an environment requires a correct map, but in order to construct a correct map it is necessary to be properly localized when elements are added to the map. This problem is currently known as Simultaneous Localization and Mapping (SLAM).

To build a map from the environment, the entity must be equipped by sensors that allow it to perceive, observe and achieve the measurements of the elements from the surrounding scenes. These sensors are classified into two kinds: exteroceptive and proprioceptive. Exteroceptive sensors are the sensors which allow the entity to obtain the information from environment such as: range lasers [2, 3, 4], sonar [5], cameras [6, 7, 8, 9] or global positioning systems (GPS) [10]. Each of these sensors has its own advantages and draw-backs. For the first three aforementioned sensors, only local views of the environment can be observed. Laser and sonar allow precise and very dense information of the environment structure. Nevertheless, the problem is that they are not useful in highly cluttered environments or for recognizing objects. They are also expensive, heavy and consist of large pieces of equipment, making their use difficult for humanoids or airborne

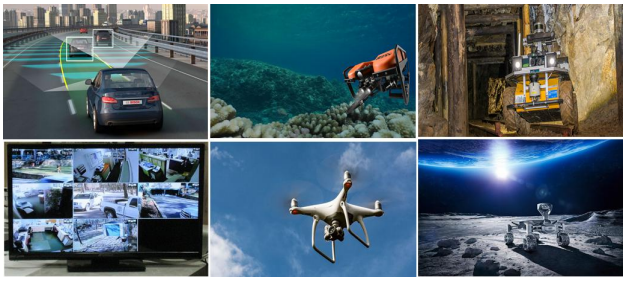


Figure 1: SLAM application in different areas

robots. Cameras are easy for installation and offer a flexibility to switch between different applications but they are noisy and require a careful calibration. For a GPS sensor, it does not work well in narrow streets, under water and occasionally is not available for indoor environments. On the other hand, a proprioceptive sensor allows the entity to obtain measurements of itself like velocity, position change and acceleration. Some widely used proprioceptive sensors are: encoders, accelerometers and gyroscopes. These allow computing an incremental estimate of the entity's movements based on means of a dead-reckoning navigation method (deduced-reckoning). They are however not sufficient to have an accurate estimation of the entity's position all the time due to their inherent noise and errors are cumulative.

SLAM systems are employed in several applications such as: innovation in unmanned ground vehicles navigation [10], underwater exploration [11, 12], high risk or difficult navigation environments [13], visual surveillance systems [14], unmanned aerial vehicles (UAVs) [15], planets exploration [16] as shown in figure 1. Besides, terrestrial map construction [17], augmented reality applications [18, 19] or medicine [20] can also be named as examples.

Motivation

Our work is related to autonomous vehicles which is a current trend in many researches. An autonomous vehicle (also known as a driverless car and a self-driving car) is a vehicle that is capable of sensing its environment and navigating without human input. In a general manner, localizing a vehicle is an essential functionality to perform any other perception or planification task. Predicting the evolution of others obstacles on the road

and choosing which maneuver is the most appropriate require to know exactly where the ego-vehicle is located and how the surrounding environment look like. The map offers a first level of perception that is needed in order to make an appropriate decision.

The SLAM framework provides an answer to this problematic. It is considered as one of the primaries towards a truly autonomous robot, and as such is an essential aspect of self-driving cars. However, many issues are still preventing the use of SLAM algorithms with vehicles that should be able to drive for hundreds of kilometers in different conditions. In recent years, visual SLAM has reached a significant level of maturity with a number of robust solutions being reported in the literature. Although these techniques permit the construction of an accurate map of an environment and are argued as a real-time performance, the fact that they are real-time in a small scale. When the environment is larger, their execution time becomes a severe problem. This issue has motivated the development of a lighter algorithm which could keep a low complexity over time.

There are many kinds of sensors which could be integrated in vehicles for solving SLAM problem. The usually used tool for localization is a GPS. However, a GPS cannot be used indoors. There are many indoor localization tools including Lidar, UWB, WiFi AP , among which using cameras to localization is the most flexible and low cost one. Moreover, cameras are ubiquitous on mobile phones that people carry with every day. Due to this reason, SLAM based camera (visual slam) provides a great motivation for researchers.

Furthermore, with heterogeneous architecture becoming more and more common place in consumer electronic devices, initially only in desktop PCs but more recently in embedded platforms such as phones and tablets, we can expect in the coming years that sufficient highly parallel processing power will be available in all kinds of platforms. There is a well matched coupling between data processing needs of visual SLAM and device processing capabilities. However, a heterogeneous architecture brings with it a need to adapt and study the algorithm partitioning that can specifically exploit parallel processing methods.

Objectives and Contribution

The main objective of this thesis is to propose of a visual SLAM algorithm and the study of the portability of this algorithm on heterogeneous architectures. The system design requires phases of proposing and validating the functionality of the vSLAM algorithm while the study of the portability includes the analysis of the algorithm complexity and the architecture constraints in a software-hardware mapping approach. This mapping is aimed to reduce the execution time and hence to have real-time performances.

The first contribution consists in proposing an algorithm called HOOFR extractor which aims to address the front-end part of a visual SLAM system for detecting, describing and matching image features. Our detector is the combination of ORB with a Hessian score, while our descriptor employs a human retina based descriptor consisting of a FREAK detector version with an enhanced overlapping. Based on experiments, our proposal offers a better compromise between speed and matching quality against others state of the art algorithms.

The second contribution is a new method for back-end part of a stereo visual SLAM system. The proposed algorithm uses key-points detected by HOOFR extractor so that it is denoted as HOOFR SLAM. Our novel approach employs a "Weighted Mean" of multiple neighbor poses. It provides a localization estimation after computing the camera poses (6-DOF rigid transformation) from the current image frame to previous neighbor frames. Taking advantage of camera motion, we conjointly incorporate two types of stereo modes: "Static Stereo" mode (SS) through the fixed-baseline of left-right cameras setup along with the "Temporal Multi-view Stereo" mode (TMS). Moreover, instead of computing beforehand the disparity of SS mode for all key-points set, the disparity map in scale estimation step is limited to the inliers of TMS mode so as to reduce the computational cost.

The third contribution of our work is presenting a capability of implementing HOOFR SLAM on CPU-GPU heterogeneous architectures where a powerful PC and an embedded platforms (Nvidia Tegra X1) are considered. Moreover, we also present our researches on emdedding the front-end part on a CPU-FPGA embedded SoC architecture. Our motivation is that FPGA devices can provide a better compromise between processing speed and

energy consumption. Moreover, there is a continuously widening performance gap favoring FPGAs from one generation to the next, especially with regards to high performance computing or data center applications. The enhanced performance combined with a superior power efficiency results in an increased performance-to-power-efficiency of FPGAs in comparison to both GPUs and CPUs.

Thesis Organization

The thesis is organized into several chapters as following:

- Chapter 1 provides an overview of the visual SLAM problem and an introduction to the formalization of visual SLAM system. A section will provide a discussion on heterogeneous architectures used to implement SLAM applications.

- Chapter 2 presents our methodology to implement and evaluate a vision SLAM system. We presents several real datasets and different heterogeneous architectures used in this thesis for performances evaluation.

- Chapter 3 presents our proposed method named Hessian ORB - Overlapped FREAK (HOOFR) for detecting, describing and matching image features. In practice, feature extractor is the very first part in a visual SLAM system. A suitable feature extractor is indispensable to provide a high localization precision. This chapter will introduce some well-known algorithms compared to our proposal with an enhanced matching quality.

- Chapter 4 presents the HOOFR SLAM algorithm and the validation of its functionality on several well-known datasets. This method is denoted as HOOFR SLAM since it uses features detected by HOOFR extractor for both tracking and loop closing. HOOFR SLAM takes images from a stereo camera for each input frame. Compared to other SLAM algorithms in the literature, HOOFR SLAM is proposed with an intention to have a lower complexity, lower resources requirement and suitable to be implemented on embedded architectures.

- Chapter 5 discusses a hardware software mapping of the HOOFR SLAM. To this end, a heterogeneous CPU-GPU architecture based vision system is considered. A thorough and extensive experimental evaluation of our algorithm implemented on an automotive architecture (the NVIDIA Tegra TX1 system) is studied and analyzed.

- Chapter 6 discusses the design of front-end part (HOOFR extractor) on a FPGA-based heterogeneous architecture using High Level Synthesis method. It is the first step of embedding the whole SLAM system on a SoC architecture based on FPGA. The motivation of this approach is to have a system with higher efficiency in terms of power consumption.

Finally, we summarize the work done in this thesis and we give comments on possible avenues for future researches.

Chapter 1

Visual SLAM Systems

1.1 Visual SLAM

In recent years, people focus on the tendency of using camera as the external perception sensor to solve the problem of SLAM [21, 22, 19, 23, 24, 25, 26, 27]. The main reason for this trend is related to the capability for a system based on cameras to obtain range information, and also retrieving the environment's appearance, color and texture, providing the possibility of integrating other high-level tasks like people detection or object recognition. Furthermore, cameras are becoming cheaper and consuming less energy. When a SLAM application employed a camera as the only exteroceptive sensor, it is called a visual SLAM application. The terms vision-based SLAM [6, 7] or vSLAM [28] are also used.

However, people can integrate information from proprioceptive sensors into visual SLAM systems in order to increase accuracy and robustness. This approach could be found in Visual-Inertial SLAM proposed by Jones [29] or Visual-Odometer SLAM used in FAST-SLAM algorithm [30]. In fact, when camera is used as the only system of perception (without making use of information extracted from the robot odometry or inertial sensors), it can be denoted as vision-only SLAM [22, 21] or camera-only SLAM [31]. There are many challenges for a visual SLAM system working in a real-world condition such as: dynamic environments, environments with too many or very few salient features, large scale environments, erratic movements of the camera and partial or total occlusions

of the sensor. An essential purpose of a successful visual SLAM system is the ability to operate correctly despite these difficulties.

Considering the way of using camera, we have two approaches: multi-camera and mono-camera:

- Multi-camera consists in using binocular, trinocular or multiple cameras with their fields of vision partially overlapped. It offers the advantage of being able to easily and accurately calculate the real 3D positions of the landmarks contained in the scene, by means of triangulation [32]. This information is of great utility in the visual SLAM problem. The first works on visual navigation were based on a binocular stereo configuration [33, 34]. The works of Konolige and Agrawal [35], Konolige et al. [36], Mei et al. [37] represent also the most current and effective binocular stereo SLAM systems. However, in many cases it is difficult to have a device with binocular or trinocular stereo cameras due to their high costs. An alternative is to use a pair of monocular cameras (for example webcams), which leads to consider different aspects such as: the camera synchronization through the use of hardware or software, the different responses of each CCD sensor to color and luminance, and the mechanical alignment according to the geometry scheme chosen (parallel or convergent axes). Some works also make use of multi-camera rigs with or without overlapping between the views [38, 39] and cameras with special lens such as wide-angle [40] or omnidirectional [41] with the goal of increasing visual range and thus decrease, to some extent, the cumulative error of pose estimation. Recently, RGB-D (color images and depth maps) sensors have been used to map indoor environments [42], proving to be a promising alternative for SLAM applications.

- While multi-camera approach is the traditional method, the idea of utilizing mono-camera [8, 25] recently has become popular due to the less calibration complexity. This is probably also because it is now easier to access a single camera than a stereo pair, through cell phones, personal digital assistants or personal computers. This monocular approach offers a very simple, flexible and economic solution in terms of hardware and processing times. However, when localization and mapping

is being done with a single camera, the map will suffer from a scale ambiguity problem [43, 44]. To obtain 3D information from a single camera, two cases exist depending on the a priori knowledge of the camera. The first is with the knowledge of the intrinsic parameters. The environment structure and the extrinsic parameters in this alternative are recovered with an undetermined scale-factor. Scale is only determined if the real distance between two points in space is known. The second is where only correspondences are known. In this latter case, the reconstruction is made up to a projective transformation (4 ambiguous cases).

Independently of the configuration used, cameras have to be calibrated offline or online, manually or automatically. Calibration estimates intrinsic and extrinsic parameters of the camera, the firsts depend on the camera's geometry (focal length and principal point), while the others depend on the camera's position in world-space (rotation and translation with respect to a coordinate system). These parameters are normally estimated from a set of images that contain multiple views of a checkerboard calibration pattern, to relate the image's coordinates with the real-world coordinates [45]. Many tools exist to execute the process of calibration, some of them are: the calibration functions of OpenCV (2009) (based on the Zhang algorithm [46]), Camera Calibration Toolbox for Matlab [47], Tsai Camera Calibration Software [48], OCamCalib Toolbox for omnidirectional cameras [49], and Multi-Camera Self-Calibration to calibrate several cameras (at least 3) [50]. If the camera calibration is performed off-line, then it is assumed that the intrinsic properties of the camera will not change during the entire period of the application. This is the most popular option, since it reduces the number of parameters calculated online. Nevertheless, the intrinsic camera information may change due to some environmental factors of the environment, such as humidity or temperature. Furthermore, a robot that works in real world conditions can be hit or damaged, which could invalidate the previously acquired calibration [51].

1.2 Visual SLAM system formalization

A vSLAM system consists of 2 principal components as shown in the figure 1.1: Image-processing part (front-end) and SLAM-core part (back-end).

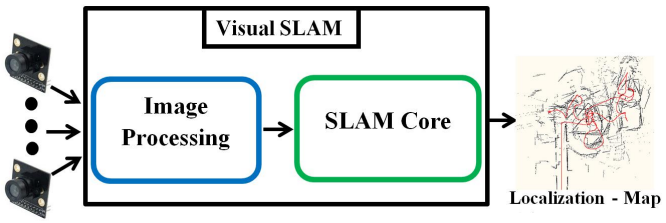


Figure 1.1: Visual SLAM system

1.2.1 Image-Processing (front-end task)

The content of image-processing (IP) task depends on the method of the SLAM-core algorithm. The actual vSLAM algorithms could be categorized into 3 approaches: feature-based, direct, and RGB-D camera-based approach. In feature-based and RGB-D camera-based approaches, IP is composed of detecting points of interest (features) in the input frame, computing the descriptions and finding the correspondence between new features and old features in the map. In contrast, the direct approach directly uses an input image without any abstraction using handcrafted feature detectors and descriptors. In that case, IP is the work of comparing the whole input image with synthetic view images generated from the reconstructed map as can be seen in DTAM [52], or computing firstly the areas which have intensity gradient and then comparing with synthetic view images as in LSD-SLAM [53].

1.2.2 SLAM-core (back-end task)

In a SLAM-core, we have 3 basic modules: Initialization, Tracking and Mapping. To launch a vSLAM, it is necessary to define the coordinate system for camera pose estimation and 3D reconstruction in an unknown environment. Hence, in the initialization phase, the global coordinate system should first be determined, and a part of the environment is reconstructed as an initial map in the global coordinate system. After the initialization, tracking and mapping are performed to continuously estimate camera poses. In the tracking phase, the reconstructed map is tracked in the image to estimate the camera pose with

respect to the map. It should be noted that most of vSLAM algorithms assume that intrinsic camera parameters are calibrated beforehand so that they are known. Therefore, a camera pose is normally equivalent to extrinsic camera parameters with translation and rotation of the camera in the global coordinate system. In the mapping phase, the map is expanded by computing the 3D structure of the environment where the camera observes.

Moreover, the following two additional modules are also included in SLAM-core algorithms according to the purposes of applications: Relocalization and Global map optimization. The relocalization is required when the tracking is failed due to fast camera motion or some kidnapped robot problems. In this case, it is necessary to find out the camera pose with respect to the map again. Therefore, this process is called relocalization. If the relocalization module is not incorporated into vSLAM systems, the systems will not work anymore after the tracking lost and such systems are not useful in practice. Therefore, a fast and efficient method for the relocalization have been an attractive discussion in the literature. The other module is the global map optimization. The map generally includes accumulative estimation error according to the distance of camera movement. In order to have a converged map, the global map optimization is necessarily performed. In this process, the map is refined by considering the consistency of whole map information. When a map is revisited such that an old region is captured again after some camera movement, reference information that represents the accumulative error from the old position to the actual position can be computed. Then, a loop constraint from the reference information is used as a constraint to optimize the global map.

Loop closing is an indispensable technique to obtain the reference information. In the loop closing phase, a closed loop is first searched by matching the current frame with the previously acquired frames. If the loop is validated, it means that the camera revisited one of previously observed scenes. In this case, the accumulative error at the loop point occurred during camera movement can be estimated. We can note that the closed-loop detection phase can be done by using the similar techniques as in relocalization module. Basically, relocalization is done for recovering only a camera pose in the map while loop detection is done for obtaining geometrically consistent map.

1.3 Hardware systems based SLAM applications

As CPU, GPU and FPGA become employed in a wide range of applications, it has been admitted that each of these processing units (PUs) has its own features and strengths. Modern multicore CPUs use up to a few tens of cores, which are typically out-of-order, multi-instruction programming and support dynamic memory allocation. Moreover, CPU cores can operate at high frequency (up to 3-4 GHz) and use large sized caches to minimize the latency of memory access. In contrast, GPUs use much larger number of cores (a dozen or hundred cores), which are in-order and share their control unit. GPU cores run at lower frequency and smaller sized caches [54]. Thus, GPUs are suited for computing-critical applications but not for memory-critical applications. On the other hand, a FPGA serves to a hardware implementation of an application. A FPGA is a programmable dedicated processor, which is composed of programmable logic blocks and interconnect network with strong parallel processing ability. Multiple threads can be executed in a different logic and pipelined parallel processing. The most advantages of a FPGA are the very low power consumption and the data flow pipelining, so that it is suitable for streaming applications.

Due to the different characteristics of PUs, performing processing jointly between CPU, GPU or FPGA is recently a popular trend to achieve high performances. The platforms using this co-processors are referred as heterogeneous computing systems (HCSs). These HCSs can provide high computing for a much wider variety of applications and usage scenarios than using one kind of processing unit alone. Nowadays in HCSs , a CPU is indispensable and it is used as a host while GPU and FPGA are confined to act as accelerators.

1.3.1 Speeding up processing with CPU-GPU architectures

GPUs have been widely used in robotics applications, especially in computer vision. The scientific community has also exploited GPUs to speed up environment reconstruction or SLAM reconstruction algorithms in general. The proposed solutions are often heterogeneous where the CPU and the GPU cooperate together to execute the tasks of the

algorithm to accelerate. Michel [55] used a GPU to accelerate the tracking of 3D objects using cameras to achieve real-time performance when controlling a humanoid robot. Zhang et al. [56] proposed a method for accelerating the particle filter (FastSLAM) on a Nvidia GPU. The authors deported the calculation of particle weights on the GPU. Ma et al.[57] proposed a visual-inertial SLAM system able to operate in wide environments. They implemented the resulting algorithm on a high-end NVIDIA GPU, TITAN NVidia, and an Intel i7 quad-CPU desktop. Persson [58] presented a stereo visual odometry system implemented on CPU-GPU architecture. The localization accuracy was validated on KITTI dataset. However, although high-end GPU was employed for features matching, execution time reaches 145 ms/image which is not real-time performance for KITTI frame rate (100 ms/image).

As for the SLAM algorithms based on graph optimization, some researches has focused on the acceleration of bundle adjustment tasks on GPU. Bundle Adjustment is well-known in the field of vision. It consists to minimize the error between the actual observation and the predicted measurements (reprojection) of landmarks observed by one or more sensors. Solving this problem leads to a graph optimization problem. The bundle adjustment is often characterized by a very large number of landmarks in order to reconstruct mainly the map of the explored environment . To accelerate this reconstruction, Choudhary et al. [59] proposed a heterogeneous approach to distribute the calculations on CPU-GPU where the Hessian (information matrix) and the Schur complement are built on the GPU. Wu et al. [60] also presented a CPU-GPU partitioning for bundle adjustment. The authors used an iterative approach to solve the problem of least squares, namely the PCG (Preconditioned Conjugate Gradients). Rodriguez-Losada et al. [61] have parallelized an algorithm for building GPU occupation grids. The resolution of the system is ensured by an external library. Ratter et al [62] presented a GraphSLAM algorithm coupled to a busy grid. The authors refine the environmental map using a GPU.

Besides the classic calculators, in the last decade, the performances of GPU-based embedded architectures for mobile has grown very fast. This promotes their use in computer vision systems. Recently, researchers have focused on the optimization and performance evaluation of vision applications on mobile architectures. Nardi et al [63] proposed the SLAMBench. This is a framework that validates and evaluates the new implementations

of the KinectFusion (KF) algorithm [64]. KF makes it possible to reconstruct 3D scenes by means of a camera with depth such as Microsoft's Kinect. The SLAMBench aims to investigate compromises in time performance, accuracy and energy consumption. Architectures used by the authors include ODROID (XU3), Arndale and Tegra K1. The authors point out that the TK1 has achieved real-time performance with 22 frames/s. Zia et al [65] have extended the SLAMBench by adding an LSD-SLAM (Large-Scale Direct Monocular SLAM) algorithm [53]. In the same context, Backes et al [66] presented several optimizations concerning the implementation of KinectFusion on embedded architectures. Evaluations were done on ODROID (XU3) and Arndale.

1.3.2 CPU-FPGA architectures based systems design

The CPU-FPGA architecture has also drawn the attention of the scientific community to accelerate and design embedded SLAM systems. In most cases, the FPGA is used to speed up detection, features matching or matrix calculations. Bonato et al [67] designed a SLAM system based on EKF-SLAM using a Stratix (EP1S10F780C6) FPGA. The SLAM algorithm is run on a NIOS II instantiated on the FPGA. The authors announce a system capable of processing 30 frames/s in color and 60 frames/s in grayscale. Mingas et al [68] introduced the SMG-SLAM (Scan-Matching Genetic SLAM). The matching between the beams laser provided by a laser sensor (Laser Range Finder) is performed using a genetic algorithm. It was implanted on an FPGA. Cruz et al [69] implemented the update phase of EKF-SLAM on an FPGA. Tertei et al [70] presented a 3D visual SLAM system based on EKF-SLAM. The algorithm is fully implemented on a Zynq-7020 (ARM + FPGA) platform. To accelerate the processing, the authors deport the matrix calculation on the FPGA. The authors claim that their system is able to maintain and correct, at 30Hz, a map of 20 landmarks with an AHP (Anchored Homogeneous Point) parameterization. Gu et al [71] proposed a stereo camera based visual odometry system. The algorithm is implemented on a Stratix III EP3SL340 FPGA using a NIOS II as the master processor. The FPGA is primarily responsible for matrix calculation. The processing frequency of the system reaches 31 frames/s with a map of 30.000 landmarks. Nikolic et al [72] have proposed a visual odometry system for MAVs (Micro Air Vehicle). The system embeds an inertial unit with four cameras interfaced to a Zynq-7020 (ARM + FPGA)

platform. To improve the temporal performance, the image processing (e.g., the detection of points of interest) is provided by the FPGA. On the other hand, Sileshi et al. [73] have performed work to accelerate the particle filter SLAM (FastSLAM) based on FPGA. In a software/hardware approach, the authors distribute the tasks of the algorithm on an embedded processor (Microblaze) and an FPGA accelerator.

1.4 Conclusion

Visual SLAM have been widely studied in recent years thanks to many advantages of cameras. This chapter started with an overview of different sensor systems used in visual SLAM. Visual-only SLAM used only camera sensors for perception with two approaches: mono-camera and multi-camera. Mono-camera approach is easy for setup and calibration but has the issue of scale drift. Multi-camera approach can solve scale problem but has issues of calibration and sensors synchronization. In contrast, visual-Inertial or visual-Odometer approach employ the combination of camera with an other proprioceptive sensor in order to have more information to increase accuracy and robustness. Afterwards, a formalization of visual SLAM system was presented with two main parts: front-end (image processing) and SLAM kernel (back-end). Finally, we presented a bibliography on hardware architectures based SLAM applications. These heterogeneous architectures have become nowadays a basic design of embedded platforms. Hence, the implementation study using these architectures allows to attack not only real-time constraints but also the embeddability of SLAM algorithms on mobile applications. This thesis will focus on a proposal of a vision system based SLAM using heterogeneous architectures. The following chapter will present evaluation methodology including programming techniques, datasets and material platforms used to validate our proposed system.

Chapter 2

Evaluation Methodology

2.1 Methodology

2.1.1 Algorithm criteria

The work of this thesis focuses on a real-time SLAM system. The software (algorithm) and the hardware (architecture) are analyzed at the same time in order to have an accurate and real-time system. The criteria below are respected during the development of the SLAM algorithm:

- The SLAM algorithm must have a high localization precision.
- The memory requirement should be low so that the system could work with long trajectories.
- The algorithm must be suitable to be parallelized on heterogeneous architectures based embedded platforms.
- Despite the intention of working on a stereo camera sensor, the algorithm must be easily applicable to an other sensor-combination of visual SLAM.

2.1.2 Algorithm-Architecture mapping

Despite of the fact that heterogeneous platforms have the potential to offer better compromise between performance and energy, it is rather challenging to achieve this efficiency.

The main difficulty is the distinct characteristics of different types of hardware. In practice, a single application is often composed of widely differing computational tasks, which can be efficiently implemented on different types of processing units. Therefore, an effective use of the heterogeneous platform is a good mapping for each part of the application on the corresponding suitable hardware in order to minimize execution time, to maximize the system throughput and to make use of all computing resources.

For heterogeneous computing, the principal technique is to decompose an application into several functional blocks and match each block to the processing unit where the execution is optimal. In this manner, scheduling and mapping are two important factors to be considered in a heterogeneous system. The scheduling problem depends heavily on the topology of the task and the data dependency, representing the relations among the functional blocks. Otherwise, the mapping problem depends on the topology of the hardware system and the chosen performance criteria.

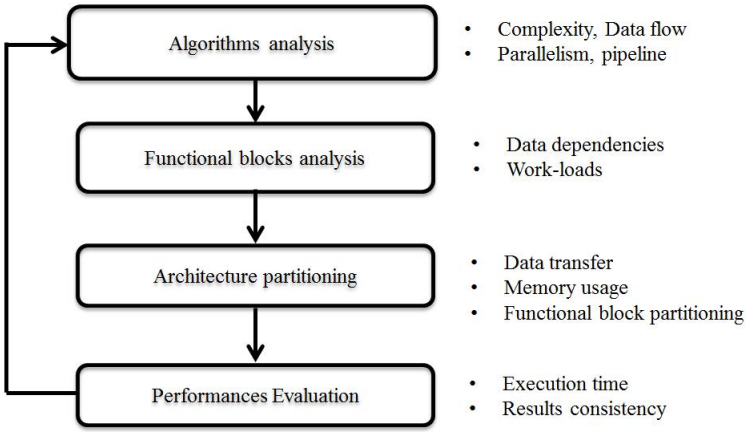


Figure 2.1: Algorithm-Architecture research methodology

This thesis employs the research methodology shown in figure 2.1. In the first step, we analyze the complexity, the data flow, the parallelization of the whole algorithm. Then, the algorithm is split into forms of functional blocks. We study data dependency and workload for each block in the second step. In the third step, we study data transfer and memory usage to determine which blocks are suitable for CPUs and which blocks are suitable for GPUs. Based on this study, we make a partitioning of the blocks on the architecture. Partitioning is followed in the last step by evaluation of execution time and

also the consistency of the algorithm. If the performance is not good, we return to the first step to re-study.

2.1.3 Programming techniques

2.1.3.1 CUDA programming

CUDA stands for "Compute Unified Device Architecture." It is a parallel computing platform developed by NVIDIA and introduced in 2006. It enables software programs to perform calculations using CPU-GPU heterogeneous architectures. By sharing the processing load with the GPU (instead of only using the CPU), CUDA-enabled programs can achieve significant increases in performance.

CUDA is one of the most widely used GPGPU (General-Purpose computation on Graphics Processing Units) platforms. It is proprietary and only runs on NVIDIA graphics hardware. CUDA can be used with several different programming languages. NVIDIA provides APIs and compilers for C and C++, Fortran, and Python. The CUDA Toolkit, a development environment for C/C++ developers, is available for many operating systems.

A CUDA program consists of a mixture of the following two parts: “the host code” runs on CPU and “the device code” runs on GPU. NVIDIA’s CUDA nvcc compiler separates the device code from the host code during the compilation process. In this thesis, the CUDA toolkit with C/C++ is used for GPU programming. The host code is standard C++ code and is further compiled with C++ compilers. The device code is written using CUDA C extended with keywords for labeling data-parallel functions, called kernels. The device code is further compiled by nvcc.

2.1.3.2 OpenCL programming

OpenCL (Open Computing Language) is a framework for writing programs that execute across heterogeneous platforms consisting of central processing units (CPUs) and several types of accelerators such as: graphics processing units (GPUs), digital signal processors (DSPs), field-programmable gate arrays (FPGAs) and other processors. OpenCL is developed by Khronos group with the motivation of offloading parallel computation to

accelerators in a common way, regardless of their underlying architecture (not restricted to GPUs as in CUDA).

Similar to CUDA, an OpenCL program also consists of the host code and device code. The programming language is based on the C99, using the Single Program Multiple Data (SPMD) model for parallel programming. Task parallelism is supported by launching multiple tasks as single-threaded kernels, performs parallelism on data vectors. However, there are slightly differences in memory notations and function interfaces between OpenCL and CUDA.

Although OpenCL programs can be compiled and linked into binary objects using conventional off-line compilation methodology, OpenCL on GPU also supports run-time compilation enabling OpenCL programs to run natively on the GPU hardware. It allows developers to implement rapidly OpenCL applications on various GPU hardwares without any need for recompilation of kernel codes.

In this thesis, OpenCL is employed for GPU programming on CPU-GPU architectures and also for FPGA high level synthesis on CPU-FPGA architectures.

2.2 Evaluation tools

2.2.1 Datasets based algorithm evaluation

We use six well-known datasets containing data from stereo camera. Among them, only New-College dataset is recorded by a robot, other datasets are from vehicle navigation as shown in figure 2.2.

2.2.1.1 KITTI dataset

KITTI dataset [74] was developed using the Annieway vehicle of Karlsruhe institute of technology. The main objective is to have novel challenging real-world computer vision benchmarks. The tasks of interest are: stereo vision, optical flow, visual odometry, 3D objects detection and 3D tracking. For this purpose, a standard station wagon is equipped with two high-resolution color and gray-scale video cameras. Data are captured by driving the vehicle around the mid-size city of Karlsruhe, in rural areas and on highways. The car

(a) KITTI and MRT (b) Oxford (c) Malaga

(d) St_Lucia (e) New College

Figure 2.2: Experimental Platforms

speed is up to 90 km/h. Up to 15 cars and 30 pedestrians are visible per image. The KITTI dataset for Odometry and SLAM applications consists of 22 sequences containing many moving objects existing in the scenes. Accurate ground truth is provided by a Velodyne laser scanner and a GPS localization system.

The cameras are mounted approximately at the level with the ground plane. The camera images are stored to a size of 1382x512 pixels using libdc's format 7 mode. After rectification, the images get slightly smaller with a resolution of 1234x376 pixels as shown in figure 2.3. The cameras are triggered at 10 frames per second by the laser scanner (when facing forward) with a shutter time adjusted dynamically (maximum shutter time: 2 ms).

2.2.1.2 Oxford RobotCar dataset

This dataset was achieved by the Oxford RobotCar platform, an autonomous Nissan LEAF, over the period of May 2014 to December 2015 [75]. The car traversed a route through central Oxford twice a week. This resulted in over 1000km of recorded driving

Figure 2.3: KITTI scenes

Figure 2.4: Oxford scenes

with almost 20 million images collected from cameras mounted onboard (1 Point Grey Bumblebee XB3, a trinocular stereo camera and 3 Point Grey Grasshopper2 (GS2-FW-14S5C-C) monocular camera). The car was also equipped with a NovAtel SPAN-CPT ALIGN inertial and GPS navigation system, providing the ground truth for trajectories.

Besides the dynamic urban environments, one of the most challenges of Oxford dataset is the presence of blur images in the scenes illustrated in figure 2.4.

2.2.1.3 Malaga dataset

Malaga dataset [76] was gathered entirely in Malaga urban scenarios with a car equipped with several sensors, including one stereo camera (Bumblebee2) and five laser scanners. One distinctive feature of this dataset is the existence of high-resolution stereo images grabbed at high rate (20fps). The challenge is that images are captured with many sky region which provides unreliable features. Moreover, there is a huge variation of image brightness during experiments as shown in figure 2.5.

Figure 2.5: Malaga scenes

Figure 2.6: MRT scenes

2.2.1.4 MRT dataset

MRT [77] is also a dataset recorded by Karlsruhe institute. It preceded the KITTI dataset. This dataset contains data of 3 sensors (3D Lidar Scanner, a calibrated Stereo Camera and a GPS/IMU). The trajectory consists of driving a loop, the car passes both below and over a bridge. Unlike KITTI, MRT provides only distorted images so that the rectification is done by user. The image quality of MRT (shown in figure 2.6) is also lower than that of KITTI.

2.2.1.5 St Lucia dataset

The UQ St Lucia Dataset [78] is a vision dataset gathered by a car driven in a 9.5km circuit around the University of Queensland's St Lucia campus. The data consists of visual data (figure 2.7) of a calibrated stereo pair camera, translation and orientation information as a ground truth from an XSens Mti-g INS/GPS and additional information from a USB NMEA GPS. The car traverses local roads and encounters a number of varying scenarios

Figure 2.7: St_Lucia scenes

Figure 2.8: New College scenes

including roadworks, speed bumps, bright scenes, dark scenes, reverse traverses, a number of loop closure events, multi-lane roads and roundabouts with speeds of up to 60 km/h.

St Lucia contains a very large loop closure which allows us to verify the functionality of loop detection after a long-term tracking and many new places added to the map.

2.2.1.6 New College dataset

The NewCollege dataset [79] is recorded by a stereo camera at 20 fps and a resolution of 512x382 pixels from a robot traversing 2.2 km through a campus and adjacent parks. Stereo images (figure 2.8) are captured at 20Hz. Images need to be rectified by a tool provided in the project before launching a SLAM algorithm. The trajectory includes several loops and fast rotations.

2.2.2 Platforms based algorithm implementation

2.2.2.1 Work station PC

A work station PC acts as a representation of “discrete CPU-GPU system”. The platform used in this thesis provides a mighty CPU integrating 8 cores i7 running at 3.4 GHz. The

CPU architecture optimizes memory access by offering 8MB smart cache that allows all cores to dynamically share access to the last level cache. The main memory (RAM) is 16 GB. This platform also integrates an NVIDIA GT-740 GPU as an accelerator with 384 Shader cores, 2GB global memory and 28.8 GB/s memory interface bandwidth. The GPU programming supports CUDA and OpenCL. Average power consumption is about 84W for CPU and 64W for GPU.

2.2.2.2 Nvidia Jetson Tegra X1

For embedded applications in visual computing, NVIDIA introduced Jetson Tegra X1 (TX1), a small form-factor Linux system-on-module shown in Figure 2.9. This module is based on system-on-chip processor TX1 using ARM's Cortex with a cluster of 4 high performance A57 big cores and a cluster of 4 high efficiency A53 little cores. However, only one cluster could be activated at a time. The A57 CPU cluster operates at 1.9 GHz, has 2MB of L2 cache shared by the four cores with 48KB L1 instruction cache and 32KB L1 data cache per core. The A53 CPU cluster operates at 1.3 GHz, with 512KB of L2 cache shared by four cores, 32KB instruction and also 32 KB data L1 cache per core. The GPU of TX1 is designed using Maxwell architecture, includes 256 Shader cores and clocks at up to 1GHz. GPU memory interface offers a maximum bandwidth of 25.6 GB/s with the capacity of 2GB global memory. Jetson TX1 draws around 8-10 watts under typical CUDA load, and up to 15 watts when the module is fully utilized. GPU programming supports only CUDA (no OpenCL).

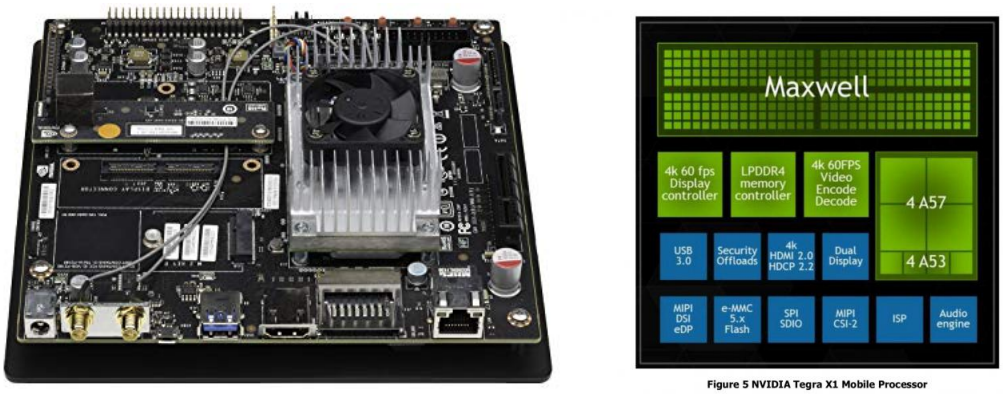


Figure 2.9: Jetson TX1 platform

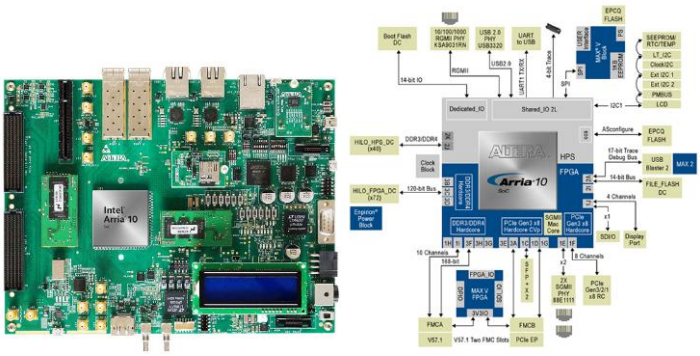


Figure 2.10: Arria 10 SoC

TX1 is particularly aimed for developers working in computer vision, deep learning, robotics and related fields. In this thesis, it is used for evaluating the performance of the proposed SLAM algorithm on an embedded system.

2.2.2.3 Altera Arria 10 SoC

Recently, Altera presented Arria 10 SoC which has been designed to meet the performance and power requirements for mid-range embedded applications. As shown in figure 2.10, the Arria 10 SoC features an ARM dual-core Cortex-A9 MPCore (1.5 GHz), up to 660 KLEs of advanced low-power FPGA logic elements, 1GB DDR4 HILO memory card for CPU and also 1 GB DDR4 HILO memory card for FPGA. It combines the flexibility and ease of programming of a CPU with the configurability and parallel processing power of an FPGA. This system corresponds to the highest coupled in CPU-FPGA heterogeneous when CPU and FPGA are integrated on chip.

In this thesis, the implementation of feature extraction on Arria 10 SoC is studied. It is the first step in embedding the visual SLAM algorithm on a reconfigurable architecture. Unlike CPU or GPU, power consumption of FPGA depends heavily on resources used in the design. The more computing resources are used, the higher processing speed is achieved but the more power is consumed. Hence, the challenge of designing using a FPGA is to achieve a compromise between computation speed and power consumption.

2.3 Conclusion

This chapter presented the methodology and the evaluation tools used in this thesis for system development and validation. We will follow a hardware software co-design approach to develop different systems. In terms of programming techniques, since CUDA is supported by only NVIDIA GPUs, that has been integrated in several devices from high-end computing to embedded platforms, we chose OpenCL that allows implementing systems on GPUs from other companies or on other kind of heterogeneous architectures (CPU-FPGA).

Six datasets (KITTI, Oxford, Malaga, MRT, St Lucia, New College) with many challenging scenes will allow an extensive evaluation about accuracy and robustness of our proposal. Besides, timing performance will be analyzed on three different heterogeneous architectures (high-end CPU-GPU, embedded device Nvidia Tegra X1 and CPU-FPGA Altera Arria 10 SoC).

We will start our study on the front-end task of the visual SLAM algorithm. Our work is based on feature based approach so the front-end task will allow detecting and matching correspondences between images. Our objective is to propose a feature extraction algorithm to achieve a robust matching result. It will be discussed in the next chapter.

Chapter 3

HOOFR: a bio-inspired feature extractor

Feature matching is the task of establishing the correspondences between two images of the same scene. In SLAM application, the stability of the matching result is very important to obtain a good localization result. To realize the matching, features need to be firstly detected and described. The detection algorithm must have a high repeatability in order that many same points can be found in both two images. Besides, the description algorithm must contain distinctive information among features to ensure an accurate matching. This chapter introduces the study on feature matching and the proposed algorithm denoted as Hessian ORB and Overlapped FREAK (HOOFR).

3.1 Overview

Through over a decade old, the most popular feature extraction algorithm was Scale Invariance Feature Transform (SIFT) proposed by Lowe[80]. SIFT identifies keypoints based on the local extremum of Different of Gaussian (DoG) over scale space and describes them by a 3D spectral histogram of the image gradients. SIFT is remarkably successful in object recognition [80], visual mapping [33], etc. However, it is affected by high computation requirements, which prohibit its implementation in real-time applications such as visual odometry, or on low-power embedded devices such as mobile phones. An alternative named Speed Up Robust Feature (SURF) was proposed in [81].

This method relies on the determinant of the Hessian matrix for keypoint detection and on the responses of Haar-like filters for the description. SURF has a comparable performance to SIFT but it exhibits a significant improvement in computation speed. The reason is that while SIFT approximates Laplacian of Gaussian (LoG) by DoG, SURF goes further and approximates LoG by box filters. By relying on an integral image, the box filter convolution may be performed efficiently. Then, two sets of SIFT or SURF keypoints may be matched by employing Euclidean floating distances among descriptors.

On the other end of the spectrum, to address real-time applications, ORB[82] uses a binary representation in order to simplify the calculation. ORB is inspired by the FAST [83] keypoint detector and by the BRIEF [84] descriptor. In fact, FAST does not provide neither multi-scale features nor orientation measurement. Therefore, in ORB the authors employs a scale pyramid representation and detect FAST features at each level; additionally, the keypoint orientation is estimated using the local intensity centroid. The ORB descriptor is then constructed based on rotated BRIEF which uses simple binary tests between pixels in a smoothed image patch. ORB algorithm offers a high efficiency to be implemented in patch-tracking application on smart phone [82] or SLAM application [85], etc.

Similar to BRIEF, there are several other variants of binary descriptors, among which BRISK[86] and FREAK [87] could be named as candidates. A clear advantage of binary descriptors is that the Hamming binary distance may replace the Euclidean floating distance for matching, by using bit-wise XOR followed by a bit count on specific architectures, which is significantly faster. The key concept of the BRISK descriptor is the use of a symmetrical pattern. Instead of random points as in BRIEF, sampling points of BRISK are located on circles concentric to the keypoint. Furthermore, BRISK divides sampling-point pairs into two subsets: *long-distance pairs* reserved to compute keypoint orientation and *short-distance pairs* reserved to build keypoint descriptor. Following this idea, FREAK is an optimized version of BRISK with two main modifications. Firstly, it uses a sampling pattern inspired from the human retina where the smoothing kernels are overlapping and their size exhibit exponential change. Secondly, it uses 45 symmetrical pairs with respect to the center to estimate keypoint orientation rather than using the *long-distance pairs* subset as in BRISK.

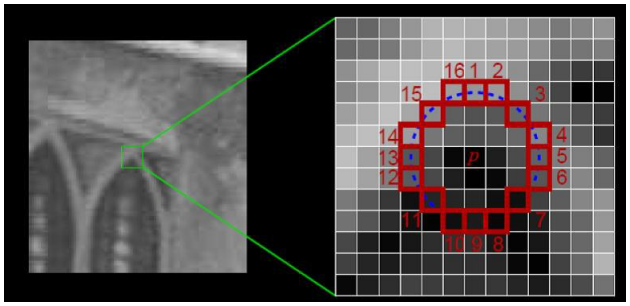


Figure 3.1: FAST Bresenham circle

3.2 FAST-9 detection

FAST (Features from Accelerated Segment Test) is an algorithm proposed originally by Rosten and Drummond [83] for extracting interest points, keypoints or features (all three are interchangeably used in literature) in an image. FAST is aimed to use in real-time frame rate application so that it is designed to have a low computational cost. It consider the points on a circular ring around one pixel. In case of enough consecutive pixels on the ring which are brighter or darker than the central pixel with a threshold t, this central pixel is recognized as an interest point. The algorithm is explained in detail below:

- Considering a pixel P in the image. The intensity of this pixel is defined as I_P.
- Set a threshold t.
- Select a circle of 16 pixels surrounding the pixel P (Rosten proposed to used Bresenham circle of radius 3 as shown in figure 3.1).
- P is an interest point if N contiguous pixels out of 16 are either above or below I_p by the value t.

The value of N is generally set between 9 and 12 depending on the application where, as in SLAM algorithms ([85] or [88]), the value of 9 (FAST-9) presented a good performance. The reason behind the high speed of FAST is the segment test. Firstly, the comparison is made for the pixels 1, 5, 9, 13 of the circle with P. As evident, at least 3 contiguous pixels ($N = 12$) or 2 contiguous pixels ($N < 12$) should satisfy the threshold criterion so that the interest point will exist. In contrast, the pixel P is not a possible interest point and the process is terminated immediately. The test could be repeated on the sets of pixels

(2,6,10,14), (3,7,11,15) and (4,8,12,16) for a rapid rejection. The majority of pixels in image are rejected during segment tests. In the case that all the sets pass the segment test, the final examination is performed to determine whether if P is really an interest point.

3.3 Hessian filtering

Despite of the high speed detection, FAST provides a significant number of features. In SLAM application, it becomes a disadvantage. A huge number of features could not increase the precision but makes algorithm more computational cost. Hence, an additional criterion is taken into account to filter the FAST features. In ORB[82], the author used score extracted from Harris matrix as feature score. He computed Harris score for all features returned by FAST. Then, the relevant points having the highest Harris response are maintained. In our work, we were inspired by the overall results of Mikolajczyk et al. [89] who evaluated different detection methods. We were interested in their conclusion that in general, the Hessian based detection overcomes that based on Harris. Therefore, we propose to employ Hessian score in the detection to keep the relevant features.

$$H = \begin{bmatrix} \frac{\partial^2 I}{\partial x^2} & \frac{\partial^2 I}{\partial x \partial y} \\ \frac{\partial^2 I}{\partial x \partial y} & \frac{\partial^2 I}{\partial y^2} \end{bmatrix} \tag{3.1}$$

Hessian matrix (as shown in equation 3.1) consists of the second order partial derivatives of the image. The eigenvectors of this matrix form an orthogonal basis highlighting the local direction of the gradient. If the product of eigenvalues of the Hessian matrix is positive, a local extremum is present. We note that for any square matrix, the product of eigenvalues is the determinant of the matrix. Another detector relying on this determinant with remarkable results is SURF [81]; therefore, we use the determinant of the Hessian matrix as the score of features.

In practice, in order to find the derivative, the image is first smoothed and then the numerical approximations are applied as this operation is sensitive to noise. Nevertheless, instead of employing a filter to smooth the image and then finding its derivative, the derivative can be directly applied to the smoothing function which can then be used to filter the image. In our work, we smooth image by Gaussian function (equation 3.2)

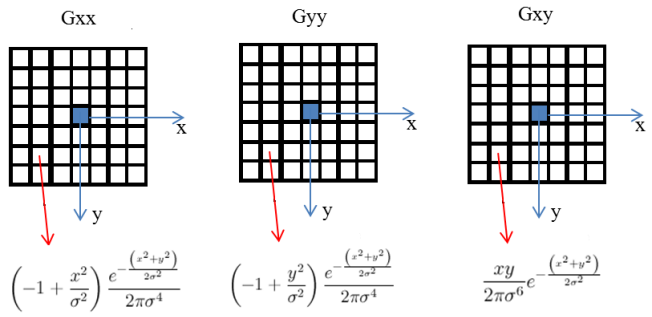


Figure 3.2: Square filters for calculating the Hessian matrix in HOOFR

where its derivatives are shown in equations 3.3-3.5. For each candidate point returned by FAST, we calculate its Hessian matrix. Each element of this matrix is generated by applying a square filter with the dimension of 7x7 shown in figure 3.2. corresponding to the second order derivative of the smoothing function. Then, the determinant of this matrix is considered as the score of the point. If there are more than K points detected by FAST, we only maintain the K points exhibiting the highest score.

$$G(x,y,\sigma) = \frac{1}{2\pi\sigma^2} exp(-\frac{(x^2+y^2)}{2\sigma^2}) \tag{3.2}$$

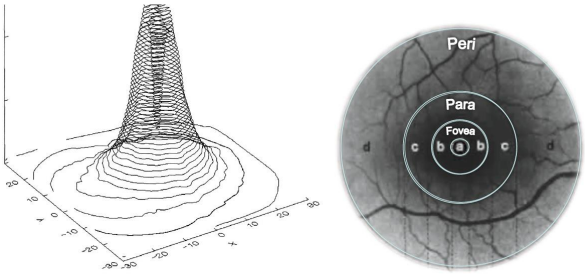
$$\frac{\partial^2 G}{\partial x^2} = (-1+\frac{x^2}{\sigma^2})\frac{exp(-\frac{(x^2+y^2)}{2\sigma^2})}{2\pi\sigma^4} \tag{3.3}$$

$$\frac{\partial^2 G}{\partial y^2} = (-1+\frac{y^2}{\sigma^2})\frac{exp(-\frac{(x^2+y^2)}{2\sigma^2})}{2\pi\sigma^4} \tag{3.4}$$

$$\frac{\partial^2 G}{\partial x \partial y} = \frac{xy}{2\pi\sigma^6} exp(-\frac{(x^2+y^2)}{2\sigma^2}) \tag{3.5}$$

3.4 Overlapped FREAK bio-inspired description

FREAK was proposed in [87] by considering human retina topology and neuroscience observations. It is believed that human retina extracts information from the visual field by using the Gaussian comparison (Difference of Gaussian) of various sizes and by encoding these differences in binary mode as a neural network.



(a) Density of ganglion cells over the retina (b) Retina areas

Figure 3.3: Distribution of ganglion cells over the retina. There are four areas of the density: (a) foveal, (b) fovea, (c) parafoveal and (d) perifoveal

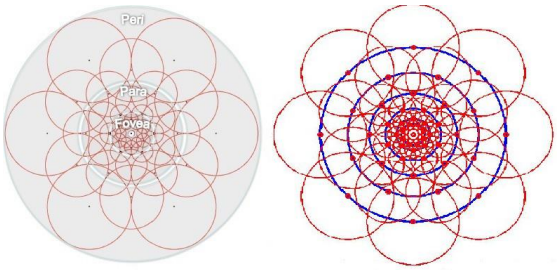


Figure 3.4: Sampling pattern in FREAK (a) and in HOOFR(b)

3.4.1 Description sampling pattern

The topology and spatial encoding of the retina is interesting. First, a ganglion cell includes several photoreceptors. The region where light influences the response of a ganglion cell is the receptive field. Figure 3.3 shows that the spatial distribution of ganglion cells reduces exponentially with the distance to the foveal. They are segmented into four areas: foveal, fovea, parafoveal, and perifoveal. Furthermore, the sizes of the receptive field and dendritic field increase with the radial distance to the foveal.

Inspired by this idea, the authors of [87] proposed a sampling pattern as showed in Figure 3.4a. The pattern is composed of 7 concentric circles with exponentially decreasing radius. Each circle contains 6 points considered as 6 receptive fields, and the receptive field at the center, so that the overall pattern is formed by 43 receptive fields. The distribution of the points on the concentric circles is similar to the method of 6-segments presented in DAISY [90].

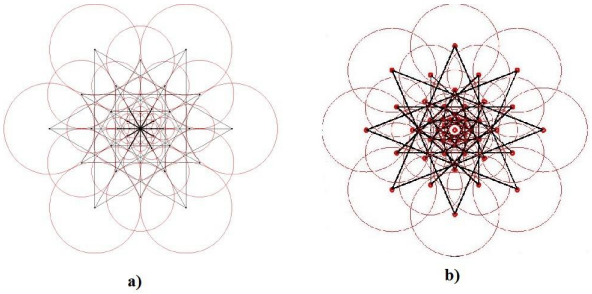


Figure 3.5: Illustration of selected pairs to estimate the orientation in FREAK (a) and HOOFR(b)

With HOOFR, we propose a different sampling pattern illustrated in figure 3.4b. Our sampling pattern contains only 6 concentric circles. However, each circle has 8 receptive fields distributed as the 8-segment method in DAISY. Therefore, including the point at the center, this pattern contains 49 receptive fields in total. The justification for our proposed configuration is that for complex image processing tasks, various descriptors exploit, either in the image space [91] or in the frequency domain[92], a certain degree of overlapping in order to be able to grasp more effectively complex correlations. With respect to FREAK, our configuration increases, in addition to the radial overlap, the amount of circumferential overlap among the fields.

Due to the fact that FREAK uses the comparison between these receptive fields to build the descriptor, with 49 fields, we have more pairs (1176 pairs) to choose than that of [87] (903 pairs). Moreover, in our sampling pattern, we have the overlap not only between the receptive fields of different concentric circles but also circumferentially.

3.4.2 Keypoint orientation in HOOFR

In order to estimate the keypoint orientation, we use the same method proposed in FREAK by summing the local gradients over selected pairs. However, our sampling pattern has more overlapping leading to more information being integrated in the receptive field. Hence, we can use fewer pairs than FREAK for orientation estimation. The latter is using 45 pairs with symmetric receptive fields with respect to the center as shown in figure 3.5a, whereas, in HOOFR, we select only 40 pairs as shown in figure 3.5b. By decreasing the number of pairs, we can improve the execution time when computing the orientation.

The orientation is then obtained by the equation 3 where S is the set of all 40 pairs used to compute local gradients, N is the number of pairs in S and $P_0^{r_1}$is the 2D vector of coordinates of the receptive field center. The space of orientation in HOOFR is also discretized by the same steps proposed in FREAK.

$$O = \frac{1}{N} \sum_{P_0 \varepsilon S} (I(P_0^{r_1}) - I(P_0^{r_2})) \frac{P_0^{r_1} - P_0^{r_2}}{\left\| P_0^{r_1} - P_0^{r_2} \right\|} \tag{3.6}$$

3.4.3 HOOFR Descriptor

The binary descriptor F is constructed by the comparison between receptive fields with their corresponding Gaussian kernel.

$$F = \sum_{0 \leq n < N} 2^n T(P_n) \tag{3.7}$$

$$T(P_n) = \{ \begin{array}{ll} 1 & if \quad (I(P_n^{r_1}) - I(P_n^{r_2})) > 0 \\ 0 & otherwise \end{array} \tag{3.8}$$

where P_n is the pair of receptive fields, N the size of the binary descriptor, $I(P_n^{r_1})$ and $I(P_n^{r_2})$ are respectively the Gaussian smoothed intensities of the first and the second receptive field of the pair n.

Here, we experience a second advantage of the increase in overlap, the fact that it contributes to reduce the descriptor size. In HOOFR, we build a descriptor of size 256 bits which is half the size of the FREAK descriptor (512 bits). This reduction is aimed not only at memory-saving, but also at accelerating the matching process where the 256-bits comparison is two times faster than 512-bits comparison. In fact, following testing, we found that a 256-bits descriptor is high enough to ensure a good performance for our sampling pattern. This boils down to selecte the 256 most relevant pairs among the total of 1176 pairs. These pairs are also chosen experimentally by running an algorithm similar to the ORB selection. This algorithm has 3 main steps:

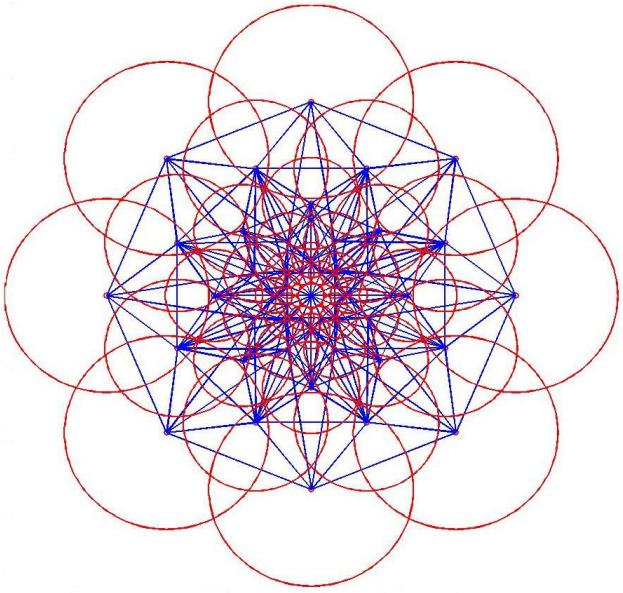


Figure 3.6: Illustration of 256 selected pairs used to construct the descriptor in HOOFR

- The first step extracts keypoints from training data. We take all the possible pairs (1176 pairs) to build the description and each keypoint has its own descriptor. A matrix M is created where the number of rows corresponds to the number of keypoints and the number of columns corresponds to the size of descriptor (1176 columns).
- For each column, we calculate the average which is situated between 0 and 1. This value represents the variance of the binary distribution. The high variance is desired to have a discriminant feature and the mean of 0.5 leads to the highest variance.
- All the columns are ordered and we keep the 256 columns which have the highest variances.

Figure 3.6 shows the 256 relevant selected pairs used in HOOFR.

3.5 HOOFR performance evaluation

Our proposed algorithm has been tested using the well-known evaluation method and datasets published by Mikolajczyk and Schmid[93]. We take eight image sequences as shown in figure 3.7, corresponding to viewpoint change (Graffiti, Wall), zoom and rotation (Bark, Boat), blur (Bikes, Trees), brightness change (Cars) and JPEG compression (Ubc) to evaluate the performances.

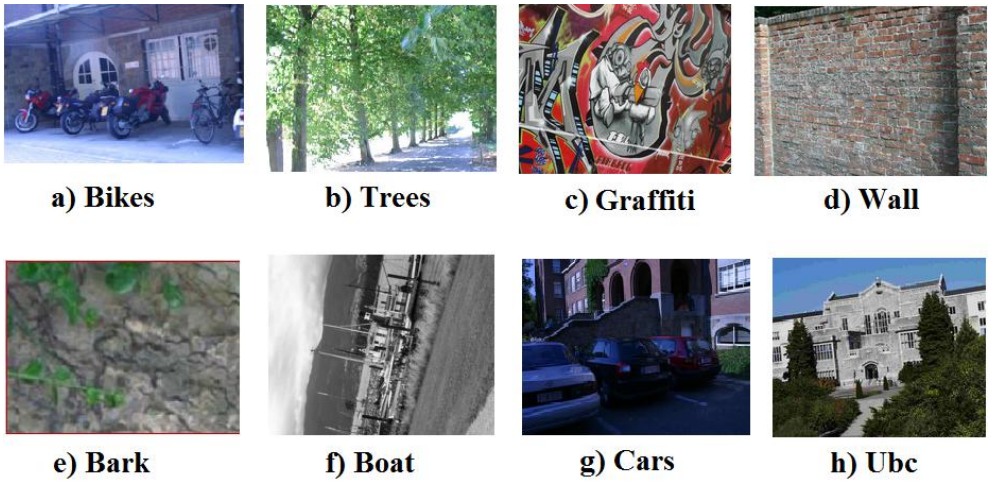


Figure 3.7: Image sequences used for evaluation

Each sequence contains 6 images ordered by the increasing amount of transformation from image 1 to image 6. All transformations are planar, so ground truth is determined based on the homography matrix. Furthermore, matching is performed between each image and the first image of the same sequence because homography matrices for these pairs of images are carefully defined in the datasets. We consider that a point p_a in one image is a correspondence of a point p_b in other image when they satisfy two conditions:

- The error in relative location of $||p_a - H \cdot p_b|| < 1.5$ pixel, where H is the homography matrix between the two images.
- The overlap area of the keypoint region in one image and the projection of the keypoint region from the other image is high enough. In our test, if the intersection is larger than 50% of the union of the two region, it is considered a correspondence.

We note that this correspondence is called point-to-point correspondence as defined in [93]. It is different from region-to region correspondence as defined in [89] which considers only the second condition above. We take other widely used algorithms such as SIFT, SURF, ORB, BRISK and FREAK to make the comparison. All matching tests employ brute-force algorithm using floating distance for SIFT, SURF and Hamming distance for binary descriptors. For the sake of fairness, we set the same value for the number of relevant keypoints returned by detectors. This value is set to be 1000 keypoints in this test. As a reminder, the SIFT detector selects the relevant keypoints based on contrast thresholds and edge filter thresholds[80], whereas SURF uses Hessian response, ORB

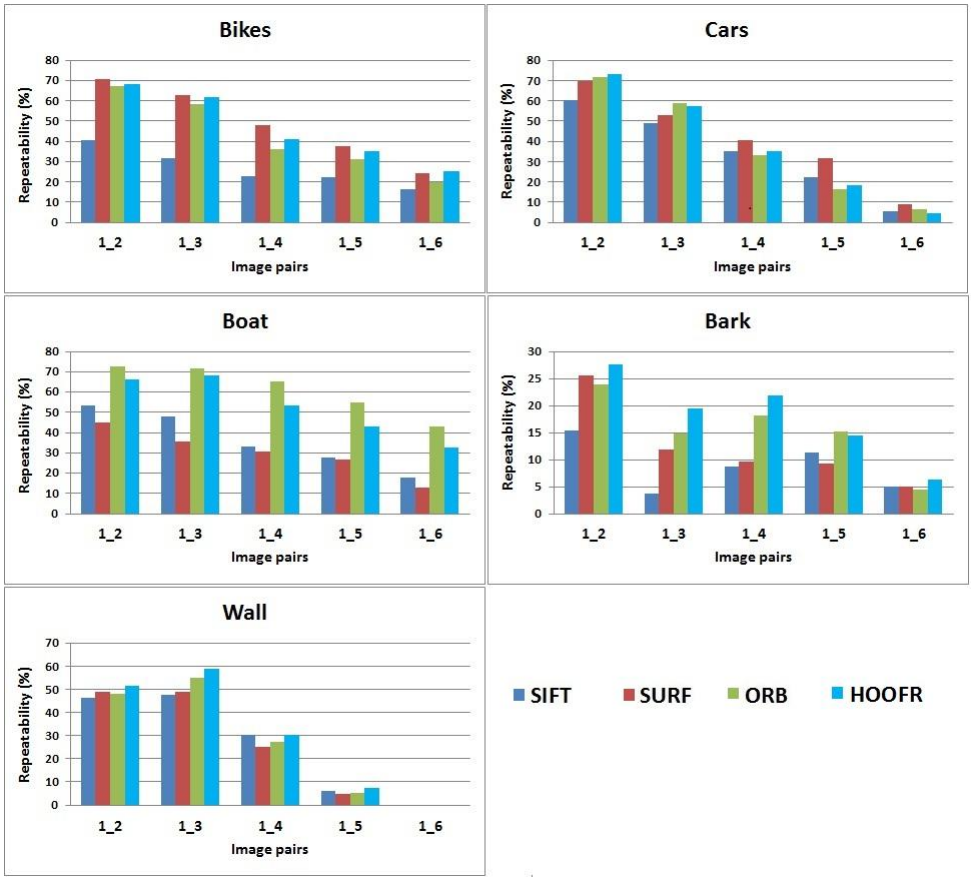


Figure 3.8: Repeatability of detectors evaluated in image datasets.

uses FAST score then Harris score. On the other hand, our algorithm uses FAST score then Hessian score to refine keypoints for the detection.

3.5.1 HOOFR detector repeatability

The desirable property for a feature detector is the repeatability. It represents the ability of a detector to find the same feature in two or more different images of the same scene. It is defined in [93] as the ratio between the number of corresponding keypoints and the minimum number of points detected in the two images. We note that the number of points here is fixed to be 1000 for all detectors.

Figure 3.8 shows the repeatability evaluation on five transformations with independent characteristics. HOOFR exhibits a remarkable performance, and outperforms ORB on most of image sequences. This result underlines the conclusion of [89] that in general, Hessian matrix based detection outperforms detection based on the Harris matrix. The

occasional low performance of SIFT is due in part to its sensitivity to rotation change and to blur (Boat, Bark and Bikes sequences); SURF exhibits competitive performance with respect to ORB and our algorithm HOOFR. Nevertheless, SURF is also time-consuming which limits its ability to be applied in real-time applications.

3.5.2 HOOFR binary descriptor comparison

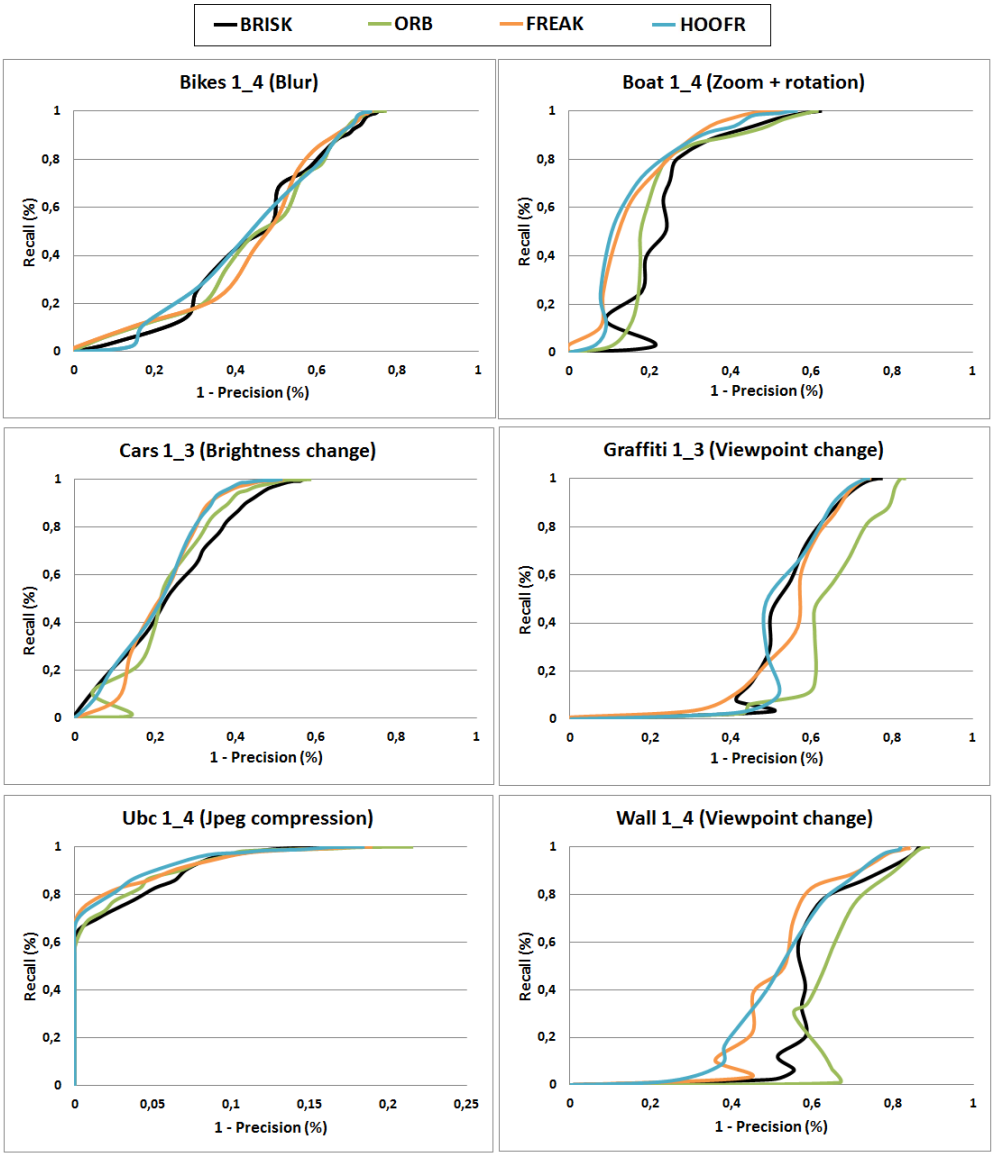


Figure 3.9: Recall-precision for the evaluation of binary descriptors

Since we use the binary method to build the description, we compare HOOFR descriptor with other binary descriptor in the literature such as BRISK, ORB and FREAK. Recall

vs 1-precision curve is used as proposed in FREAK [87] and BRISK [86] to judge the performances. Recall is defined as the ratio of number of correct matches/number of correspondences, while 1-precision is the ratio of number of false matches/number of matches. In fact, the result of matching largely depends on the combination detector-descriptor. Nevertheless, the global ranking of matching performance of the descriptors remains the same regardless of the selected detector. Therefore, to ensure a fair comparison, we evaluate all descriptors by using the same detector. In this test, we chose ORB detector and the number of relevant keypoints returned is also 1000.

Figure 3.9 shows the recall-precision curves using thresholds based similarity matching of Hamming distance for a collection of images pairs from datasets. As confirmed in figure 3.9, HOOFR is generally more robust than FREAK. On the other hand, it overcomes ORB for all the tested image transformations. Moreover, despite the fluctuation in some cases, HOOFR has better performance than BRISK.

3.5.3 Overall evaluation of HOOFR extraction

Our work proposes modifications in terms of detection and description at the same time, so we also evaluate the joint performance of both propositions compared to the well-known algorithms which have their own detector and descriptor such as SIFT, SURF or ORB. Due to the fact that SIFT and SURF use the floating descriptor while ORB and our work use binary descriptor, it does not make sense to use a similarity based method in matching. The reason is that similarity method highly depend on the threshold and it is difficult to determine equivalent value for each type of the descriptor. Therefore, in order to match two set of keypoints extracted from two images, for each keypoint in the first set, we simply select the keypoint in the second set which is the nearest neighbor (smallest matching distance). We present a factor called “Matching rate”(number of correspondences / number of matches) to compare the performances in this case.

In order to have a high matching score, an algorithm must exhibit a high detector repeatability and must concurrently have a high discrimination for the keypoint descriptor. As illustrated in figure 3.10, HOOFR performs competitively with SURF. It outperforms SURF for the viewpoint change (Wall, Graffiti) or JPEG compression (Ubc), has a fluctuation for zoom-rotation (Bark, Boat) or blur (Bikes, Trees) and slightly falls behind

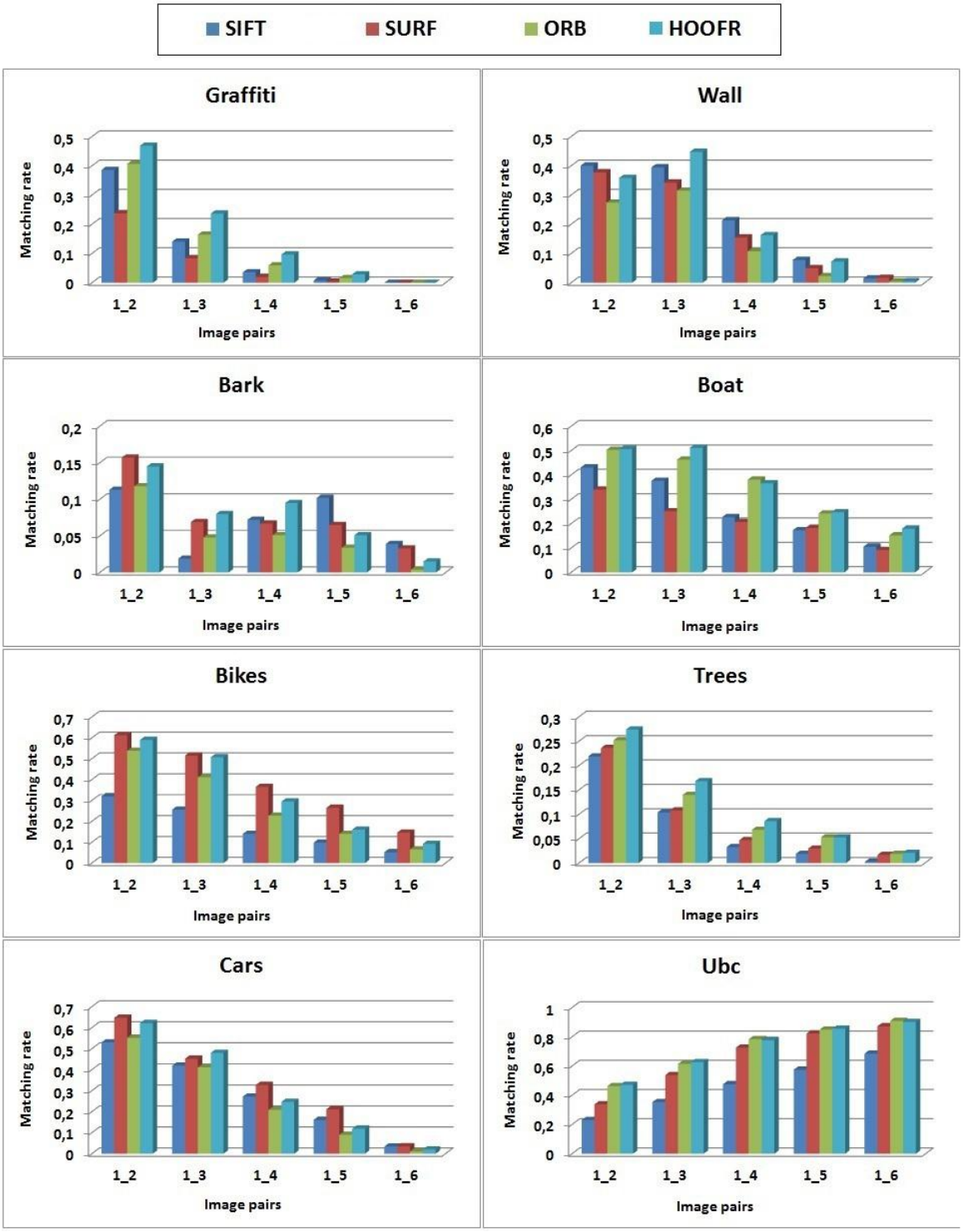


Figure 3.10: Evaluation of matching rate in image datasets

Table 3.1: Detection time (milliseconds) of different detectors (1000 relevant keypoints returned)

	Bark_1 (512x765)	Graffiti_1 (640x800)	Boat_1 (680x850)	Wall_1 (700x1000)
SIFT	860	919	1554	1722
SURF	129	137	169	202
ORB	34	44	79	107
HOOFR	33	42	76	105

Table 3.2: Description time (milliseconds) of different descriptors for 1000 keypoints

	Bark_1 (512x765)	Graffiti_1 (640x800)	Boat_1 (680x850)	Wall_1 (700x1000)
SIFT	3611	3873	4024	4093
SURF	479	488	492	501
ORB	16	18	18	20
BRISK	23	24	24	24
FREAK	20	21	21	21
HOOFR	18	20	20	20

SURF for brightness change (Cars). In contrast, HOOFR normally has overall better performance than SIFT and ORB.

3.5.4 Timings

Execution times have been recorded using a single core on a PC with Intel Core i7 3.4 GHz processor and 16GB RAM. Operating system is Window 8.1. Table 3.1 presents the results corresponding to detection of the first image in 4 selected sequences, while table 3.2 presents the description time for the same images. Moreover, table 3.3 shows the extraction time (detection+description) of the algorithms having its own detector and descriptor. The values are averaged over 50 runs.

Regarding the detector, the timings show an advantage of HOOFR. Its computation is even faster than ORB detector although the latter is the fastest detector currently available. The reason is that the Hessian response is time-saving to compute against Harris response.

In terms of description, we also clearly highlight the advantage of binary descriptors, with an order of magnitude faster than SURF and two order of magnitude faster than SIFT. Among the binary descriptors, FREAK is inspired by BRISK and it is more efficient

than BRISK. Following the optimization trend, HOOFR is inspired by FREAK, and it is more robust, memory-saving and slightly faster than the original. We note that although the descriptor size and the number of pairs for orientation estimation were reduced in HOOFR in comparison to FREAK, we can not gain a significant acceleration due to more receptive fields being sampled (49 points) than in the case of FREAK (43 points). Hence, for each keypoint description, HOOFR takes more time to compute the Gaussian filter for all receptive fields. However, even though ORB is the fastest descriptor, in general, the extraction time (detection + description) of ORB is similar to that of HOOFR while our proposal maintains the better matching results.

Table 3.3: Extraction time (milliseconds) of different algorithms (detection + description) for 1000 relevant keypoints returned

	Bark_1 (512x765)	Gratifi_1 (640x800)	Boat_1 (680x850)	Wall_1 (700x1000)
SIFT	4471	4792	5578	5815
SURF	608	625	661	703
ORB	50	62	97	127
HOOFR	51	62	96	125

3.5.5 HOOFR features validation in object tracking application

In this part, HOOFR is integrated in object-tracking application. We apply the method based Homography matrix as used in many researches in literature. Two experiments were conducted to evaluate tracking performance: one is multi-objects tracking in the same image, the second is object-tracking in video frame. The latter is implemented on embedded system to evaluate the time constraint. Our conventional pipeline to track an object in an image is:

- We firstly detect and describe HOOFR features points of the reference object image.
- For each image frame, we also detect and describe HOOFR features points in the image and match them to the features points of reference object image by brute-force matching.
- Homography matrix (H-matrix) is then estimated based on the matching result using RANSAC algorithm [94].

Figure 3.11: Multi-objects tracking

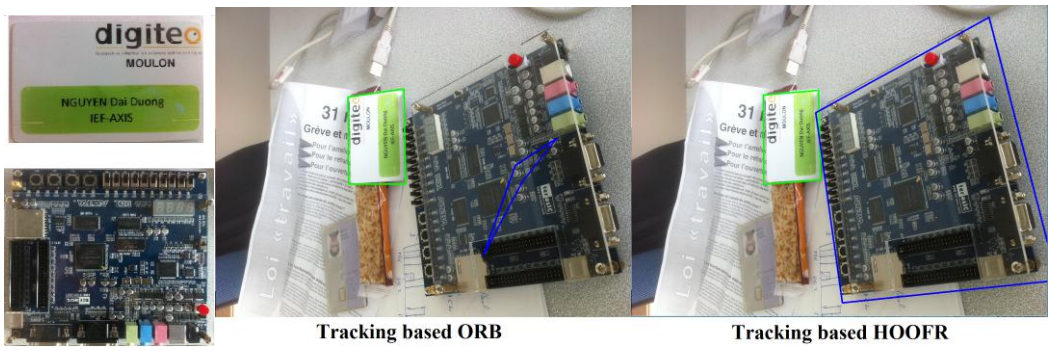


Figure 3.12: Multi-objects tracking

3.5.5.1 Multi-objects tracking

Figure 3.11 shows the result of multi-objects tracking. We consider three objects: a student card, a postcard and a magazine which are the three reference images presented on the left side. The picture on the right side is an experimental scene where these objects are situated together with other items. We extract 1000 keypoints for each reference image while 9000 keypoints are extracted on the image scene. Due to that image scene is in high resolution (1296x968 pixels), a large number of keypoints is necessary for this image in order to have enough matching for all objects. As the result, even though the magazine is covered partially by the student card and the postcard, the three articles are recognized perfectly in the scene.

Figure 3.12 is the comparison between ORB based tracking based and HOOFR based tracking. We employ the same conventional pipeline, the same number of relevant keypoints and all the other parameters in homography estimation. The student card can be localized by the 2 extractors but the FPGA card is found only by HOOFR. This demonstrates that HOOFR provides more relevant keypoints than ORB for objects tracking.

Figure 3.13: Tracking results of the postcard in a video sequence

3.5.5.2 Embedded objects tracking in video

We take into account a video recorded by a smart-phone giving a sequence of images (568x320 pixels). The postcard is presented in the video and its position is aimed to be tracked. We apply the same conventional pipeline as described above for postcard-tracking in each video frame. This experimental test is realized on the ODROID-XU4 card which contains 8 cores ARM processor (4 cores A15-2.0 GHz and 4 cores A7-1.4 GHz) and 2GB LPDDR3 RAM. This application is optimized using OpenMP to profit the advantages of the multi-cores processor.

The result of the postcard tracking is given in Figure 3.13 with the high precision of the position estimation. The number of extracted keypoints is fixed to be 800 for the reference image and also for each video frame. Besides, table 3.4 shows the tracking time on ODROID-XU4 for 200 frames. We can note that in HOOFR detection, keypoints must be classified based on their Hessian score in order to select the most relevant ones. Therefore, the execution time of HOOFR detection is high variant due to the classification time. On the other hand, RANSAC algorithm is used to estimate Homography matrix. This algorithm builds a sub-set of matchings by choosing randomly the elements. Hence, its execution time depends on the ratio of the good matchings in the matching set. In our experimental test, in the worst case (all steps take the maximum time), the execution time for one frame is 76.5 ms corresponding to the frequency of 13 Hz (13 fps). On the average, after 200 frames, the execution time is 56.4 ms for each frame corresponding to 17 fps.

Table 3.4: Tracking time (ms) on ODROID-XU4 for 200 frames

	HOOFR		Matching	H-matrix	Total
	Detection	Description			
The best case	15.7	9.1	27.2	1.3	53.3
The worst case	35.1	9.8	29.5	2.1	76.5
Average	17.2	9.7	28.0	1.5	56.4

3.6 Conclusion

This chapter has presented the HOOFR extractor which aims to address the front-end part of visual SLAM system on detecting, describing and matching image keypoints. The detector is the combination of ORB with a Hessian score, while the descriptor employs a human retina based description consisting of a FREAK version with enhanced overlapping. The proposal offers a better compromise between speed and matching quality compared to others state of the art algorithms. The experimental tests show that HOOFR exhibits competitive performance but much faster than SURF, SIFT. Besides, HOOFR exhibits comparably low computational cost as ORB but has better performance. HOOFR extractor was also proved to be implemented efficiently on an embedded platform such as ODROID-XU4 with a low processing time [95]. After having an enhanced extractor, in next chapter, we will present our proposed SLAM algorithm using features extracted from HOOFR.

Chapter 4

HOOFR Stereo SLAM

Following the presented HOOFR extractor, this chapter introduces a proposed SLAM algorithm based on HOOFR features.

4.1 Related Works

In the literature of VSLAM, existing approaches are based in two predominant perception strategies: monocular and stereo. Stereo VSLAM is generally transposable to RGBD systems [96, 97]. The most versatile of VSLAM approaches is the monocular VSLAM [98, 25, 99] since its hardware requirement is only one camera to observe the environment. However, "scale drift" remains an open problem of this approach. This is due to the fact that frame-to-frame motion estimates are integrated over time up to an absolute global scale. On the other hand, stereo VSLAM uses two calibrated cameras to capture the scenes so the depth from camera to points can be computed for each frame using the disparity.

Over almost two decades, there have been many successful stereo VSLAM methodologies. Hereafter, existing VSLAM frameworks are surveyed with a particular focus on their core methodology (i.e. filter or keyframe based methods) and data representation (i.e. feature or dense image based methods) as shown in figure 4.1.

Early stereo VSLAM frameworks were based on classical EKF approach enclosing a large Extended Kalman Filter for managing all landmarks [100, 101, 102]. This approach suffers from two main problems: firstly, the quadratic complexity of the EKF limits the

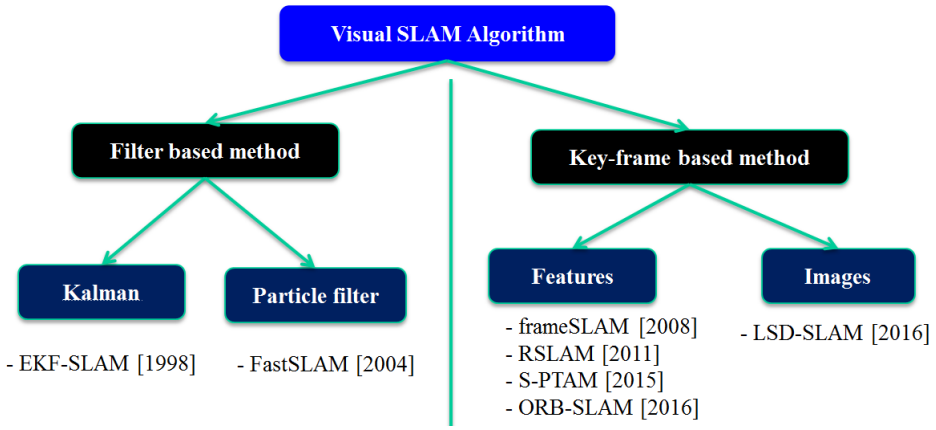


Figure 4.1: Visual SLAM overview

number of processed landmarks; and secondly, the consistency of EKF is known to be poor causing the impossibility of re-linearizing the cost function. In order to tackle such drawbacks, authors in [103] proposed to split a large EKF filter into sub-maps. However, this variant limits the application to environments with an area of 100m². An alternative variant is FastSLAM [104] which represents trajectories by means of a set of particles and small EKF filters are assigned to each landmark. FastSLAM framework is also afflicted by its complexity when the number of particles is not bounded for a given environment. It also suffers to achieve loop-closures due to the 6-DOF nature of visual SLAM, making this approach not well-suited for large-scale scenarios.

Since EKF and FastSLAM are filter-based frameworks, they marginalize all past poses and summarize the information gained over time with a probability distribution. In contrast, keyframe-based VSLAM performs a windowed optimization (e.g. bundle adjustment) on a small set of past frames to optimize current pose. Then, the global optimization in case of loop closure could be done using a Graph-based SLAM method [105, 106]. Strasdat *et al.* [107] compared filter and keyframe based VSLAM demonstrating that the latter achieves a better balance between computational cost and precision.

Inside keyframe-based VSLAM methodologies, the use of two different data representations can be highlighted: feature and image based. Feature-based strategy comes out earlier and was inspired on several researches. For instance, a scalable stereo visual SLAM have been introduced in frameSLAM [35] and RSLAM [108]. The contribution of frameSLAM consists in reducing the complexity of large loop-closures by constructing sub-maps and simplifying feature constraints into frames constraints. In this way, the

mapping task was optimized so as to maintain a subset of frames (skeleton). RSLAM implements a local bundle adjustment with a bounded complexity in order to provide an accurate map and trajectory. Even if RSLAM achieves constant time complexity, the global consistency is not warranted. S-PTAM proposed by Pire et al. [88] exploits, in parallel, the tracking and mapping in order to achieve real-time performances. However, it lacks large loop closing which is indispensable in an accurate SLAM system. Recently, ORB-SLAM appears to be one of the most actively developed VSLAM framework. After the monocular version, Mur-Artal et al. contributes with a stereo version of ORB-SLAM in [96] to handle the problem of scale drift. They inherit the main spirit of S-PTAM and complement it with a loop closing procedure. A first dense image-based approach is latterly presented in which LSD-SLAM [109] can be named as candidate. This approach provides depth estimation and mapping by direct image alignment with affine lighting correction on a rich set of pixels having a high intensity gradient. LSD-SLAM provides good results under low image resolution and small camera motions. The use of high resolution images or video sequences with an important inter-frame camera motion with LSD-SLAM provides poor localization results and its computational cost becomes a severe issue.

4.2 Algorithm description

The transformation (translation and rotation) of a stereo camera can be computed in homogeneous coordinates (up to scale) by one image chain (left or right). Therefore, in our system, we employ only the left image for relative motion estimation between two camera positions. The right image is used in further step to calculate the real scale. For each input stereo frame, HOOFR features are extracted in the left image and HOOFR description is then computed for both motion estimation and loop detection. HOOFR features are matched with those of previous left image in order to estimate relative transformations between the current frame and the previous frame. We define the "previous neighbor frame" (PNF) as the frame that we can estimate the camera transformation from it to current frame through the essential matrix.

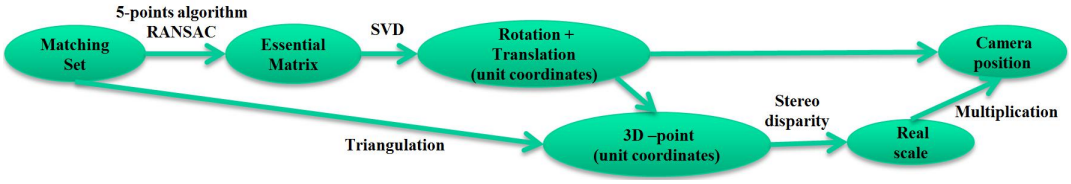


Figure 4.2: Frame-to-frame estimation

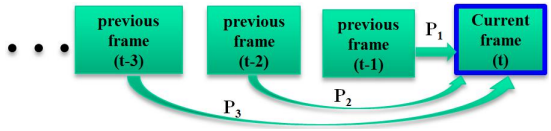


Figure 4.3: Optimized current pose estimation

The frame-to-frame estimation is shown in figure 4.2. The Essential matrix (E) [45] corresponds to one relative transformation and can be computed from the matching sets using RANSAC [110]. The camera translation, camera rotation and landmarks positions are extracted from E by triangulation but they are in homogeneous coordinates (up to scale). In order to have a real scale, we use stereo matching to match the position of triangulated landmarks in the left image to those corresponding in the right image. The real landmark-camera distance is computed based on this stereo matching and the distribution of the left and right cameras. Scale factor is the ratio of real distance on the triangulated distance. Finally, the camera motion is estimated as the product of the homogeneous motion and the scale factor.

To achieve the optimized camera pose, the main idea of our design is that we do not employ bundle adjustment which presents a high processing cost. Instead, we propose another method denoted as "windowed filtering" illustrated in figure 4.3 which estimates camera pose of current frame from a set of previous neighbor frames. For each previous neighbor frame, we apply the entire motion estimation to achieve one prediction of current pose. Each predicted pose is associated with a weight corresponding to its confidence in comparison to others predictions. The optimized current pose is then the mean of all predictions by their weights respectively.

In parallel with current-to-neighbors motion estimation (mapping), we perform a loop detection test for left image as shown in figure 4.4. HOOFR binary feature description is used once again to extract image description. The current left image is queried in key-frame set to find the max-likelihood. In the case of low matching score, current

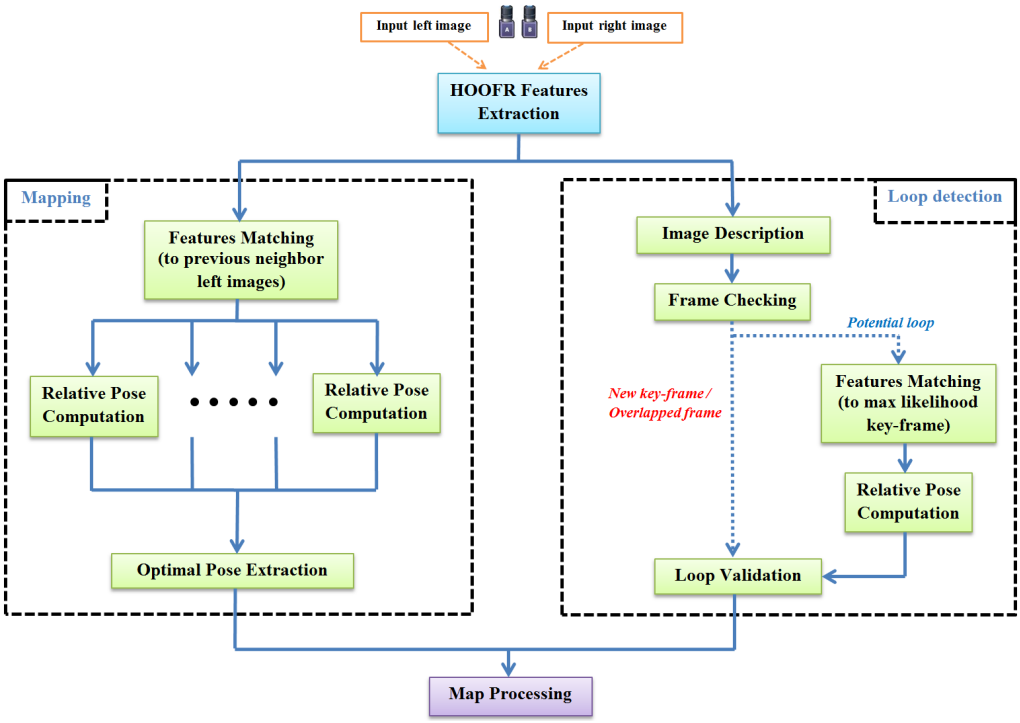


Figure 4.4: Functional blocks of the algorithm flow at each input stereo frame

image is considered as a new key-frame, we add current position attached with its image description to pose graph. In contrast, potential loop closing is considered when high matching score is presented and the max-likelihood key-frame is far from current frame in pose graph. The motion estimation between current frame and max-likelihood key-frame is then computed to validate the loop closure. A real loop is taken into account only if motion estimation is successful.

4.3 HOOFR features

4.3.1 Bucketing feature detection

HOOFR [111] extracts key-points that are used for motion estimation. To ensure a precise estimation, many correspondences are required so that many points should be detected and described in an image. Due to this reason, we need a high speed extractor. HOOFR detector is the combination of ORB detector with Hessian score and provide better compromise between execution time and matching precision [95]. The main idea of HOOFR detector is that it detects features in an image by applying FAST detector over multiple

scales of an image pyramid. Then, the detected features are filtered to keep the most relevant key-points based on their Hessian score (instead of Harris score in ORB). This filtering provides a good repeatability. It is eliminated in the processing flow of some works such as LSD-SLAM [53]. However, in our system, we maintain this filtering step to improve the matching result for an enhanced pose estimation.

In order to warrant a homogeneous distribution of features, we employ bucketing technique as used in all others systems. The input frame is divided into a grid where the number of cells depend on the image resolution. HOOFR features are then detected with adapting threshold trying best to extract enough points. We fix the maximum number of points retained in one cell to PTS_PER_CELL. In each cell, at the first detection, FAST threshold is set to a high value $FAST\_THRES$. Unless the number of key-points is higher than PTS_PER_CELL, the second detection is operated with lower FAST threshold value ($FAST\_THRES/2$). After detection, if the number of points is higher than PTS_PER_CELL, we compute Hessian score for each point and maintain only PTS_PER_CELL points having the highest score. The orientation and description are computed by HOOFR descriptor for the most relevant points retained by each cell. HOOFR descriptor (256 bits) is an enhanced version of FREAK (512 bits). It has a low sensitive to viewpoint and is fast to compute and match.

4.3.2 Binary descriptor for place recognition

Scene recognition is the fundamental step for loop closure. Typically, this step uses SIFT or SURF full-featured descriptors due to their high matching score among the existing approaches. Nevertheless, their computational cost has degraded performances of SLAM systems. Recently, binary bag of words [112] is proposed with a competitive performance by an order of magnitude faster than floating approaches. this approach is widely used and has remarkable results in visual SLAM systems such as [25] and [27]. Thus, in our system, we integrate the idea of binary word so as to keep place recognition process light. Among the relevant key-points provided by HOOFR detector in the whole image, we select K points (K = 150 in our implementation) having the highest Hessian score to get corresponding words based on their HOOFR description. Image description is built from these binary words.

4.4 Mapping

4.4.1 Features matching

In mapping thread, features matching is carried out between the current frame and the previous neighbor frames. We note that the camera frequency is high (10-50 fps) leading to a little change between consecutive images. For this reason, the correspondence in neighbor frame is located not too far from key-point position in current frame, so we can limit the searching region instead of the whole image. Figure 5.4 illustrates our searching strategy. For each key-point I in PNF, we perform "Brute-Force" matching with all key-points locating in the same cell or the “neighbor cell” in the current frame. We find the most and the second correspondences (J and $J^{'}$) by the smallest and the second smallest Hamming distance respectively. The result of feature matching has an important role in the precision of pose estimation so that it should be done carefully. Hence, we apply further three following conditions to select pairs among the most matching pairs $I-J$ (smallest Hamming distance):

Firstly, the matching pair must have a high distinction in comparison to its second matching. It means that the ratio of Hd_{I-J} to $Hd_{I-J'}$ must be lower than a threshold φ where Hd_{I-J} represents the Hamming distance of the pair $I-J$. The value of φ is 0.85 giving a good exhibition in our experiments.

Secondly, if the positions of I and J have a small difference in the images ($\|p_I - p_J\| <$ 2), it means probably that the point is too far from camera or the camera does not move significantly compared to the previous pose. Such two cases do not provide a good estimation so that these matches should be also rejected. Furthermore, if too many matches have a small position change, the camera can be considered staying nearby the previous pose.

Thirdly, in contrary to the second condition, if I and J have too big differences in positions ($\|p_I - p_J\| > 120$), it could be a false matching and also must be eliminated.

In some other researches like ORB-SLAM or LSD SLAM, people use guiding search to find correspondences. They rely on the last transformation of camera and point position in the previous frame to predict the point position in the current frame. This method has a good performance when the transformation is small and stable but it is easy to

loose the tracking when the transformation becomes more critical. In our algorithm, we apply Brute-Force to find the best candidate in a large set of local features. After 3 test conditions above, we can get a reliable matching set for the following step.

4.4.2 Relative Pose Computation

The goal of Relative Pose Computation (RPC) block is to compute the relative pose between two frames (always from left images of a stereo camera) and to triangulate a set of map points. There are many RPC blocks executed in parallel. We defined these execution as sub-threads inside mapping thread. Each of them estimates one relative motion from current frame to one previous neighbor frame. RPC block consists of 3 principal steps: rotation and translation extraction from essential matrix, solution determination and scale estimation. We assume the image domain to be given in stereo-rectified coordinates, the intrinsic (focal length, center points) and extrinsic (baseline, relative angle) camera parameters are calibrated a-priori.

- Rotation (R) and translation (t) extraction from essential matrix (E)

Essential matrix is estimated from HOOFR matching set returned by the previous step. The epipolar geometry is described by equation (4.1):

$$[p_I;1]^T K^T E K [p_J;1] = 0 \tag{4.1}$$

where K is the intrinsic camera matrix, p_I (x_I, y_I) and $p_J(x_J, y_J)$ are respectively positions in PNF and current frame of a correspondence $I - J$. Each matching pair gives a constraint to solve E. In others works such as ORB-SLAM, people use 5-point algorithms [43] inside a RANSAC scheme to extract an optimized model E_{op} from matching set. They assume a standard deviation of one pixel in the measurement error. Then, they consider R and t extracted from E_{op} as the initial state for the optimization of bundle adjustment (BA).

In our proposal, we intend to avoid BA which has a high computational cost. Hence, we focus on the method to improve the precision of estimating E, which makes the most difference of our system with others in state of art. Before computing E, number of

matching pairs (np) in each RPC block is checked. If np is under a threshold λ, the corresponding RPC block is considered as an invalid estimation and its sub-thread will be stopped immediately. In contrariwise, E estimation is processed when np is bigger than λ. Through experiments, we found that a high precise localization is presented when the measurement error (me) of inlier in RANSAC scheme is smaller than 0.4. However, when we apply RANSAC with me = 0.4 to the initial matching set, the execution time severely increases. The reason is that the number of iterations in RANSAC is updated during the estimating process. After each iteration, the remaining number of iterations is computed by equation (4.2):

$$N_I^i = max(N_I^{i-1} - 1, log\frac{1-c}{1-(n_e/N_e)^5}) \tag{4.2}$$

where N_I^i is the remaining number of iterations at time i, c is the parameter of confidence (normally between 0.95 and 0.99), $n_e$is the number of inliers in the best model at time t and N_e is the total number of elements in the whole set. When the measurement error is smaller than 0.4, it is obvious that n_e decreases leading to the increase of remaining number of iterations. Therefore, in order to accelerate the processing, we propose the Algorithm 4.1 for estimating E_{op} :

Algorithm 4.1 Essential matrix estimation from matching set

1. Apply 5-points algorithm inside RANSAC scheme to the initial matching set with the measurement error equal to 1.0.
2. Select the inliers corresponding to the optimal model of step 1 to form another set (refined matching set).
3. Apply 5-points algorithm inside RANSAC scheme to the refined matching set with the measurement error equal to 0.4 to get a final optimal model (E_{op}).
4. Test the final optimal model of step 3 on the whole initial matching set to select the inliers (measurement error reclaims the value of 1.0).
5. Compute the mean of measurement errors returned by inliers from step 4. The inverse of this value represents the score of the estimated model.

We mark the inliers of E_{op}, while outliers are rejected. Given that E_{op} has been determined; our method for estimating rotation R, translation t and 3D points triangulation

is based on performing single value decomposition (SVD) of E (mentioned in Hartley & Zisserman's book [45]). Due to the fact that E is "up to scale" so that SVD provides the solution of $[t]_m$ in homogeneous coordinates (scale is not defined). Furthermore, we have 2 opposite directions which are possible for translation (t) and two different rotations (R) which are compatible with an essential matrix. This gives four classes of solutions in total for the relation between two camera coordinates. However, there is only one correction solution where the triangulated point is in front of camera at both positions (current and reference positions).

- Solution determination

In order to select the correct solution among the four possibilities, for each inlier matching pair, we compute 3D triangulated position in the 4 solutions. The point is arranged to the solution in which it is in front of camera at both reference and current positions. The chosen solution is the one containing the most points seen in comparison to others. In theory, if the estimation of E is noiseless, one solution will contain all triangulated points. In that case, we can check only one matching pair to find the valid solution. However, matching is affected by noise in practice, so checking all matching pairs provides a more robust method. In particular, if image is too degraded by noise leading to no clear winner solution, the relative pose estimation of the corresponding sub-thread is stopped immediately and will be marked to be invalid.

- Scale estimation

As the essential matrix is "up to scale", the translation and 3-D triangulated points computed above are in unit coordinates. Therefore, the residual problem after selecting the valid solution is determining the "real scale" of map and camera motion.

The most advantage of a stereo camera is providing two images from different physical cameras, taken at the same time. Hence, the depth from 3D point to camera can be estimated without scale ambiguity using stereo-disparity (static stereo). Assuming that we have a point in the left image, the correspondence of this point can be searched along epipolar line in the right image. In our case of rectified-stereo, this search can be performed along the horizontal lines.

As stereo correspondence measure, we use 5 pixels-SSD method [113] along the scan-line. In our system, we obtain *a-priori* a point in the left image. Hence, if we consider the same position in the right image, the correspondence is located undoubtedly on the left side of this position. In practice, the disparity range in the right image is constrained to $[(x_J - \sigma, y_J), (x_J, y_J)]$ where σ is the limited search region (σ=30 in our experiment). Once the correspondence is defined, the real 3D point $\bar{P}_J(\bar{X}_J,\bar{Y}_J,\bar{Z}_J)$ with reference to camera will be extracted by well-known static-stereo triangulation (using disparity, baseline and camera focal length) as mentioned in [45].

Algorithm 4.2 RANSAC scheme for scale estimation

1. Take the value (k_{av}) of one element in the factors set.
2. Find the number of inliers in the entire set. A factor k_J will be classified as an inlier if the difference is small enough ($|k_J - k_{av}|/min(k_J, k_{av}) < \varepsilon$). ε is set to 0.1 in our test. Mark the scale value if it is the best model (contain the most inliers).
3. Repeat the processing for all other elements in factors set. The value of the best model is considered as the estimated scale.

We compute the real distance for all triangulated points arranged to the selected solution. Scale factor (k_J) is the ratio of the real distance of static stereo on the triangulated distance of temporal stereo. In fact, this factor is simply computed by the ratio of $(\bar{Z}_J/Z_J)$. In the case of noise absence, all points have the same scale factor. However, it is never the case in practice. To have an appropriate value, we consequently employ 1-point scheme on the scale factor set as in Algorithm [4.2].

In order to have reliable scale estimation, after doing 1-point scheme, we additionally evaluate the best model if the number of iterations reaches to the bound. The model is invalid when the number of inliers could not attain an acceptable value ($N_{inliers} < \gamma$). In this case, we also reject the current process, the sub-thread returns invalid estimation. Otherwise, camera translation and 3-D point position are multiplied with the scale to get the non-scale value and only 3-D points computed from inliers are maintained. Through experiments, we found that the value of γ is set to 10 giving a good performance.

4.4.3 Optimized pose extraction

This block is the summary step in mapping thread and takes into account all predictions from sub-threads to calculate the optimal camera pose. In practice, for each sub-thread, we notice that relative pose is extracted from Essential matrix which is obtained beforehand from features matching set. Therefore, we propose to use inverse of mean error retained by inliers after essential matrix estimation as a weight factor of predicted pose. Equation (4.3) shows the computation of optimal current left camera pose C^l (also defined as $[R|t]$ in some references) from all predictions:

$$C^l = \Sigma_{n=1}^{N} \frac{\sigma_n}{\Omega} \hat{C}_n^l \tag{4.3}$$

where $\hat{C}_n^l$ is the predicted position of the sub-thread Th_n, σ_n is the inverse of mean error of inliers and $\Omega = \Sigma_{n=1}^{N} \sigma_n$. Besides, N represents the number of valid sub-threads which compute a prediction with positive weight. Contrariwise, when a sub-thread is marked to be invalid, its weight takes the value of 0 and it will be ignored in optimal pose extraction. When all prediction are invalid, current frame would not be tracked. In this case, map could not be updated and we proceed directly to the next frame.

4.5 Loop detection

Loop detection processes the current frame and tries to detect a loop closure.

4.5.1 Place recognition using FABMAP 2.0 and binary word

For loop detection and re-localization, our system implements place recognition method based on FAB-MAP 2.0 module exhibiting a robust performance as shown in [114]. It is tested on 1000 km dataset proving an ability to work on a very large scale environment.

FAB-MAP employs a Bag of Words (BoW) to describe images. To train the BoW vocabulary, the original proposal extract SURF or SIFT feature descriptors from a training dataset. These descriptors are then clustered and the BoW vocabulary is achieved from these cluster centers. From there, an image can be described using this vocabulary by quantizing its SURF or SIFT, and listing which words were seen. An image descriptor

(BoW descriptor) can be performed as binary of word presence, or as a list of which words were observed. In order to learn a factorized probability prior distribution over image descriptors set, FAB-MAP trains a Chow-Liu tree using the BoW descriptors generated from training dataset. A place is represented as a vector of Bernoulli variables indicating the existence of the generator for each word in the vocabulary. There are two different version of FAB-MAP. In FAB-MAP 1.0, the measurement model is given by the trained Chow-Liu tree and full Bayesian inference determines the posterior generator probabilities. This approach work at the scale of a few kilometers (or extended to few tens kilometers thanks to an approximate inference techniques) due to its computation cost and memory requirement. Otherwise, FAB-MAP 2.0 speeds up the inference using an inverted index for each word in the vocabulary with slightly modified computing method.

In HOOFR SLAM, we reuse feature descriptor extracted from HOOFR extractor for both motion estimation and place recognition. In contrast to floating descriptor of SIFT or SURF in the original FAB-MAP, our feature descriptor is binary so that we replace floating distance by hamming distance in the word clustering. The bag of words generated from training data is also in type of binary words.

In fact, we use the HOOFR extractor to build the bag of words in a large training images set. A 256-bit binary descriptor contains in total 2^{256} different words. However, a huge vocabulary not only takes much time to build image description but also has a poor efficiency in loop detection. The issue is that we have a low tolerance when too many words are maintained in the vocabulary. In such case, a 3D point will be assigned easily to two different words when camera has little position change. Consequently, a low similarity between 2 images is presented through these images of the same scene. In experiments, we found that a vocabulary of 10000 words provides a favorable compromise between precision and execution time. The vocabulary is created offline one time from a large set of random images and is used for all test sequences.

4.5.2 Map and Key-frame set

In our system, map is represented as a set of $M_i = C_i^l, T_i^l, L_i$. Each M_i contains position of left camera C_i^l in global coordinates, relative transition T_i^l to previous left camera position and all 3-D landmarks positions L_i in camera coordinates. Our developed system is similar

to recent SLAM systems that do not consider all processed frames as key-frames due to 2 main reasons:

- For each input frame, each element in key-frames set will be queried to compute the likelihood percentage. The frame sampling is aimed to reduce the size of key-frames set. Hence, the computation will be light. This strategy is suitable for implementing the application on an embedded system where memory resources are limited.

- There exists always an overlap between consecutive frames. It means that when images are taken from close times, they contain many common words. In many cases, two images take exactly the same words from environment. The overlap will cause problem in computing likelihood percentage when these two images attain the same value. In this case, all percentages have small values causing the ambiguity in loop detection.

Therefore, we consider a frame as a new key-frame when there is no likelihood percentage value bigger than η (0.99 in our experiments). Each key-frame is then updated into key-frame set $KE_i = id_i, V_i, D_i$where id_i is the index of the key-frame position in the map, V_i and D_i are respectively features and their HOOFR description extracted from the key-frame.

4.5.3 Frame Checking

First of all, in "Frame Checking" block, K binary words retrieved from the most K relevant features in HOOFR extraction are employed as well as vocabulary to build image description. Specifically, each of K binary words will be queried in vocabulary to find the best matching word (lowest Hamming distance). Image descriptor is formed by taking into account which words that image takes from the vocabulary. Likelihood percentage is then computed for all elements in key-frames set based on their image descriptor. If the maximum likelihood is less than η, "Frame Checking" decides that current frame is a new key-frame and we will update it to the key-frame set in "Map Processing" block.

When maximum likelihood is bigger than η, the frame is not a new key-frame. However, we fall into two possibilities: the overlap with previous images or potential loop

detection. In practice, to manage key-frame set, a variable called "historical time" (*ht*) is additionally attached to each element in the set. Once key-frame is added to the set, this value is initialized as the size of the set at that moment. Besides, when loop closing is successfully processed at this key-frame, *ht* is updated by the size of the set at the update moment. A "new-comer" (newly added or processed) is identified when *ht* is closely to actual size of key-frame set. Moreover, after a loop closing, it is probably that we have many loop points nearby. In order to avoid too many loops processing, we count the number of new key-frames added from a loop point. "Frame Checking" recognizes a potential loop when two conditions below are satisfied:

- Historical time of maximum likelihood key-frame ht_m is smaller than the size of actual key-frame set by t ($ht_m < keyframeset.size() - t$).
- Ne new key-frames have been already added from the last loop point.

where t, Ne are respectively set to 5 and 10 in our experiments. Otherwise, it is recognized as an overlapping frame.

The features matching and relative pose estimation between current frame and maximum likelihood frame are performed only in the case of potential loop. We use the word "potential" because the current frame must finally be validated by pose estimation. In our experiments, most of the frames are recognized as a new key-frame or overlap with the previous frame. Features matching and Pose estimation tasks are processed only when a loop point is closely attained. Nevertheless, we found that some particular frames are potential loops but they are not the real loops. This is inevitable and it occurs when two images of different places take too many common points in the vocabulary. However, these frames are rapidly rejected after features matching due to the lack of valid correspondences or rejected in pose estimation based RANSAC since there is not enough inliers.

4.5.4 Features matching

In loop detection thread, features matching block uses current frame and its max-likelihood frame when a potential loop is detected. As a precise loop requires severe checking conditions, we propose to use "cross Brute-Force" matching instead of the high

distinction checking. The idea is that we keep the second and the third checking conditions as in features matching of mapping thread. However, we change the first condition as following:

- Two feature sets are matched using local Brute-Force matching in two direction. For each point I in the max-likelihood frame, we find the best local match J (smallest Hamming distance) in the current frame and vice-versa.
- The matches verify the first condition if they have the same matching results in two direction ($I \rightarrow J$ and $J \rightarrow I$).

This stricter condition allows us to detect the "false positive" of potential loop (a high similarity but not the same scene) where few matching pairs retained after checking.

4.5.5 Relative pose estimation

RPC block in loop detection is similar to that in mapping thread except the change of the threshold λ. We also increase the value of λ to insure that only "true positive" of potential loop is handled. The reason is that after a tightening matching, we require more number of matching pairs retained to compute essential matrix. In experiments, we found that this combination exhibited a tremendous performance with no "false positive" loop passing.

4.6 Map Processing

Map Processing block considers results returned from mapping and loop detection threads to make the decision. Table 4.1 resumes all possibilities that the system can meet. If the mapping consecutively fails after a fixed number of frames due to some reasons such as abrupt movement or occlusion, our system turns into tracking-lost state (tracking lost = true). In this state, each frame is processed only by loop detection thread. Mapping thread is disabled. Once the camera is relocated in the map, we return to tracking-active state. However, map optimization will be neglected as the lack of previous poses. Moreover, map will be discrete at the relocated point and the incoming optimization is limited to this point. In a normal situation when tracking-lost is false, if mapping is invalid for current frame while loop detection provides a legal result, we to store the loop information. In

Table 4.1: Possibilities and decision of “Map Processing” block

Blocks									
Tracking lost	TRUE			FALSE					
Mapping	-			Valid			Invalid		
Loop detection	Key-frame	Neutral	Loop	Key-frame	Neutral	Loop	Key-frame	Neutral	Loop
Decision									
Update pose graph	-	-	+	+	+	+	-	-	-
Add key-frame	-	-	-	+	-	-	-	-	-
Activate Map Correction	-	-	-	-	-	+	-	-	-
Re-localization	-	-	+	-	-	-	-	-	-

the limited following frames, in the case that mapping revives, the loop closing will be performed.

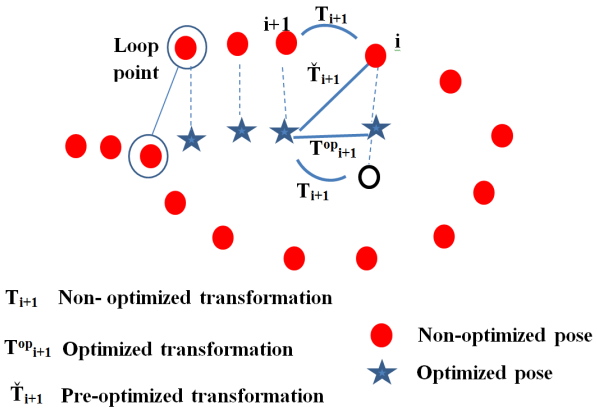


Figure 4.5: Loop correction

"Map correction" is called only if both loop and mapping threads return valid estimations. Once it is activated, the trajectory is optimized by distributing loop closing error along pose graph. The propagation starts from loop point, follows the trajectory back to the point to which loop point is attached. Figure 4.5 shows the correction applied for each position in the map. Assuming that position $(i+1)$ is optimized, we can compute the transformation $\check{T}_{i+1}$ between the optimized pose C^{op}_{i+1} and the non-optimized pose C_i, while T_{i+1} is the transformation between two non-optimized poses already maintained in the node $(i+1)$. T^{op}_{i+1} is then estimated by equation (4.4) where μ represents the “propagation coefficient”. In our experiments, we propose to compute μ depending on the number of optimized positions (N_p) in total by equation (4.5). The optimized pose of node i (C^{op}_i) is finally computed by equation (4.6).

$$T_{i+1}^{op} = \mu \check{T}_{i+1} + (1-\mu) T_{i+1} \tag{4.4}$$

$$\mu = \pi / N_p \tag{4.5}$$

$$C_i^{op} = C_{i+1}^{op} * T_{i+1}^{op^{-1}} \tag{4.6}$$

The execution time of map correction depends on the map size. Following time, this step becomes costly with a large loop closure. To warrant frame-rate processing, "map correction" can be launched as a thread in parallel and continue to process next frame. However, the key-frames set is blocked in order to avoid memory accessing dump during map correction. As a consequence, any new key-frame is added and we just update pose graph until the current correction thread is finished.

4.7 Evaluation results with experiment datasets

We evaluate our proposed algorithm on different well-known datasets: KITTI [74], Oxford [75], Malaga [76], MRT [77], St_Lucia [78] and New-College [79] with full image resolution.

4.7.1 Stereo image rectification

In HOOFR SLAM algorithm, we search the stereo correspondences for a feature along the x-axis. It means that the cameras are supposed to be a stereo system of horizontal epipolar lines (simple stereo configuration). In practice, we cannot physically place the two cameras to have the such system due to their different focal length, different center points or distortion. However, we use an algorithm to change a general configuration (figure 4.6 on the left side) to a simple configuration(figure 4.6 on the right side), this is known as the stereo rectification in the state-of-the-art. It also compensates image distortion. This algorithm is considered as a pre-processing step before images are used in SLAM algorithm.

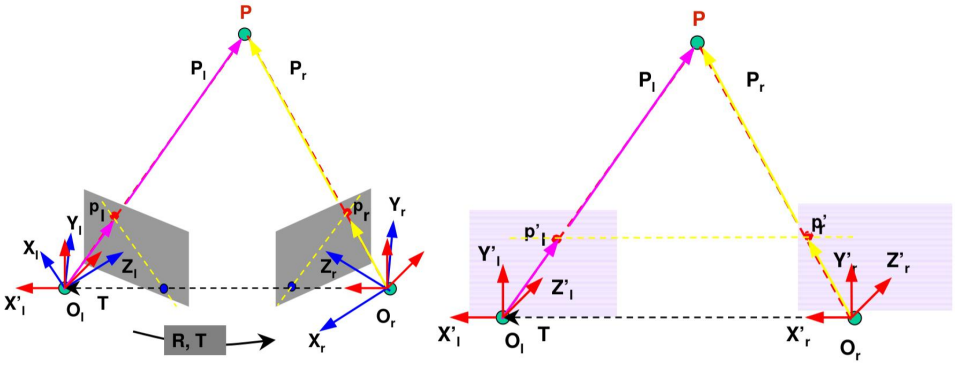


Figure 4.6: General(left) and Simple(right) stereo configuration

Among six datasets, only KITTI provides images already rectified. New-College and Oxford present source code to generate rectified images from the raw images. The others present only raw images with calibrated camera matrices. The pre-processing is hence required for these datasets and is realized using our source code (written in C++ base on OpenCV 3.0). Moreover, similarly to other SLAM based feature, our algorithm work on grayscale image so that all color input frames are converted to monochrome during the frame reading.

4.7.2 Parameters

Parameter name	Value
Number of features	1500-2500
FAST threshold ($FAST\_THRES$)	12
Difference threshold in feature matching (φ)	0.85
Low threshold of pixel position change	2
High threshold of pixel position change	120
RANSAC threshold in E estimation	1.0-0.4
Inliers scale threshold (ε)	0.1
Number of binary words for loop detection (K)	150
Maximum neighbor frames ($nframes$)	4

Table 4.2: Algorithm parameters

Table 4.2 regroups the main parameters of our algorithm used in the experiments. The parameters were chosen through the experiments on different sequences to have the optimized value. In order to warrant the precision, we detect 2000 features per image in KITTI, MRT, St-Lucia sequences; 2500 features in Oxford and 1500 features in Malaga sequences. Corresponding to these number of features, the number of binary words for loop detection (K) was set to 150 where FAB-MAP 2.0 offers a high likelihood percentage when two images are taken from the same scene.

4.7.3 Evaluation with KITTI dataset

In KITTI dataset, ground truth is provided in the 11 sequences (00-10) by an accurate GPS and a Velodyne laser scanner. Some sequences contain a significant loop-closure, i.e 00, 02, 05, and 07. We compare the performances with stereo ORB SLAM: one of the most robust algorithms which uses high cost bundle adjustment and contains loop closure in the state-of-the-art. We apply the algorithm on 11 first sequences, blue curves represent the ground truth provided by a precise RTK-GPS.

The entire localization of the 11 sequences are shown in figure 4.8 observed in 2D of X-Z axis. We present camera postion on the 3 axis seperately for all frames of each sequence in the appendix - section 6.7. By reference to camera, X is the horizontal line pointing to the right side, Y is the vertical line pointing to ground and Z is the line pointing forward. Regarding the figure, our proposal have a competitive performance with respect to ORB SLAM except the sequence 01. The reason is that this sequence is captured by a car traveling on a high way with very high velocity. As shown in figure 4.7, on the high way enviroment, half of image is sky which do not provide relevant keypoints. Furthermore, on the road, there are also many low texture regions which do not contain keypoints. In this case, ORB SLAM obtains a precise localization by saving the map-points history and using the high cost bundle adjustment optimization. In contrast, our algorithm is aimed to get high speed processing and reduce memory resources usage, so that the precision is sacrificed in this case.

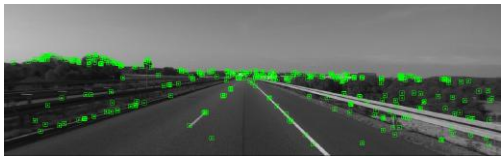


Figure 4.7: High way environment

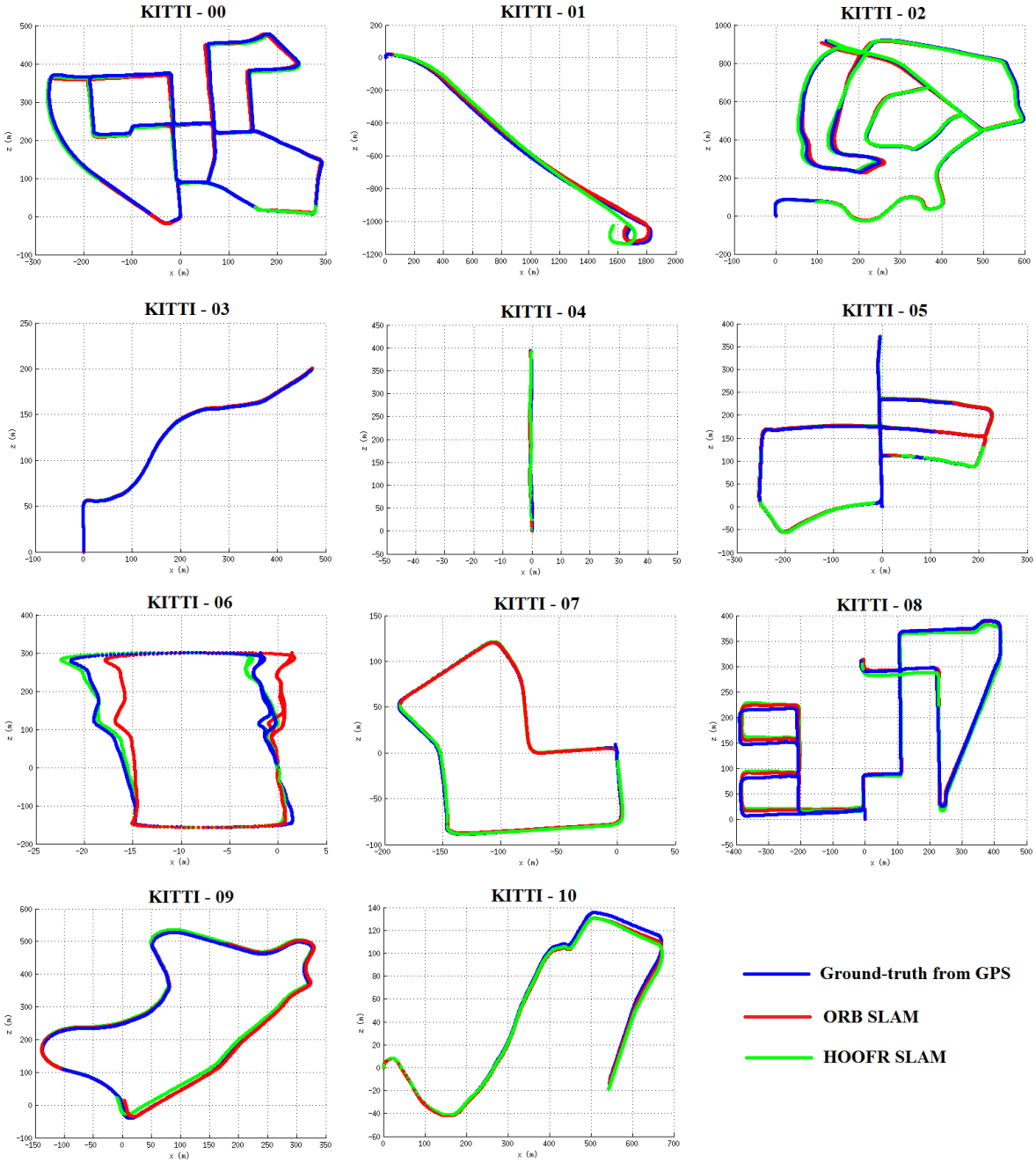


Figure 4.8: Localization results of ORB SLAM and HOOFR SLAM evaluated with KITTI dataset

Table 4.3 shows the Root Mean Square Error (RMSE) of trajectory for each sequence computed only for X and Z axis due to the fact that although GPS is corrected by RTK signal, ground-truth in Y axis is still not reliable. The results indicate that our system has a considerable accuracy with a trajectory error around 1% of its dimension (except 2% for sequence 01). The percentage is computed by the ratio of RMSE over the maximum value of 2 dimensions. Despite of the less complexity, our proposal even surpasses ORB-SLAM in some sequences such as 00, 02, 04 or 06.

Table 4.3: Root Mean Square Error (RMSE) in KITTI dataset of stereo HOOFR SLAM calculated for X and Z axis

Seq	Dimension(mxm)	Frames	ORB RMSE(m)	HOOFR RMSE (m)
00	564 x 496	4541	4.7612	3.2306
01	1840 x 1140	1101	17.7170	50.2589
02	599 x 946	4661	6.6243	4.7042
03	471 x 199	801	1.2390	1.2609
04	0.5 x 394	271	0.3677	0.3225
05	479 x 426	2761	1.1884	1.3507
06	23 x 457	1101	1.6343	0.8061
07	191 x 209	1101	0.9304	0.9199
08	808 x 391	4071	4.8629	6.4138
09	465 x 568	1591	4.2835	6.7374
10	671 x 177	1201	2.7623	3.7944

4.7.4 Evaluation with Oxford dataset

Oxford dataset is recorded by 6 cameras mounted onboard a vehicle traversing a route through central Oxford. The ground truth is provided by the fused GPS+Inertial solution. In order to evaluate HOOFR SLAM, we choose two sequences: the “static sequence” (recorded on 2014/05/06 at 12:54:54 GMT) contains very few moving objects and the “dynamic sequence” (recorded on 2014/06/24 at 14:47:45 GMT) is a challenging by a longer trajectory and in presence of many moving objects in the scene. Figure 4.9 shows the performances of HOOFR SLAM on these two sequences.

On static environment, HOOFR SLAM and ORB SLAM present a very robust performance where RMSEs are respectively 2.24m and 2.22m. However, the localization error is increased on “dynamic sequence” where the RMSE is 40m for HOOFR SLAM and 70m for ORB SLAM. One of the most challenge of the dynamic sequence is that there

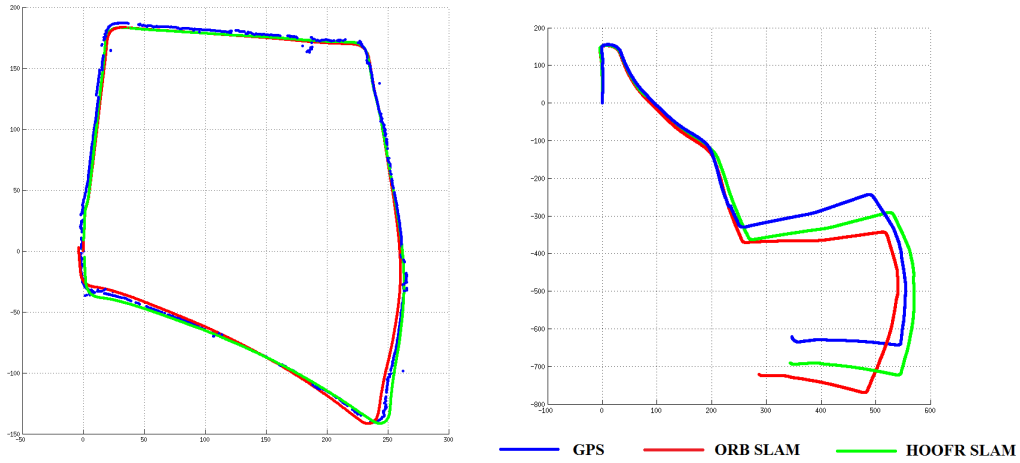


Figure 4.9: Localization results of HOOFR SLAM on static (left) and dynamic (right) sequences of Oxford dataset.

are some blurry images caused by sunlight. Hence, the degraded result can be explained due to two factors: the moving objects and the poor image quality.

4.7.5 Evaluation with MALAGA dataset

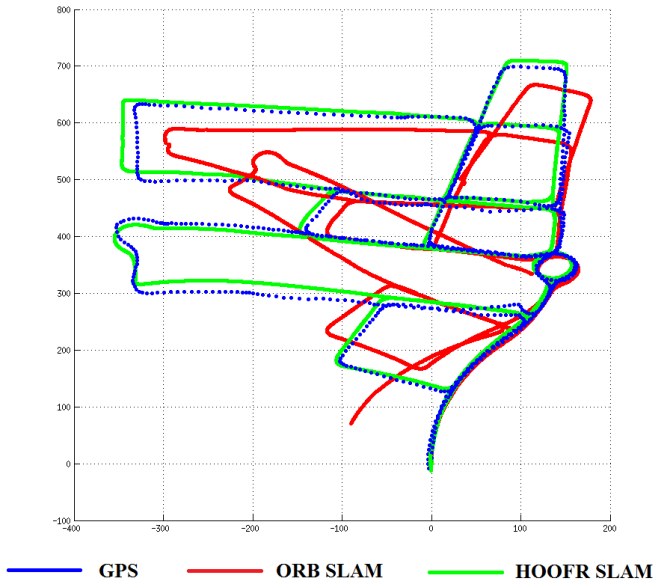


Figure 4.10: Localization result using Malaga sequences: GPS (blue), ORB-SLAM (red) and HOOFR-SLAM (green)

Malaga stereo dataset was gathered entirely in urban scenarios with a car equipped with a Bumblebee2 stereo camera running at a high rate (20fps). We chose the 10th sequence

of dataset because it contains a very long trajectory, several loop closing and a huge variation of image brightness during the experiment. We also test localization performances of stereo HOOFR-SLAM in comparison to stereo ORB-SLAM and the result is shown in figure 4.10. To the best of our attempts, ORB-SLAM could not exhibit a converging trajectory. At some points when the image brightness is low, ORB-SLAM provides a poor localization or even lost tracking. Otherwise, our algorithm shows a remarkable localization result with $nframes = 2$ and the number of keypoints set to 1500. The argument explaining this situation is that ORB-SLAM uses ORB detector while our proposal uses HOOFR detector. Following our previous publication [111], HOOFR detector has better repeatability than ORB detector in case of brightness change. By reference to GPS result, the reconstructed trajectory of our proposal is more reliable than that of stereo ORB-SLAM.

4.7.6 Evaluation with MRT and St-Lucia datasets

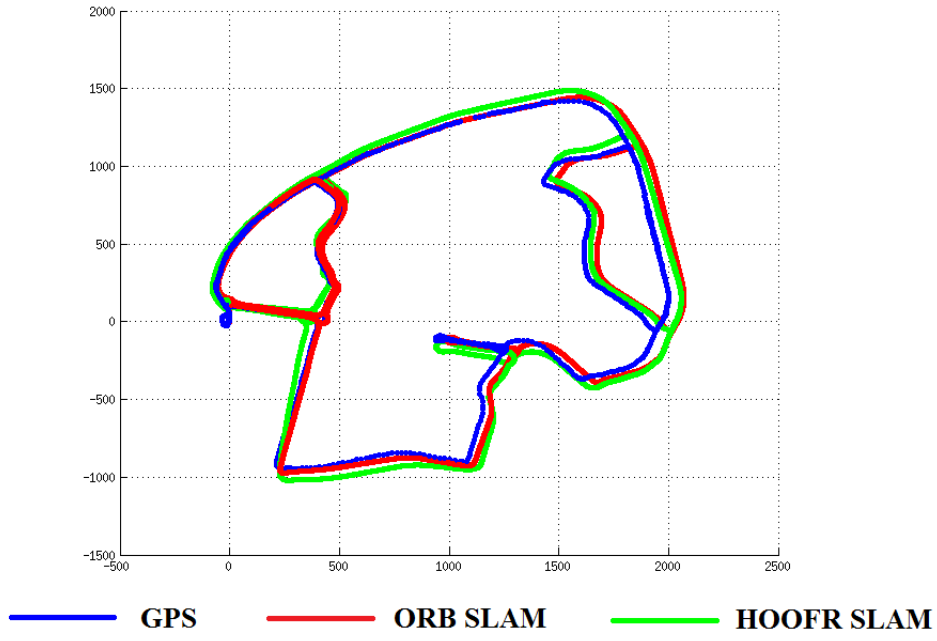


Figure 4.11: HOOFR SLAM reconstruction using St-Lucia dataset

In our experiments, we also validate HOOFR SLAM performances on two old datasets: MRT [77] and St-Lucia [78]. MRT is realized in 2010 using 20Hz calibrated stereo cameras. The stereo images are recorded from the AnnieWAY vehicle driven in a loop at

a bridge in the city of Karlsruhe. Besides, St-Lucia dataset is gathered from 30 Hz calibrated stereo cameras embedded on a car driven on 9.5 km around the University of Queensland's St Lucia campus. The reconstruction results of HOOFR for these two datasets are shown in figure 4.11 and figure 4.12 including trajectory generated by GPS (no RTK correction). Comparing to GPS data, HOOFR exhibits a considerable localization result. Noting that although GPS devices used in the two datasets are not very precise, it allows us to recognize the general shape of the trajectories.

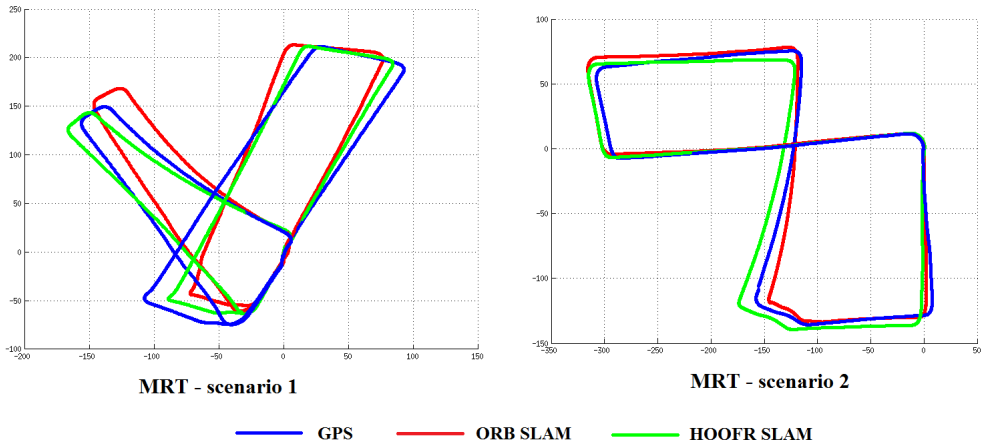


Figure 4.12: HOOFR SLAM reconstruction using MRT dataset

4.7.7 Evaluation with NewCollege dataset

Ground truth is not presented in this dataset, so that we cannot calculate RMSE on this dataset. Figure 4.13 shows the reconstruction for the full sequence with a view in details of a large loop closing. We note that by using a stereo camera, the scale can be computed independently for each frame, so we do not face the problem of scale drift as in ORB monocular SLAM [25]. Combing with strict conditions in selecting the correspondences, we achieve small localization deviation after a long trajectory.

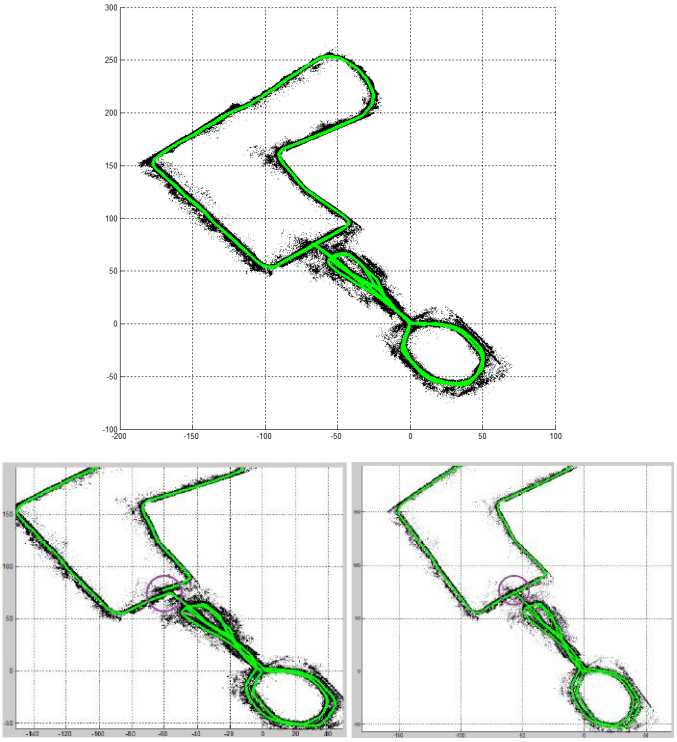


Figure 4.13: Reconstruction on NewCollege dataset

4.8 Conclusion

In this chapter, a novel estimation algorithm for feature-based stereo VSLAM has been presented. This approach is referred as HOOFR SLAM since it integrates HOOFR features extractor [111]. The binary descriptor is employed for both motion estimation and loop closure detection. Motion estimates are integrated over time following a hybrid filtering/key frame strategy. That is, position is estimated using a windowed weighted mean using previous neighbor frames. Weights are computed from inter-frame robust feature matching. A thorough experimental evaluation was carried out on six well-known datasets (KITTI, Oxford, Malaga, MRT, St-Lucia and New College). The evaluation on KITTI (reliable ground truth is presented) provides considerable localization results in terms of RMSE (around 1% of sequence dimension).

HOOFR SLAM satisfies the requirements described in the section 2.1.1 where:

- HOOFR SLAM has a high localization precision.

- HOOFR SLAM resources requirement are low, the computational complexity of mapping is constant overtime. Loop detection is detected rapidly without false positive results.
- HOOFR SLAM is suitable to be parallelized on heterogeneous architectures containing a massive parallel devices such as CPU multi-core or GPU.
- HOOFR SLAM is also easy to work on others sensor-combined systems such as: mono camera - IMU or mono camera - Odometers systems. The reason is that the most functional blocks of HOOFR SLAM algorithm use only monocular camera images. The second camera is used only for estimating scale. If a system contain one camera and other sensors allowing to know real scale, HOOFR SLAM will work without a doubt with a minor change in scale estimation block.

After the functional validation, in next chapter, we will present the implementation of HOOFR SLAM on CPU-GPU heterogeneous architecture and discuss about the timing performance.

Chapter 5

Embedding HOOFR SLAM on a CPU-GPU architecture

The HOOFR SLAM functionality was tested on several real datasets. To reach a high speed performance, this chapter introduces the study of implementing HOOFR on heterogeneous CPU-GPU architecture. This type of architecture is considered due to its popularity in current embedded computing platforms, particularly on devices of Nvidia coporation shown in figure 5.1. The algorithm data flow was analyzed related to each functional block. The evaluation methodology consists on the identification of blocks consuming significant processing time or having a low data dependency. During the experiments, we found that Features Matching block has a high computational cost but could be parallelized thanks to its independence in data flow. In this functional block, each point correspondence in an image will be found by comparing the HOOFR 256-bits descriptor of this point to that of each point in the other image. For a high localization precision, a large number of points detected is required, leading consequently to a high matching cost. However, processing of each point is not related to others, so a parallelization can be performed. Otherwise, HOOFR features extraction is also accelerated using OpenMP to exploit all the computing cores of the CPU.

Figure 5.1: CPU-GPU embedded platforms

5.1 Overview

The requirement of SLAM algorithms in terms of calculation, accuracy and embeddability is a critical factor limiting the use of existing approaches in embedded applications. Meanwhile, trends towards low cost implementations and low power processing require massive parallelism and implementation on heterogeneous architectures. The implementation of SLAM algorithms in this case is often preceded by an algorithm-architecture-mapping study, which allows formal verification as soon as possible to warrant the feasibility of the design and to reformulate optimization problems so as to exploit at the best the target architecture. Author of [61] analyzed the acceleration of a laser SLAM on two desktop GPUs: GF8400M and GTX280. The speed-up factors achieved are respectively 8 and 57 in comparison to the execution on a T7250 CPU (@2GHz). More recently, Whealan et al. [115] evaluated their approach for a dense visual SLAM based RGB-D camera on a powerful system consisting of an Intel CPU (i7 - 3.4GHz) and an Nvidia GeForce GTX 780 GPU. A fast execution is achieved where the average time ranges from 31ms to 45ms per frame. Heterogeneous architectures (CPU-GPU, CPU-DSP or CPU-FPGA) are a common trend nowadays on computing platforms, specially for embedded systems. Therefore, many researchers took advantage of these architectures to accelerate SLAM applications. B. Vincke et al. [116] proposed an efficient EKF-SLAM system based on a low-cost and heterogeneous architecture. The hardware contains an ARM processor, an SIMD coprocessor (NEON) and a DSP core. The system implements low-cost sensors: a camera and odometers. Abouzahir et al. [117] also provided a case

study of the FastSLAM 2.0 algorithm on different embedded architectures. However, although the real-time performance is announced, the consistence of the algorithm need to be verified on more datasets. Some other works have been presented in section 1.3.1. The emergence of embedded systems has lead to several works addressing the embeddability issue of SLAM algorithms. However, few works deal with hardware-software mapping of visual SLAM algorithms on embedded architectures. The appearance of the recent heterogeneous architectures should lead to a great improvement in designing visual SLAM systems.

5.2 GPU programming

This work presents the system developement on a CPU-GPU architecture. CUDA and OpenCL are two well-known languages for GPU programming. OpenCL is supported by several high-end GPUs (NVIDA, AMD, Intel, etc..). It is also a framework for programming across various heterogeneous platforms such as: CPU-GPU, CPU-DSP or CPU-FPGA. Otherwise, CUDA is less flexible when it is only supported by NVIDIA hardware. In some powerful embedded platforms (Tegra K1, X1, X2), NVIDIA supports only CUDA programming.

5.2.1 GPU thread organization

The paradigm of OpenCL processing contains a notion of “kernel”. A kernel is a subroutine or mini-program. Kernels are the parallel programs to be run on the device (the GPU inside the host system). A number of primitive “work-items” will simultaneously execute a kernel program. The number of all the work-items is equal to the global work size which is conceptually organized into 1D, 2D or 3D arrays of work-items for convenience. The global memory and the constant memory are shared across all the work-items. Batches of these primitive work-items can also be organized into “work-groups” for each dimension respectively, which forms the local-work-size. Users should define the specific local-work-size of a work-group based on the amount of available local memory, as well as the memory access latency, depending on the architecture constraints. Each work-item within a work-group can communicate efficiently using the local memory scoped to others in the

same work-group. Using this local memory, all work-items within a work-group can also be synchronized.

The paradigm of CUDA processing has a similar characteristic to that of OpenCL with a little changes in the naming. Work-items and work-groups are replaced respectively by threads and thread-blocks. Moreover, while OpenCL defines directly the global-work-sizes which must be multiple by local-work-size, CUDA does the opposite by defining the number of local works (number of thread-blocks).

5.2.2 GPU memory hierarchy

There are several levels of memory on the GPU device as shown in figure 5.2, each with distinct read and write characteristics. Memory model seen by OpenCL and CUDA is divided into two parts:

- Host Memory: a memory directly available to the host. Memory objects move between the Host and the devices through functions within the API or through a shared virtual memory interface.
- Device Memory: a memory directly available to kernels executing on devices.

For device memory, OpenCL and CUDA have equivalent models with a little bit changes in terminology presented in table 5.1. The following description is intended for OpenCL, the notion of CUDA can be inferred easily. In fact, device memory consists of four named address spaces or memory regions:

1. Global Memory: this memory is located off-chip on the main GDDR memory module which therefore has the largest capacity but is the most costly to interact with. It permits read/write access to all work-items in all work-groups running on any device within a context. Work-items can read from or write to any element of a memory object. Reads and writes to global memory may be cached depending on the capabilities of the device.
2. Constant Memory: a region of global memory that remains constant during the execution of a kernel-instance. The host allocates and initializes memory objects placed into constant memory.

3. Local Memory: a local region of memory related to a work-group. Every work-item in a work-group also has access to a unified local memory, shared among all work-items for the life of that work-group. This memory region can be used to allocate variables that are shared by all work-items in that work-group.

4. Private Memory: a private region of memory related to a work-item. Every primitive work-item has access to private memory as well as registers. This memory is really a misnomer meaning that the memory is private to the work-item, it is not stored in the work-item's registers but rather off-chip in the global GDDR memory available on the graphics card. Variables defined in one work-item's private memory are not visible to another work-item.

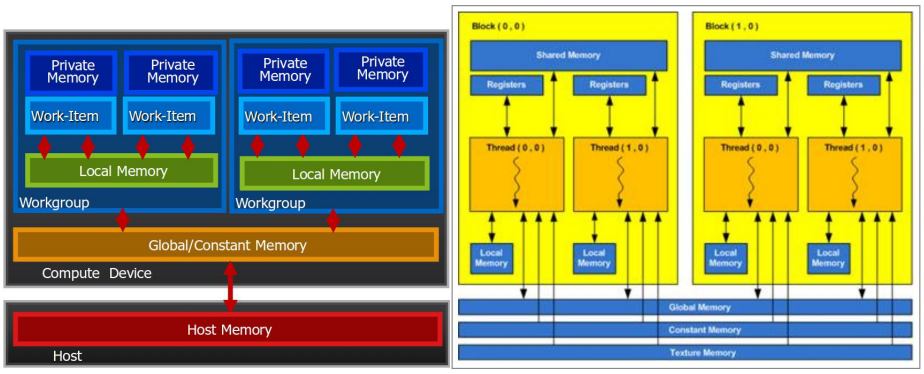


Figure 5.2: GPU memory model. Registers and private memory are unique to a work-item, local memory is unique to a work-group. Global, constant, and texture memories exist across all work-groups

The global, constant and texture memory are optimized for different memory usage models. Global memory is not cached, though memory transactions may be coalesced to hide the high memory access latency. These coalescence rules and behaviors are dependent on the particular device used. The read-only constant memory resides in the same location as global memory, but this memory may be cached. On a cache hit, regardless of the number of threads reading, the access time is that of a register access for each address being read. The read-only texture memory also resides in the same location as global memory, and is also cached. Texture memory differs from constant memory in that its caching policy specifically exploits 2D spatial locality.

OpenCL	CUDA
Work-Item	Thread
Work-Group	Thread Block
Multi-dimension Range (NDRange)	Grid
Global / Constant Memory	Global / Constant Memory
Local Memory	Shared Memory
Private Memory	Local Memory

Table 5.1: OpenCL vs CUDA Terminology

5.3 HOOFR SLAM mapping on a CPU-GPU architecture

Our algorithm pipeline is recalled in detail in figure 5.3. After HOOFR features extraction, we launch at one time the Loop detection and Mapping threads. Inside Mapping thread, Features Matching block finds the correspondences for each key-point of the current frame in each PNF. We offload this block to GPU due to its computational cost. Then, a number of Relative Pose Computation tasks are executed, each of them computes one predicted camera pose from one PNF. The number of PNFs ($nframes$) hugely depends on the camera movement speed. However, in practice, due to the architecture constraints, $nframes$ is fixed to 3 or 4 for the maximum number of neighbor frames. The "Optimal Pose Extraction" block evaluates the predictions to get an optimal current pose. Otherwise, inside Loop detection thread, Image Description block describes the current frame by comparing the descriptions of relevant key-points to a bag of words (BoW). The image description is then passed to Frame Checking block to find the max-likelihood in key-frames set. We define an overlapped frame as a frame having a max-likelihood near to it in pose graph with high matching score. Normally, when we have a new key-frame, some of the following frames could be overlapped frames. Features Matching and Relative Pose Computation between current frame to max-likelihood key-frame in case of potential loop are processed by stricter condition than that of Mapping thread to warrant an accurate loop closure. "Map Processing" block gathers the result of two main threads (Loop detection and Mapping). It always updates current pose and points to the map if mapping is successful, updates the key-frame set if loop detection determines that current

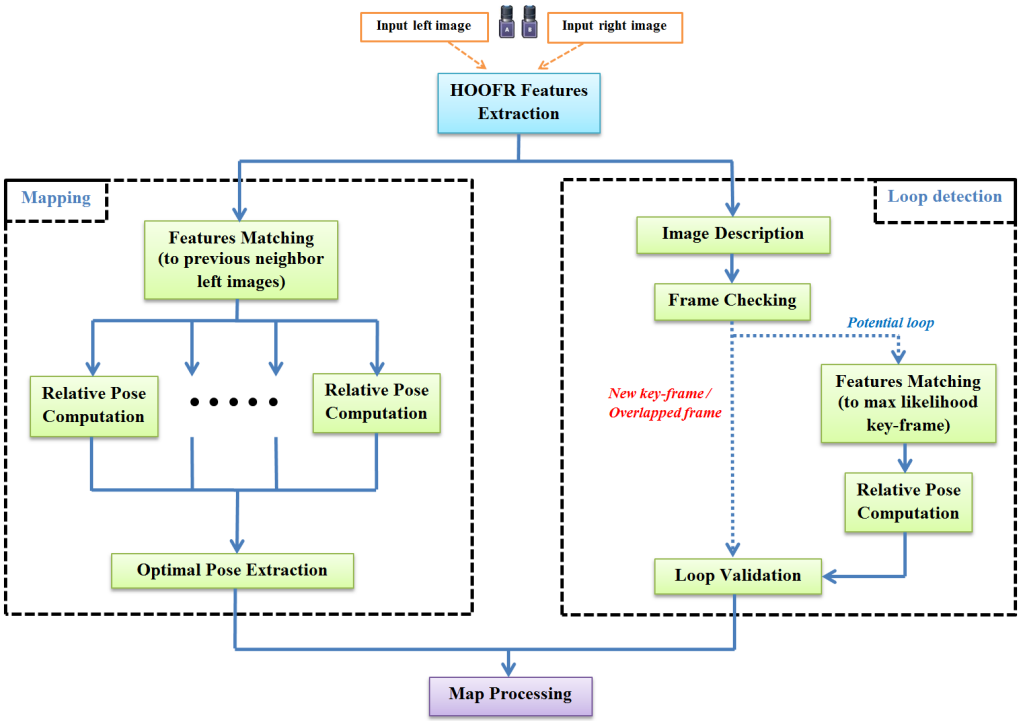


Figure 5.3: HOOFR algorithm flow

frame is a new key-frame or corrects the map by distributing error along the pose graph when a real loop is presented.

5.3.1 OpenMP Implementation of HOOFR Extraction

The HOOFR detection is more suitable to implement on CPU than GPU architecture due to 2 main reasons. Firstly, HOOFR is based on FAST detection which employs a segmentation test to accelerate feature extraction processing. In the segmentation test, a pixel can be rejected after one or two pixel tests. Such a strategy makes the difference in processing cost for each pixel (some pixels require much more time to be processed than others). Hence, it is not suitable to be implemented on a GPU architecture where each work-item requires the same complexity to make use of computation resources. Secondly, the next step after FAST detection is Hessian filtering. Hessian score is computed for all the features returned by FAST detector and then only some relevant features with the highest Hessian score are kept. This filtering is much more rapid on CPU thanks to the binary classification (used in std::nth_element function of C++ stdio.h library). However,

binary classification needs a dynamic memory allocation which is not supported on GPU. Hence, in our system, we employed OpenMP to implement HOOFR feature extraction.

There are two parts in the images: passive zone and active zone. As we select a pattern of surrounding points to make its description, passive zone is a part of image where the pixels are close to the border so that the description pattern is out of image. Passive zone is determined by $edge\_threshold$ in HOOFR descriptor and it is useless to detect key-point inside this part. Otherwise, active zone is the part where key-points could be described without doubt by HOOFR descriptor. Active zone is divided into grid. The number of cells in X and Y axes are set based on the image resolution in such a way that each dimension of one cell is about 80 – 150 pixels. The detection performing on one cell is independent from others cells.

Algorithm 5.1 OpenMP implementation of HOOFR extraction

```
//////******Detection******//////
#pragma omp parallel for num_threads(NUM_THREADS)
For each image cell do
    keypoints_cell ← FAST_Detection(FAST_THRES);
   ///***adapting detection***///
   if ( keypoints_cell.size() < PTS_PER_CELL)
       keypoints_cell ← FAST_Detection(FAST_THRES/2);
   end if
   if ( keypoints_cell.size() > PTS_PER_CELL)
       Compute_Hessian_score;
       keypoints_cell ← Retain_relevant_points;
   end if
end for
//////******Key-points Regrouping******//////
   [Points_Distribution, keypoints_set] ← Regroup_keypoints;
//////******Description******//////
#pragma omp parallel for num_threads(NUM_THREADS)
For each keypoint in keypoints_set do
    Compute_keypoint_descriptor;
end for
```

HOOFR detection is demonstrated on the first part of algorithm 5.1. Each OpenMP thread processes an image cell and individual key-point sets are created for each cell to assure data independence. $NUM\_THREADS$ represents the number of cells handled in parallel. We assign a value to $NUM\_THREADS$ by the total number of cores inside the processors to make use of computing resources. A great value of $NUM\_THREADS$ is meaningless in practice since a maximum parallelism was employed. In each cell, FAST detection is performed with adapting threshold. Then we extract the relevant key-points

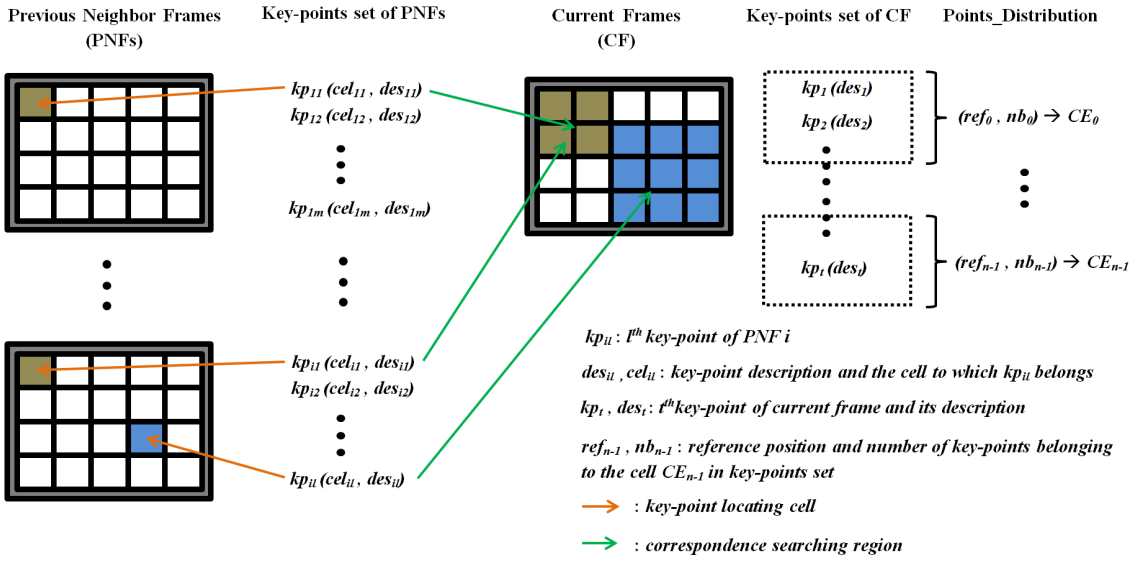


Figure 5.4: Matching strategy

corresponding to the highest HESSIAN scores. At the end of detection phase, all key-points are regrouped in one global set to which a structure defined as *Points_Distribution* is attached. This structure represents the distribution of key-points in the image and is later required in Matching block to specify the searching regions. We chose the static mode for OpenMP scheduling instead of the dynamic mode. The reason is that computational complexity in each thread is comparable to that in another thread. Static mode is hence more suitable in which the chunks can be scheduled to threads during compilation while dynamic mode is not efficient due to the more locking.

Similar to the detection phase, features description is also parallelized using OpenMP but the strategy is modified. We note that the number of key-points detected in each image cell is not constant. Specially, when non-texture parts appear in the scene, some image cells contain very few key-points in comparison to other cells. If we keep the parallelism on image cell level, the threads handling many key-points will be extremely more costly than the threads with few key-points. In such case, some computing units finish the work too fast and have wasting time to wait the others. To avoid this issue, we propose to use OpenMP at key-point level as shown on the second part of algorithm 5.1. Orientation and description of each key-point are extracted without dependence on any another key-point. The same complexities are presented for all threads leading to an efficiency in work distribution among computing units.

5.3.2 GPU implementation of Features Matching

In features matching of HOOFR SLAM, we benefit from all kinds of GPU memory to have an optimized implementation. In order to make HOOFR SLAM works on several architectures, we developed Features Matching block in three versions: OpenCL and CUDA versions running on a GPU and a standard C++ version running on a CPU. CUDA uses the same manner to observe GPU memory but with a little change in naming: *global memory*, *shared memory* (corresponding to *local memory* in opencl) and *local memory* (corresponding to *private memory* in opencl). The CUDA programming is also similar to that of OpenCL. Hence, in the following, we only detail the implementation in OpenCL while the others could be deduced easily.

To implement features matching on GPU, key-point information must be transferred to the GPU global memory. As shown in figure 5.4, two parameters (cel, des) are required for each key-point in PNFs. cel is in the form of integer number corresponding to the cell where the key-point is located. It takes the values from 0 to (n-1). Besides, des is 256-bit HOOFR description of the key-point. In practice, des is performed using a matrix having 1 row and 32 columns with 32 elements of type "unsigned char". To regroup all parameters for PNFs, we create two matrices as in equations (5.1, 5.2) where Pnf_Cels and Pnf_Dess are respectively in dimension of (pnf_np x 1) and (pnf_np x 32), pnf_np is the total number of key-points in PNFs.

$$Pnf\_Cels = [cel_{11}\ cel_{12}\ ...\ cel_{1m}\ .....\ cel_{i1}\ cel_{i2}\ ...\ cel_{il}]^T \tag{5.1}$$

$$Pnf\_Dess = [des_{11}\ des_{12}\ ...\ des_{1m}\ .....\ des_{i1}\ des_{i2}\ ...\ des_{il}]^T \tag{5.2}$$

For the current frame, two parameters are also taken into account. Firstly, we create the Cur_Dess matrix having the dimension of (cur_np x 32) for key-point description. cur_np is the number of key-points in current frame. Similar to Pnf_Dess, each row of Cur_Dess serves as one 256-bit description based on 32 unsigned char numbers. Secondly, key-points set of current frame is organized by the order of image cell so that a structure denoted as $Points\_Distribution$ is employed. This structure is transformed into an integer matrix with the dimension of (N_CELLS x 2) while N_CELLS is the number

of image cells. In *Points_Distribution* matrix, each row corresponds to the distribution of one cell in the whole key-points set: the first element ref is the position where the first key-point of the cell is located in the whole set, the second element *nb* is the number of key-points of the cell.

In practice, Pnf_Dess, Pnf_Cels, Cur_Dess and $Points\_Distribution$ matrices are transferred to GPU_Pnf_Dess, GPU_Pnf_Cels, GPU_Cur_Dess and $GPU\_Points\_Distribution$ respectively on GPU global memory. These memory parts are set to "read-only" to don't be changed by any work-item. Moreover, we also allocate on GPU global memory a "write-only" integer matrix referred as $GPU\_Correspondence$ (pnf_np x 3) on which matching result is returned. We note that all input matrices are aligned to 1-D array on the GPU memory since GPU programming do not support pointer-to-pointer variable.

A natural implementation at our first try is that we process the whole matching of one key-point on one work-item. However, by this naive approach, we encountered the "overhead computation" problem. In fact, when the kernel has too high computational cost, the kernel execution takes too much time to complete one work-item. At this time, the "watch-dog" in GPU driver considers that GPU is idle since there is no feedback from kernel during an amount of time. This confusion leads to the GPU frequency reduction which severely decreases GPU timing performance. Therefore, in order to avoid such issue, we keep the kernel light by splitting the matching of one key-point into several work-items. In practice, we search the correspondence in the current frame at the same cell and neighbor cells as mentioned in figure 5.4. The searching on one cell is rapid due to a small number of key-points so that it is suitable to be operated on one work-item.

Algorithms 5.2 and 5.3 show the calling function on CPU and the kernel running on GPU for feature matching in mapping thread. The main idea is to use 9 work-items in a work-group to find correspondence in 9 neighbor cells of current frame. In kernel, $cell\_id$ variable is the index of image cell where the work-item performs the searching. $cell\_id$ is one neighbor cell so that it is determined by local_id *kc* of the work-item and image cell $GPU\_Pnf\_Cels[i]$ of the PNF key-point. Keypoints locating in the image cell $cell\_id$ of current frame are classified from position ref_l to position ref_h in the key-points

set. Besides, *dist_min* and $trainIdx$ correspond respectively to the distance and the index in key-points set of the first matching, while *dist_min*2 is the distance of the second matching. Opencl local memories are allocated to save 9 local results and are synchronized by **barrier** function. After the synchronization, only one of these 9 work-items ($kc = 0$) continues handling the local results to extract the final matching. It specifies final *dist_min* and *dist_min*2 from local results and validates the matching if the ratio $dist\_min/dist\_min2$ is lower than *0.85*. BLOCK_SIZE represents the number of PNF key-points processed also in the same work-group. Thus, *local_work_size* is assigned to {BLOCK_SIZE, 9}. The value of BLOCK_SIZE depends on many factors defined in GPU architecture such as the maximum *local_work_size* in each dimension or the local memory capacity. In our implementation, BLOCK_SIZE is set to 16 which provides a good performance. OpenCL programming claims that *global_work_size* must be a multiple of *local_work_size* in all dimension. Hence, the first dimension of *global_work_size* must be the nearest multiple of BLOCK_SIZE that is greater or equal to pnf_np. The work-items having the global identification bigger than pnf_np will be stopped rapidly after the test at the first line in kernel. The second dimension of *global_work_size* takes the value of 9 similarly to the second dimension of *local_work_size*.

Algorithm 5.2 Calling function on host (CPU)

```
function Matching
.....
workitems =
        (pnf_np+BLOCK_SIZE-1)/BLOCK_SIZE*BLOCK_SIZE;
global_work_size[] = {workitems,9};
local_work_size[] = {BLOCK_SIZE,9};
clEnqueueNDRangeKernel(cmd_queue, matching_kernel,
2, NULL, global_work_size, local_work_size,
0, NULL, NULL);
clFinish(cmd_queue);
.....
end function
```

GPU programming is also employed for features matching in loop detection thread and we use the same approach as in mapping thread to find correspondence. However, matching conditions have a little changes leading to some modifications in matching kernel. Firstly, due to the fact that "cross BruteForce" is used, only the last matching will be

searched in each matching direction. *dist_min*2 will not be considered so that we do not need to allocate GPU memory to save it. After barrier function, the process is also simpler when only the last matching is extracted from 9 local ones. Secondly, in CPU calling function, two kernel calls (**clEnqueueNDRangeKernel**) are required: one for "*current frame* to *max_lilkelihood frame*" key-points matching and the second is for the opposite direction "*max_lilkelihood frame* to *current frame*". On the other hand, BLOCK_SIZE still keeps the value of 16. *local_work_size* and *global_work_size* in each kernel call are computed by the same manner as used in mapping thread CPU call.

Figure 5.5 presents the CPU-GPU mapping of the algorithm. In order to avoid memory access conflict, mapping and loop detection thread work on separate zones of CPU memory. Each zone is pinned respectively to that of GPU global memory where the corresponding matching kernel is performed. Memory pin also allows to active DMA high-bandwidth data transfer between CPU and GPU.

Algorithm 5.3 OpenCL matching kernel on device

declare global arrays: *GPU_Pnf_Dess, GPU_Pnf_Cels, GPU_Cur_Dess, GPU_Points_Distribution, GPU_Correspondence;*
function KERNEL: MATCHING
declare 3 local arrays: DIS_min[9*BLOCK_SIZE elements], DIS_min2[9*BLOCK_SIZE elements], MatchingId_min[9*BLOCK_SIZE elements];
i ← *get_global_id*(0);
ki ← *get_local_id*(0);*////from 0 to BLOCK_SIZE-1*
kc ← *get_local_id*(1);*////from 0 to 8*
////identify neighbor cell
cell_id ← **Get_Neighbor_Cell_ID**(*GPU_Pnf_Cels*[*i*],*kc*);
//// Get keypoint descriptor
point_pnf_des ← **Get_Keypoint_Descriptor**(*GPU_Pnf_Dess*[32∗*i*]);
////Get local matches to from the neighbor cell
{DIS_min[9*ki+kc], TRAINIdx_min[9*ki+kc], DIS_min2[9*ki+kc]}
← **Find_Local_Best_And_Second_Matches**(*point_pnf_des*, *GPU_Cur_Dess*, *GPU_Points_Distribution*);
barrier(CLK_LOCAL_MEM_FENCE);
*///**At this point, local matching result of 9 neighbor cells are saved at the positions from 9*ki to 9*ki+8 **///*
if (kc==0)
GPU_Correspondence ←
Find_Global_Matches(DIS_min, TRAINIdx_min, DIS_min2);
end if
end function

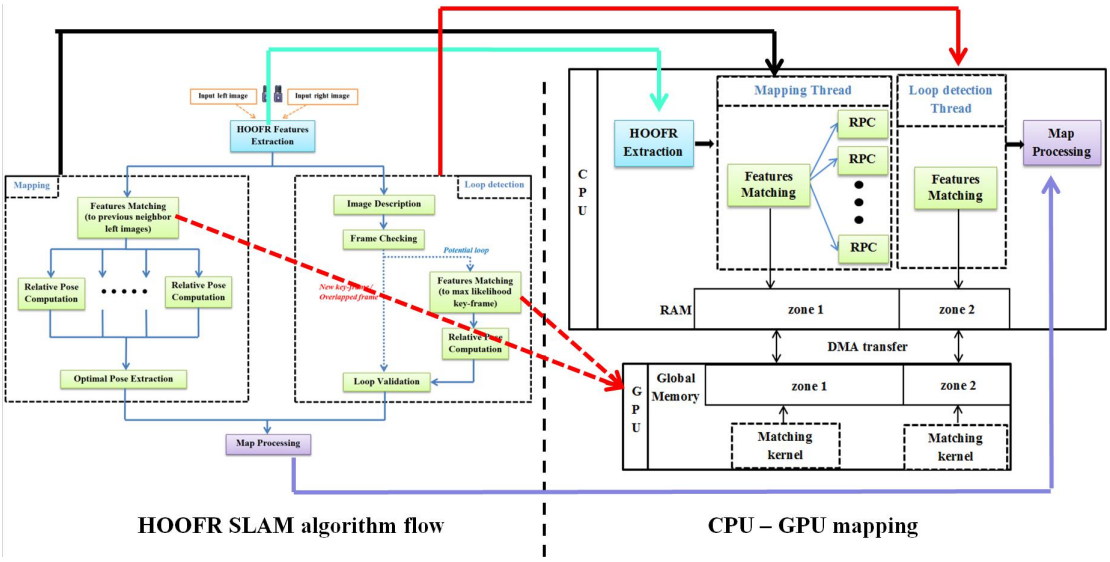


Figure 5.5: CPU-GPU Mapping

Table 5.2: Architecture specifications (JETSON Tegra X1 embedded system vs Powerful Intel PC)

	TX1	Intel PC
CPU	4-cores ARM A57 4-cores ARM A53	8 intel cores i7
CPU clock rate	1.3-1.9 GHz	3.40 GHz
Cache	2 MB	8 MB
RAM	4 GB LPDDR4	16 GB
GPU	256-core Maxwell	384-core Geforce GT 740
GPU clock rate	1 GHz	1.07 GHz
Operating System	Ubuntu 14.04	Ubuntu 14.04
CUDA version	7.0	7.5
OpenCL version	-	1.2

5.4 Performances evaluation

In experiments, we have implemented HOOFR SLAM on two CPU-GPU platforms: a powerful Intel PC and an NVIDIA JETSON Tegra TX1 development system. Table 5.2 shows their specifications as a recap. Due to the fact that NVIDIA supports only CUDA for GPU programming on TX1 (not OpenCL), so that we use CUDA version of HOOFR SLAM matching block during the experiments on this board.

Table 5.3: Mean of execution time (milliseconds) using KITTI dataset for each functional block in HOOFR SLAM on the Intel powerful PC and the TX1

	$nframes = 2$			
	Intel		Tegra TX1	
	CPU (8 cores)	CPU-GPU	CPU (4 cores)	CPU-GPU
HOOFR Extraction	8.536	8.555	16.783	16.731
Mapping	52.126	27.332	119.937	99.185
Loop detection average	15.001	7.881	21.916	16.466
Loop detection cost-time	36.553	15.253	95.248	80.223
Map Processing	0.349	0.137	0.584	0.403

5.4.1 Timing evaluation

We evaluate the mean processing times of the proposed algorithm on 11-first sequences of KITTI dataset. All timings are given in milliseconds. The values are the mean of 11 sequences where timing on each sequence is also the mean of 5 launches. Table 5.3 represents the timing of each functional block in our proposal pipeline. The number of neighbor frames is 2. In the table 5.3, Loop detection average is the sum of execution time divided by the total number of frames. However, this value can not be a good representation because execution time of Loop detection thread is not constant. In fact, with an overlapped frame or in case of not enough inliers, loop detection thread is terminated rapidly. Otherwise, when loop closure is reached, this thread becomes time-consuming because the relative movement is estimated. To have a better representation, we presents "Loop detection cost-time" which is the mean time of loop detection thread when the movement estimation is performed.

We notice that Mapping thread and Loop detection thread are launched in parallel. Hence, the per-frame time is only the sum of HOOFR extraction and Mapping (the most consuming thread). Moreover, when loop closure is valid, map correction inside Map Processing is launched in an other thread so that it does not slow down the new frame acquisition. On PC Intel, without GPU implementation, the algorithm runs at ~62 ms per frame. By offloading processing to GPU, we have a better performance when the mean of execution time of the whole algorithm is decreased to 36 ms per frame.

For Tegra TX1 embedded system, it is obvious that the processing task is much slower than that of the Intel PC because of many reasons: lower frequency of CPU and GPU,

nframes	1			2			3			4		
	Min	Mean	Max	Min	Mean	Max	Min	Mean	Max	Min	Mean	Max
Intel (CPU)	33.646	58.254	78.156	36.512	65.689	82.241	38.989	76.263	96.358	43.989	81.124	98.416
Intel (CPU-GPU)	18.154	36.487	50.164	19.498	39.456	57.129	20.846	41.354	61.487	22.498	50.462	64.624
TX1	40.268	101.265	130.748	46.894	130.128	170.854	48.657	152.238	201.418	50.658	162.624	240.859

Table 5.5: KITTI-07 processing time on Intel PC and Nvidia TX1 with different values of $nframes$

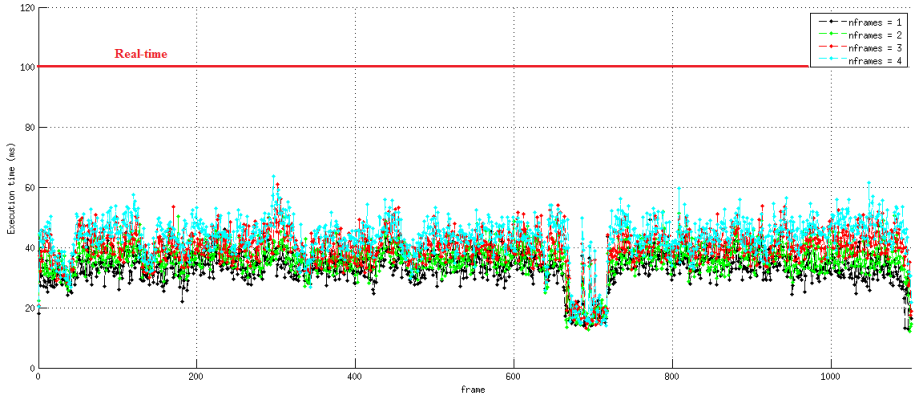
smaller cache memory resources and low number of CPU and GPU cores. On this platform, our partitioning exhibits a considerable performance where the algorithm takes in average ~116 ms per frame.

Table 5.4: Mean per-frame execution time comparison on KITTI

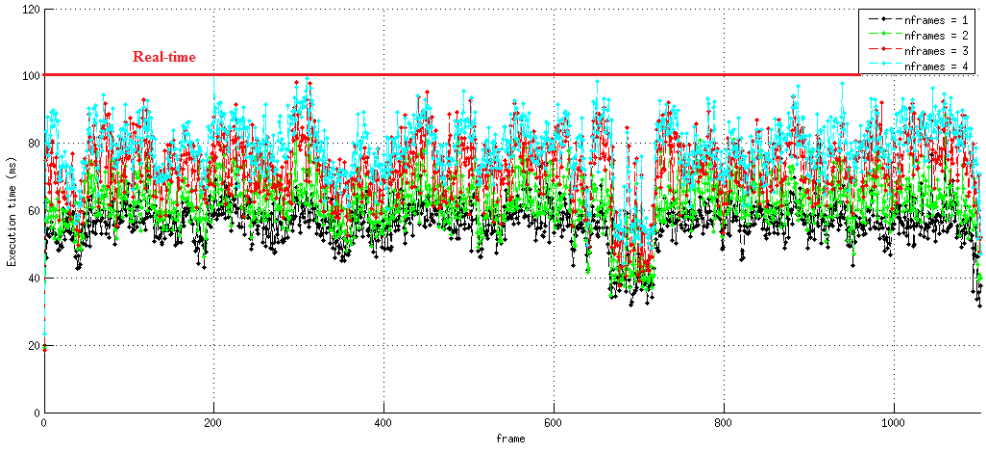
Algorithm	Execution time (ms)	
	Intel PC	Tegra TX1
Stereo ORB SLAM	69.924	190.710
CPU - HOOFR SLAM	62.235	137.235
GPU - HOOFR SLAM	36.154	116.552

We also evaluated the timing performance of ORB-SLAM and table 5.4 shows the mean per-frame timing comparison between our proposal (HOOFR SLAM) and ORB SLAM on two platforms using KITTI dataset. With Intel PC, the ORB execution time is approximately 69ms per frame (7 ms costly than CPU-only version or 32ms costly than CPU-GPU version of our algorithm). On TX1 embedded platform, ORB-SLAM takes 190 ms per frame (53 ms costly than CPU version or 74 ms costly than GPU version of our proposal).

To evaluate the timing in more details, we studied the timing and localization precision in terms of the number of neighbor frames ($nframes$). Figure 5.6 and figure5.7 present the per-frame processing time on KITTI-07 when the $nframes$ parameter changes from 1 to 4, while table 5.5 presents the minimum, the maximum and the mean values of these figures. For the powerful Intel PC, we still have a frame-rate running at less than 100ms when the $nframes$ increases to 4 for both GPU and without GPU version. For TX1 embedded system, due to limited resources, processing time could not meet the frame-rate



a) Intel PC - with GPU



b) Intel PC - without GPU

Figure 5.6: KITTI-07 per-frame processing time on Intel PC using different values of $nframes$

(10 Hz) performances. The variation of time in each frame is primarily as a consequence of the motion estimation step. In fact, in order to compute essential matrix from matching set, this step uses RANSAC scheme which selects the subset by random choices and the proportion of inliers is not identical for different matching sets. Some of high proportion of inliers normally take less time to compute than that of low proportion.

We also notice that around the 700th frame, processing times are much smaller than others. This situation occurs at the point that has the coordinates (-150,-75) in KITTI-07 trajectory. This step corresponds to a situation where the vehicle stops temporarily. In this case, the camera does not move and takes always images of the same scene. After features matching, our system found that there are so many points having the similar positions in two consecutive images so the motion estimation task is ignored and the camera is considered to keep the old position.

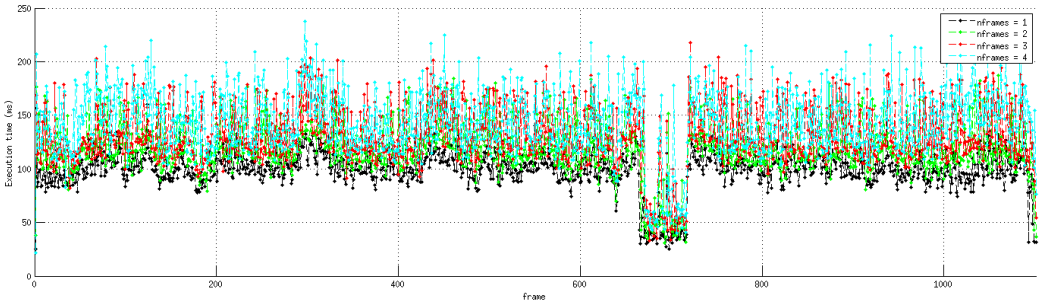


Figure 5.7: KITTI-07 per-frame processing time on TX1 (GPU implementation) using different values of $nframes$

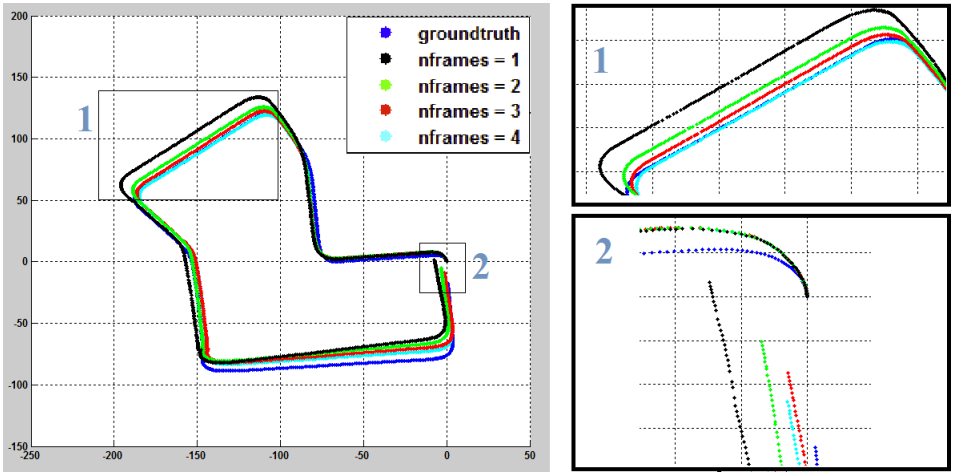


Figure 5.8: KITTI-07 localization results using different values of $nframes$

Besides, Figure 5.8 shows the effect of $nframes$ on the localization result. Ground-truth is always presented by the blue curve. We can notice that more we take into account the number of neighbor frames, more we get a higher localization precision. The explanation for this exhibition could be found at the features detection level. In fact, at some points in the trajectory, especially in turning scenarios, current frame contains less common points with nearest neighbor frame than with a further neighbor frame. Therefore, the motion estimation with further neighbor frame provides more confidence and has a higher weight. By integrating a more precise prediction in optimal pose extraction, the localization error would decrease.

5.5 Conclusion

The implementation of HOOFR SLAM on CPU-GPU architecture was obtained as a result of a hardware-software mapping study addressing feature extraction, data processing, hardware building implementation and benchmarking. The real-time algorithm implementation on high performance Intel-based PC architecture processes frames at more than 20 Hz using KITTI dataset. On the Tegra TX1 embedded system, the processing time is close to real-time performances with 6 fps running rate. In the near future, besides the algorithm optimization, the emergence of new heterogeneous CPU-GPU architectures such as Xavier Nvidia (8 Core ARM64 CPU, 512 Core Volta GPU) provides a high potential to embed the HOOFR SLAM algorithm with better timing constraints.

Chapter 6

Towards FPGA based embedded SoC architectures

6.1 Motivation

Field-programmable gate arrays (FPGAs) are attractive due to the high performance with power efficiency and low latency. These benefits are given through their massive parallel processing coupled with reconfigurability. An FPGA presents a reconfigurable set of gates on which developers can design a custom hardware accelerator, deploy it for a particular application, or reconfigure the device as a new accelerator for others applications.

On GPUs, kernels are compiled to a sequence of instructions to execute. The hardware processors are fixed and consists of cores that are specialized for common uses. Hence, with one specific kernel instruction requirements, some parts of the hardware may be unused. In contrast, on FPGAs, kernels are compiled to custom processing pipelines built on from the programmable resources such as ALMS, DSP or memory blocks. By focusing hardware resources only on the algorithm to be executed, FPGAs can offer a better performance per watt than GPUs for many specific applications.

However, one of the main challenge in FPGAs utilization is their complexity of programming. FPGAs are generally programmed using one of the hardware description languages (HDL) such as Verilog or VHDL used by hardware designers. In practice, these programming language are complex, hard to analyze and debug so that designers usually spend much time to develop an application. However, this limitation can be tackled

by a technique called high-level synthesis (HLS). HLS enables designers to program an FPGA using high-level languages (C, C++, SystemC or OpenCL). This in turn reduces both verification and design time in comparison to HDL.

FPGAs are inherently parallel, so they are naturally suitable for OpenCL's parallel computing capabilities. FPGAs offer a pipeline parallelism where tasks can be spawned in a push-pull configuration with others tasks using different data from the previous task with or without host interaction. OpenCL allows to develop the code in the familiar C programming language with the additional capabilities provided by OpenCL. The developers can send kernels to FPGAs without having to learn the low-level HDL coding. Generally, there are several benefits for software developers and system designers to use OpenCL to develop code for FPGAs:

- Ease of development: OpenCL keeps us at a higher level of programming, making our system open to more software developers because most of them are familiar only with the C programming language, but not low-level HDL languages.
- Code profiling: using OpenCL, we can profile our code and determine the performance-sensitive parts that could be hardware accelerated as kernels in an FPGA.
- Efficiency: the FPGA has a fine-grain parallelism architecture, by using OpenCL we can generate only the logic needed to deliver one fifth of the power of the hardware alternatives.
- Flexibility: with OpenCL, we can develop kernels that can switch simply between different types of target (FPGAs, CPUs, GPUs, and DSPs). It seamlessly give us a truly heterogeneous system design.
- Extended code life: code reuse is often an ambitious goal for software and system designers. OpenCL kernels allow us to carry the developed code on different families and generations of FPGAs from one project to the next.

These reasons above encourage us to study the use of OpenCL based FPGA-soc architectures in embedding SLAM applications. In this chapter, we present our work on implementing the front-end part (feature extraction) of HOOFR SLAM system on a FPGA based SoC architecture.

6.2 Related works and contribution

In the state of the art, several researches have previously investigated the acceleration of feature extraction using FPGA. In 2009, Yao et al [118] proposed an optimized architecture for SIFT feature detection running at 31ms per frame (640x480) on Xilinx ML507 FPGA. In 2010, Bouris et al [119] implemented SURF detector on Xilinx Virtex 5 XC5VFX130T FPGA that processed at 56 fps (~18 ms per frame) with the same resolution. The limitation of these work is that they studied only the detection task on hardware while the description task was out of the scope. In 2013, Chiu et al [120] designed a parallel hardware for the whole SIFT extraction. The algorithm is modified to reduce computational amount by 90% and memory usage by 95%, running at 30 frames per second with VGA resolution.

Due to the fact that SIFT and SURF are floating computation, the hardware design of these algorithm performs a slower speed than binary algorithm such as FAST, ORB, ... Lee [121] presented an ORB extraction system in 2014 that operated at 108 fps for 640x480 images. This system however did not consider the whole ORB algorithm when missing Harris filtering step. An other ORB system is proposed by Weberruss et al [122] in 2017, running on an Altera Arria V with throughput equivalent to 72 fps at 1920x1080 or 488 fps at 640x480. Despite mentioning ORB, they employed Harris algorithm for detecting keypoints. It is not a raw idea of ORB which uses Harris score to filter keypoints only after FAST detection. An alternative of ORB implementation on FPGA was presented by Sun [123] where the performance is 42 fps with 1000 features with full-HD images. The proposed architecture is tested on a Zynq-family FPGA.

For OpenCL programming, there are many researches investigating the FPGA acceleration by OpenCL on various algorithms. As an example, Pu et al [124] experiments KNN algorithm on FPGA-based heterogeneous architecture. OpenCL is used to program

Stratix IV 4SGX530 FPGA from Altera. The performance was compared to Intel Core i7-3770 processor and an AMD Radeon HD7950 graphics card where the authors found that FPGA-based implementation was more power efficient. In 2017, Muslin [125] evaluated the OpenCL implementation on Xilinx Virtex-7-series FPGA of three well-known algorithms: KNN, Monte Carlo for financial models and Bitonic sorting. A comparison in terms of execution time, energy and power consumption with some high-end GPUs is done as well. The author also concluded that FPGAs are much more energy-efficient in all the test cases and can sometimes be faster than GPUs.

Nevertheless, to the best of our knowledge, there is not a whole system of feature extraction implemented on a FPGA using OpenCL programming until the present. Moreover, all designed systems above are developed for naive implementations. For a SLAM application, it is not enough to have a high precision. In practice, almost SLAM systems used bucketing method to extract keypoints from image [88, 26, 35]. None of the researches above however considered this method into an optical flow approach. Due to these reason, in this chapter, **our contribution** can be stated as following:

- Design a feature extraction system dedicated for SLAM application taking into account the bucketing method.
- Use OpenCL programming to implement the system on FPGA-based heterogeneous architecture.
- Our system use HOOFR extractor, our previous proposal published in [111] due to its robust performance.

6.3 OpenCL programming advantages on FPGA

The main difference between launching kernels on GPUs and on FPGAs is how the parallelism is handled. GPUs are known as single-instruction, multiple-data (SIMD) devices where a set of processing elements perform the same operation on their own individual work-items. On the other hand, FPGAs exploit pipeline parallelism when different stages of the instructions are applied concurrently to different work-items.

A question arises as a result of this difference in parallelization methods: how branching is managed. When branching occurs on a GPU, it is still necessary for all work-items within the same SIMD unit to correctly execute the various branches. However, because the SIMD unit as a whole operates on a single instruction at a time, all code-paths taken by the individual work-items must be executed one after one, with individual work-items disabled or enabled based on how they evaluated the branching condition. As a result, encountering a branching condition with N options could potentially result in execution time equal to the sum of execution times for all N options (for N up to the SIMD width). On the other hand, branching is less of an issue on FPGAs because all code-paths are already established in hardware. All branch options can be executed concurrently or even computed speculatively to allow overlap with branch condition computation.

OpenCL on FPGAs presents the advantage of I/O channels and kernel channels (OpenCL 2.0 pipes): an optimization that is not currently implemented on GPUs. Kernel channels allow kernels to transfer data via a first-in-first-out (FIFO) buffer and without the host interaction. Traditionally, when a GPU wants to transfer data from one kernel to another, it must reads and writes to global memory combined with some synchronization methods. The removal of these intermediate reads and writes on FPGA allows us to achieve performance and power efficiency gains. Moreover, Altera FPGAs also extend the idea of kernel channels even further to allow I/O interfaces (I/O pipes) allowing kernels to access directly from a streaming interface without host interactions. It is also known as IO channels. In practice, the host can effectively configure the data pipeline and then steps out of the data path. Figure 6.1 illustrates a kernel being executed on three sets of data coming from an I/O source. Significant time savings are possible because the FPGA communicates directly with the I/O source, and no longer needs the host to serve as a middle-operator as in GPU.

SIMD-based parallel processing is suitable for dealing with loops when there are no dependencies across iterations of the loop. In that case, parallelization can occur by simply mapping work-items to individual loop iterations. However, in most real applications, data-dependencies are unavoidable to the structure of the algorithm, and cannot be removed easily. In order to ensure correct computations, GPU programmers must rely on relatively complicated constructs involving resources shared by work-items in a work

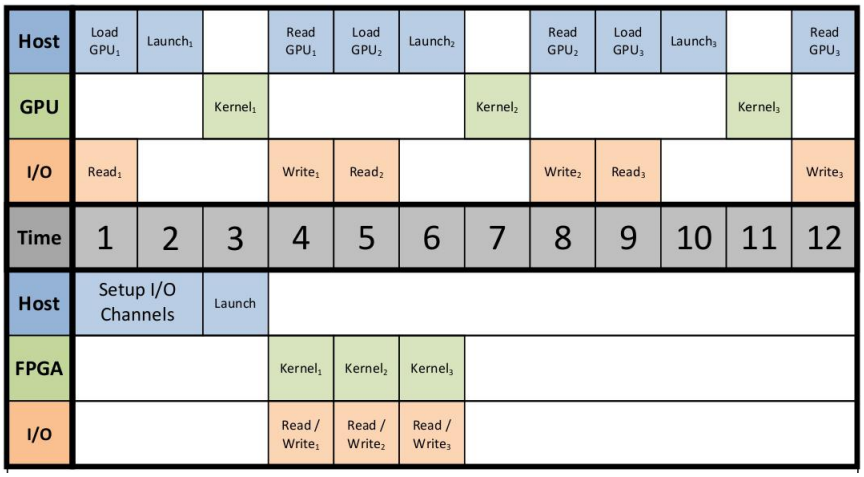


Figure 6.1: OpenCL based FPGA channel benefits

group along with synchronization primitives. GPU programmers could alternatively handle the data-dependency section of work by the method of only a single work-item (also called an OpenCL task) but it will hamper parallelization and overall performance due to the idleness of other processing cores. In contrast, pipeline-parallel devices such as FPGAs have less of issue dealing with single work-items because single work-items are actually the unit of work in the pipeline anyways. In fact, FPGA can achieve additional performance by pipelining iterations of a loop which contains loop carried dependencies. It means that the next iteration will be launched as soon as loop dependencies are completed. This scheduling is built primarily by the compiler. Besides, loop pipelining performance can also be improved by software developer in a number of ways such as removing some dependencies, simplifying dependence complexity or relaxing dependencies. Removing dependencies can be realized by using simple access patterns results in faster launch times for the next iteration. Similar results occur when avoiding expensive operations when computing loop-carried values. Relaxing dependence increases the number of iterations between generation and use of a value, which means that the immediate next iteration can be launched sooner. In Altera OpenCL tool, setting the kernel attribute "task" informs the compiler that the kernel will run with a single work item.

OpenCL program is implemented on ALTERA FPGA using AOCL tool and our design flow is shown in figure 6.2. In the first step, host and kernel codes are developed in parallel to warrant the conjunction between kernel interface and kernel calling of the host. Then, the functional verification is done using FPGA SDK for OpenCL emulator.

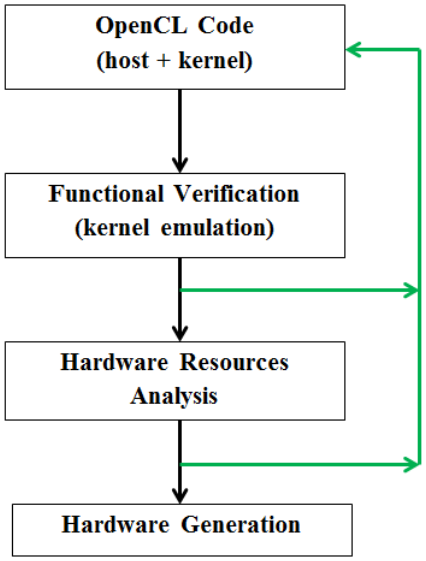


Figure 6.2: Design flow

This feature allows us to test the functionality and iterate on the design without executing it on FPGA material each time. The emulation of our design is run on x86-64 ubuntu 14.04. Once the functionality is verified, the hardware resource usage for all kernels are estimated. This step requires a specific FPGA architecture to be defined. After the estimation, in the case that kernels take too much resources or the design is not suitable to be implemented on a target platform, we return to the first step to modify and optimize the design. Finally, hardware implementation is generated and is loaded to the target board to validate performances.

6.4 HOOFR extractor partitioning an a CPU-FPGA architecture

HOOFR extractor is divided into 4 functional blocks (FAST, HESSIAN_COMPUTE, FILTERING and DESCRIPTION) with respect to the algorithm process. This decomposition is based on an analysis of the data flow to achieve a compromise between consuming resources (memory, logic elements) and processing speed. For details, FAST block is to detect FAST features in the images. HESSIAN_COMPUTE is to compute hessian score for all keypoints returned by FAST detector. Keypoints are then filtered in the FILTERING block to keep the relevant ones based on their hessian scores. Finally,

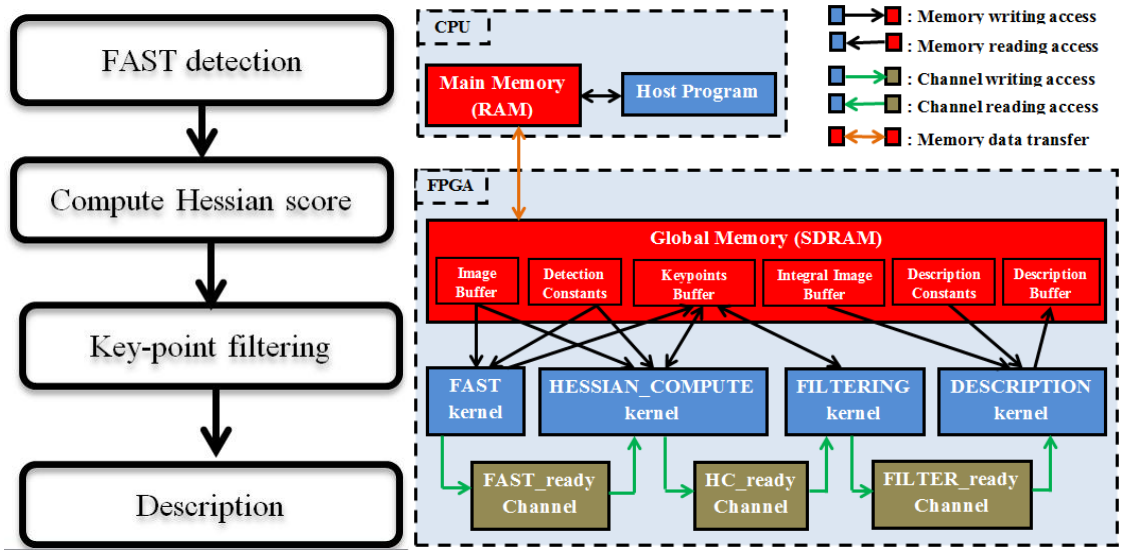


Figure 6.3: HOOFR extractor architecture

DESCRIPTION block builds 256-bit HOOFR descriptor for all relevant keypoints after the filtering.

CPU-FPGA system for HOOFR extractor is shown in the figure 6.3. Four blocks are implemented on FPGA for the pipelining. Each functional block is programmed as one kernel and all kernels are lauched concurrently. CPU plays a role of a controller and computes integral image required in description block. Noting that the computation of integral image is irrelevant to be executed on FPGA device in OpenCL design as it could be realized rapidly by one query pixel-to-pixel. Otherwise, this operation is suitable on CPU side. Hence, a partitioning is proposed as demonstrated in figure 6.4 in order to make use of the computing resources. As we can see, for each input image, CPU firstly transfers the image to FPGA global memory. Then, CPU launches consecutively the three detection kernels. The DESCRIPTION kernel will be launched only when integral image has already been computed and transfered to FPGA from CPU. This partitioning allows us to employ CPU and FPGA resources in parallel. There is no interruption on CPU after lauching kernel. A synchronization occurs only when all kernels are active and CPU awaits until FPGA finished the extraction. This synchronization is present to ensure that valid results are ready to be reloaded to CPU from FPGA.

To have a high precision in SLAM applications, bucketing detection is always employed to warrant the homogeneous keypoints distribution. It means that image is divided

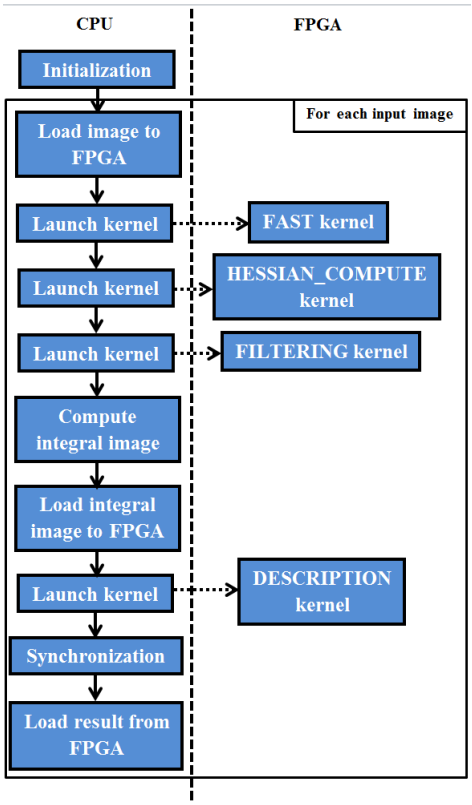


Figure 6.4: CPU-FPGA execution planning

into grid and a specific number of keypoints is aimed to be extracted for each image cell. Hence, the pipeline is realized at the level of image cells. When a kernel finishes its work for one image cell, the next kernel starts to work immediately on this image cell as shown in figure 6.5. The communication control between kernels is done using altera channel extension for passing data and for synchronizing kernels with low latency. The implementation of channel allows kernels to communicate directly with each other via FIFO buffers. Unlike the typical OpenCL model, data movement across kernels is coordinated without host intervention.

OpenCL does not warrant the execution order of work-items. Therefore, the execution order of image cells is undefined. In figure 6.5, three cells a, b, c do not correspond to cells 0, 1, 2 in the image. In practice, when FAST kernel is finished on cell “a”, it writes the identification of cell “a” to FAST_ready channel. HESSIAN_COMPUTE kernel reads this identification and launches the processing for cell “a”. The procedure continues by a similar way for other kernels. We denote this design as “pipeline of pipeline” due to

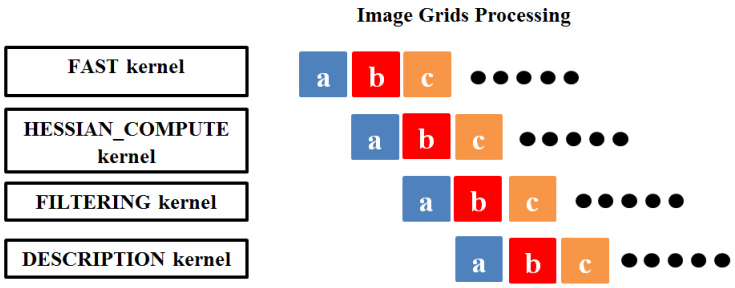


Figure 6.5: Pipeline kernel processing

the fact that inside each kernel, work-items are also parallelized following the pipeline natural characteristic of FPGA.

6.5 HOOFR architecture design

6.5.1 FAST kernel

The intensity of these 16 pixels are compared to the intensity of reference pixel. Each comparison takes one of three states: darker, brighter, not darker, not brighter. In practice, the smallest data type supported in OpenCL programming is 8-bits (char or unsigned char). Hence, we put the comparison result of 16 pixels into 4 elements (fast8_d1, fast8_d2, fast8_b1, fast8_b2) of the 8-bits type. Each bit of fast8_d1 and fast8_d2 performs that the pixels are darker (value = 1) or not (value = 0) while each bit of fast8_b1 and fast8_b2 shows that the pixels are brighter (value = 1) or not (value = 0).

The advantage of FAST detection is that the segmentation test could be employed to accelerate the processing. It means that the feature verification could proceed to some tests to ignore rapidly a pixel. HOOFR extraction used FAST-9 where a central pixel is considered as a feature when it is darker (dark feature) or brighter (bright feature) than at least 9 consecutive points in Bresenham circle. The segmentation test can be done with 8 symmetric pairs. In fact, if a central pixel is a dark feature in FAST-9 , the central point must be darker than at least one of two pixels in a symmetric pair. It is applied for all 8 symmetric pairs in Bresenham circle. The condition is similar to the case of bright feature.

OpenCL implementation of FAST detection on FPGA is shown in algorithm 6.1. After the segmentation test, central pixel is ignored in case of negative sign (dark = 0 and bright = 0). Otherwise, when positive sign is found (dark = 1 or bright = 1), two 8-bit variables will be concatenated to form a 16-bit variable. The function Verify_FAST_corner takes this 16-bit variable to check the feature condition. The central pixel is added to features_list if the presence of 9 consecutive darker or brighter pixels is valid.

The number of work-items launched for the kernel is equal to the number of cells in the image. Each work-item works on one image cell where the coordinate is determined by one top-left (tl) pixel and one bottom-right (br) pixel. The boundaries for image cells are fixed. They are pre-computed and are saved to grid_coors array in the initialization step. At the end of kernel, the work-item writes the identification of grid to FAST_ready_channel. From that, the next step knows which cell is ready for processing.

After FAST kernel, FAST features are added to features_list. However, the features_list is a global array used for all image cells and the issue is that number of features in each cell is different from other cells. To avoid a memory conflict, seperate zones are created for each image cell in features_list. Noting that the maximum number of features in one cell is equal to the number of pixels, features_list array is hence created with *N_CELLS* x *RES* elements where *N_CELLS* is the number of image cells and *RES* is the number of pixels (resolution) in the biggest cell. Each element is composed of three factors (x, y, score) corresponding to 2-D coordinates of the feature in the image and its hessian score. The Hessian score will be computed in the next kernel. Each image cell with an identification id will work on the memory zone from the position at id x RES to the position at (id+1)*RES in features_list.

6.5.2 HESSIAN_COMPUTE kernel

As shown in algorithm 6.2, before computing the hessian score, a work-item of HESSIAN_COMPUTE kernel must call read_channel_intel function to get from FAST_ready_channel an identification (ptidx) of an image cell and its number of FAST features (num_ktps). The oldest identification in the channel will be returned since AOCL channel is in type of FIFO array. The implementation of read_channel_intel function is

Algorithm 6.1 FAST kernel

```
declare global arrays: img, features_list, grid_coors;
function KERNEL: FAST
    declare 8-bit private variables : fast8_d1, fast8_d2,fast8_b1, fast8_b2;
    declare 16-bit private variables : fast16;
    ptidx ← get_global_id(0);
    image_cell ← get_Image_Cell(grid_coors, ptidx);
    num_ktps ← 0;
    For each pixel in image_cell do
        /////segmentation test for each symmetric pair of pixels//////////
        p ← Get_intensity(pixel);
        dark ← 1;
        bright ← 1;
        For i from 0 to 7 do // for each pair of 8 symmetric pairs
            p1 ← Get_intensity(bresenham_circle[i]);
            p2 ← Get_intensity(bresenham_circle[i+8]);
            if(dark ==1)
                fast8_d1 ← set_bit(p,p1,i);
                fast8_d2 ← set_bit(p,p2,i+8);
                dark ← Segmentation_test(fast8_d1,fast8_d2);
            end if
            if(bright==1)
                fast8_b1 ← set_bit(p,p1,i);
                fast8_b2 ← set_bit(p,p2,i+8);
                bright ← segmentation_test(fast8_b1,fast8_b2);
            end if
            if ((dark ==0) && (bright==0)) break; end if
        end for
        if((dark ==0) && (bright==0)) go_to_next_pixel; end if
        ////////////////////////////////////////////////////////////////////////////
        ////////////////////verify corner/////////
        if(dark || bright)
            if(dark)
                fast16 ← Concatenation (fast8_d1, fast8_d2);
            else
                fast16 ← Concatenation (fast8_b1, fast8_b2);
            end if
            test_corner ← Verify_FAST_corner(fast16);
            if(test_corner)
                features_list ← Add_to_list (pixel_coordinates);
                num_ktps++;
            end if
        end if
        ////////////////////////////////////////
    end for
    write_channel_intel(FAST_ready_channel,{ ptidx, num_ktps}); end function
```

Algorithm 6.2 HESSIAN_COMPUTE kernel

```
declare global arrays: img, features_list;
function KERNEL: Hessian_Compute
   {ptidx, num_ktps} ← read_channel_intel(FAST_ready_channel);
   For i from 0 to num_ktps do
      feature ← Get_pixel_from_list(features_list, i);
      hessian_score ← Compute_Hessian_score(feature);        features_list ←
Update_features_list(hessian_score);
   end for
   write_channel_intel (HC_ready_channel, {ptidx, num_ktps}); end function
```

blocking so that the processing will wait until an identification is succesfully read. Following HOOFR algorithm, hessian computation is simply applying three 7x7 gaussian square filters on the feature and it is realized for all FAST features in the image cell. The features_list will be updated with the computed Hessian score.

Similarly to FAST kernel, the work-item writes the identification of image cell to HC_ready_channel at the end of function to communicate with FILTERING kernel.

6.5.3 Module duplication

During experiments, we found that FAST kernel and HESSIAN_COMPUTE kernel are bottle-necks of the algorithm flow. These two kernels do not consum much logic resources but take much time to compute. Despite of the advantage of the FAST segmentation test allowing to reject rapidly the non-valid features, the test of the whole image (for example: 453620 pixels with the dimension of 1226x370) makes FAST kernel become costly. HESSIAN_COMPUTE kernel works only on pixels considered as FAST keypoints. However, FAST detection returns many keypoints and Hessian score computation for each keypoint is costly so that HESSIAN_COMPUTE kernel is also time consuming. To accelerate the processing, we duplicate these two blocks.

There are two ways for the duplication: using num_compute_units attribute or physical duplication. For the first method, the value of num_compute_units is set to 2 in the declaration of the kernel function. The work-items are scheduled automatically to execute on 2 compute_units with uncontrolled ordering. However, AOCL tools only support the channel implementation with single compute_unit kernel. Hence, physical method is used in our design.

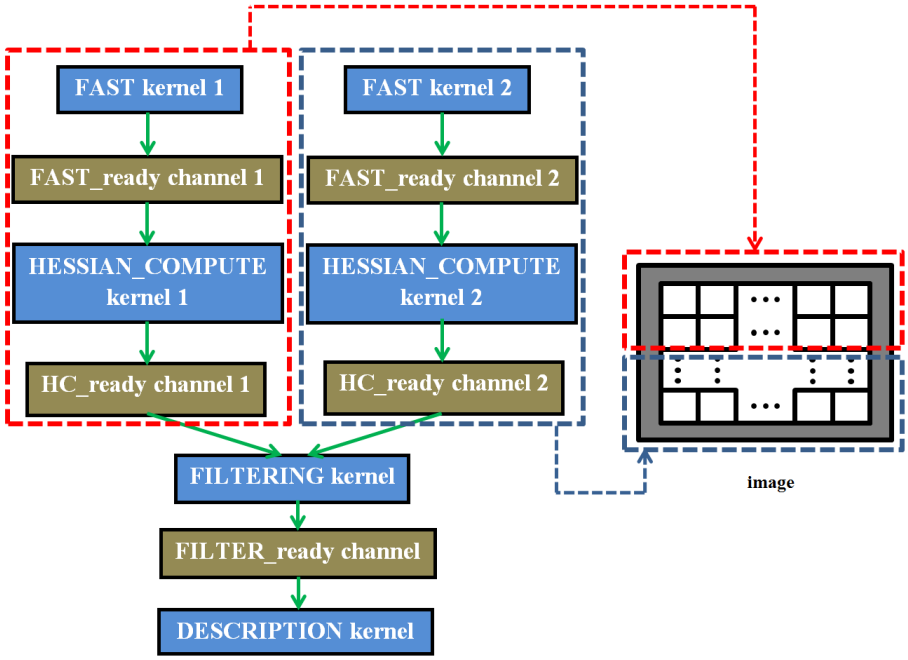


Figure 6.6: Kernels duplication schema

As shown in figure 6.6, two identical kernel functions are created for each duplicated block with exactly the same interface except the function name. To avoid the memory conflict, each function is called from the host to work on seperate image zone: one for the first half and one for second half. Following the instruction of AOCL tool consisting that one kernel can read and write to multiple chanels but one channel can be read and writen from only one kernel, the FAST_ready_channel and HC_ready_channel are also duplicated for kernel communication on each image zone.

6.5.4 FILTERING kernel

This kernel is the last step of detection phase, it uses FILTERING_ready_channel to communicate with DESCRIPTION kernel. To read from HC_ready_channel, due to the fact that this channel is duplicated, we must use nonblocking channel reads as shown in algorithm 6.3 to get one image cell identification from two seperate FIFO chains.

This kernel is aimed to keep a limited relevant features in one image cell. The maximum number of keypoints is defined by POINTS_PER_CELL. With the same objective of avoiding the memory accessing conflict, we declare an array called filtered_features_list.

Algorithm 6.3 FILTERING kernel

```
declare global arrays: features_list, filtered_features_list;
function KERNEL: Filtering
   declare private variables : hessian_min, filtered_num_elements;
   valid ← false;
   while(!valid) do
      {ptidx, num_ktps} ← read_channel_nb_intel(HC_ready_channel, &valid);
       if(!valid) {ptidx, num_ktps} ← read_channel_nb_intel(HC_ready_channel_2,
&valid); end if;
   end while
   filtered_num_elements ← 0;
   For i from 0 to num_ktps do
      hess_score ← get_score(features_list, i);
      if ( (filtered_num_elements < POINTS_PER_CELL) || (hess_score > hessian_min)
)
            {filtered_features_list, hessian_min, filtered_num_elements} ← Up-
date_filtered_list (features_list, i);
      end if
   end for
   write_channel_intel (FILTERING_ready_channel, {ptidx, filtered_num_elements});
end function
```

An image cell occupies POINTS_PER_CELL individual positions in this array. In total, filtered_features_list is the size of POINTS_PER_CELL*N_CELLS. Each element is in the same form with the elements of features_list containing information about the coordinates (x,y) and hessian score of a feature. The number of relevant keypoints (filtered_num_elements) is initialized to zero. For every FAST feature detected in the image cell, the filtering procedure in Update_filtered_list function is described as follows:

- If *filtered_num_elements* is smaller than POINTS_PER_CELL, feature is added to *filtered_features_list* and *filtered_num_elements* increments by one.
- When *filtered_num_elements* attains the value of POINTS_PER_CELL, filtered_features_list is queried to find the position which contains keypoint having the smallest hessian score (*hess_min*).
- Then, for each new feature, its hessian score is first compared to hessian_min. if its score is smaller than hessian_min, it is discarded rapidly without changing the *filtered_features_list*. In contrast, when its score is bigger, it is added to the list and a new *hessian_min* is determined by query *filtered_features_list* once again.

Algorithm 6.4 DESCRIPTION kernel

```
declare global arrays: imgintegral, filtered_features_list, descriptors, num_kpts_list;
function KERNEL: Description
   declare private variables: keypoint, keypoint_angle, pattern_points,
keypoint_descriptor;
   {ptidx, num_ktps} ← read_channel_intel(FILTERING_ready_channel);
   For i from 0 to num_ktps do
      keypoint ← Get_keypoint(filtered_features_list, i)
      /////// compute orientation ////////////////////
      pattern_points ← Gaussian_smooth(imgintegral, keypoint, 0);
      keypoint_angle ← Compute_keypoint_angle(pattern_points);
      ////////////////////////////////////////////////////////
      ///////compute descriptor/////////
      pattern_points ← Update_Gaussian_smooth(imgintegral, keypoint,
keypoint_angle);          keypoint_descriptor ← Make_description(pattern_points);
      descriptors ← Add_to_descriptors_list(keypoint_descriptor);
      ////////////////////////////////////////////
   end for
   num_kpts_list ← Add_to_num_ktps_list(num_ktps);  end function
```

6.5.5 DESCRIPTION kernel

The DESCRIPTION kernel is shown in algorithm 6.4. The processing task of each feature consists of two parts: orientation estimation and binary descriptor construction. The variable pattern_points is an array which contains the intensity of surrounding pixels used to describe the central pixel. In each part, the intensities of surrounding points are firstly smoothed by gaussian. In HOOFR, to have a high efficiency between precision and timing, this smoothing is approximated by mean intensity requiring integral image of the original input image. This integral image is computed by CPU and is loaded to global array *imgintegral* before this kernel is launched.

Features description is saved to a global array denoted as *descriptors* . The structure of *descriptors* is an 2-D array of 32-bit elements where the number of rows is equal to the number of elements in filtered_features_list and the number of columns is 8. In practice, each row is a 256-bit descriptor of one feature. Each image cell will describe its own features and save result to the rows from the position ptidx*POINTS_PER_CELL to the position (ptidx+1)*POINTS_PER_CELL. Due to the fact that the number of features in each image cell could be varied (from 0 to POINTS_PER_CELL), some unused

Kernel Name	ALUTs	FFs	RAMs	DSPs
FAST	21021	29762	242	4
Hessian_Compute	11736	18193	122	9
Filtering	11485	24003	180	1
Description	59820	73948	376	40
Channel resources	230	1094	5	0
Total (no duplication)	**104292**	**147000**	**925**	**54**
	(23%)	**(17%)**	**(51%)**	**(3%)**
FAST_2	21021	29762	242	4
Hessian_Compute_2	11736	18193	122	9
Total (duplication)	**137049**	**194955**	**1289**	**67**
	(31%)	**(22%)**	**(71%)**	**(4%)**
Available	**448160**	**896320**	**1805**	**1633**

Table 6.1: FPGA resource usage

rows could exist. Hence, the quantity of features must be saved to a global array called num_kpts_list to determine the useful rows in each memory zone.

After DESCRIPTION kernel, three global arrays (*filtered_features_list*, *descriptors* and *num_kpts_list*) are uploaded back to CPU to regroup the information.

6.6 Implementation and Evaluation

6.6.1 Resource Usage

Our design was synthesized for an Arria 10 SoC SX660 architecture including a dual-core ARM Cortex-A9 processor (1.5 GHz) and a FPGA with 660K LEs. The version of AOCL tool is 17.0. As shown in table 6.1, availability of the resources in Arria 10 SX660 does not constrain any design model (with duplication or without duplication). Our description kernel consumes the most resources and it is much more costly comparing to the description module in [126] or [122]. The reason is that processing complexity of the HOOFR algorithm was respected in our design where the keypoint orientation and keypoint descriptor are generated in description module. Moreover, noting that instead of using raw value as in BRIEF, pixel intensity in HOOFR flow is filtered to be robust to image noise. As a result, description kernel takes more resources to handle its operation.

6.6.2 Timings

We tested our design on images of the 16th sequence of KITTI dataset. The image size is of 1226 x 370 pixels. However, to evaluate the effect of image resolution to processing time, we rescale the original images to the different resolution. The scaling was done using resize function in OpenCV. Then, for each resolution, measuring time was achieved by a mean value after 100 launches. As shown in table 6.2, our design reaches a frequency of 54 frame per second (fps) at the original scale (1226 x 370), generating on average of 1750 keypoints per image. At full-HD scale (1920 x 1080), we obtain a frequency of 14 fps with 6929 keypoints per frame.

In the reference [123], the author demonstrated that his design achieves 42 fps with ORB extractor. However, he deal with only 1000 keypoints and the algorithm was extremely simplified by changing Harris score to SAD score or changing Gaussian smooth to Binominal smooth. Besides, he used score only for 3x3 Non-maximal suppression, which is not the original idea of ORB developers. Indeed, in the original version, Harris score is aimed to filter keypoints in an image zone. If the number of features returned after FAST detection is more than a value K in a zone, only K relevant ones having the highest Harris score are maintained as done in our design.

Another advantage of our system is that the performance is stable across the frames when the maximum number of keypoints in each image zone is limited. In contrast, for other systems in the state of the art, the only way to manage the number of keypoints is changing FAST threshold. Given an random image, if the FAST threshold is not determined so the number of keypoints is unbounded. Otherwise, if FAST processing is stopped when N keypoints are found, we could not warrant the homogeneous keypoint distribution which is very important in SLAM application. Figure 6.7 demonstrates the acceleration on Arria 10 board of our design in comparison to the C++ version running completely on the embedded ARM CPU. It is obviously that the higher image resolution is,the higher computation cost is. By offloading the processing to FPGA, we could obtain a speed from 7x to 9x times faster.

Resolution	N keypoints	NX x NY	Time (ms)	fps
352x240	360	6x4	4.031	248
350x480	682	6x8	8.210	121
580x480	1074	10x8	11.993	83
720x480	1462	14x8	15.362	65
720x576	1780	14x10	18.590	53
1226x370	1750	24x6	18.247	54
1280x720	3661	24x14	38.262	26
1920x1080	6929	38x20	68.684	14

Table 6.2: Timimg performance (FAST_threshold = 12, POINTS_PER_CELL = 15)

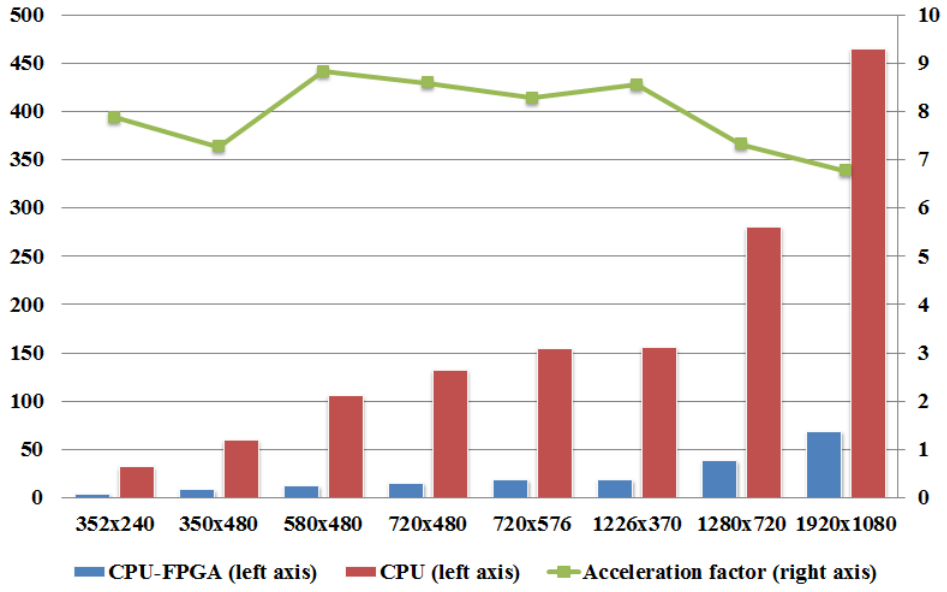


Figure 6.7: Acceleration factor evaluated for an Arria 10 SoC (Right axis: execution time in ms, Left axis: acceleration factor)

Resolution	GPU time (ms)	GPU power	FPGA time (ms)	FPGA power	FPGA Power efficiency
352x240	15.362	64 (W)	4.031	21 (W)	11.612
350x480	19.779		8.210		7.341
580x480	20.075		11.993		5.608
720x480	20.191		15.362		4.005
720x576	21.195		18.590		3.474
1226x370	23.441		18.247		3.914
1280x720	27.127		38.262		2.160
1920x1080	49.023		68.684		2.175

Table 6.3: FPGA - GPU comparison

6.6.3 Perforances comparison: FPGA vs GPU implementations

Our design is realized using OpenCL which gives us a capability of implementing not only on FPGAs but also on various alternative hardware such as GPUs. Here, to compare benchmarks, we used a powerful GPU Nvidia Geforce GT 740 containing 384 CUDA cores clocked at 1.0 GHz. The essential difference is that GPUs do not support channel communication so that kernel blocks must be launched sequentially. Table 6.7 shows timing and power efficiency comparison between the FPGA and the GPU. As can be seen, FPGA is faster than GPU at low resolution but at higher resolution, GPU becomes faster. The reason is that a GT 740 GPU includes a huge number of CUDA cores. At the low resolutions, the number of thread is small so that it did not make use of all computation resources. Otherwise, when the resolution increase, the number of thread increase. All GPU resources are hence employed in processing and the GPU becomes faster in our benchmark.

The Power efficiency factor is defined as the processing speed given a power energy supply. As we can see, up to full-HD resolution (1920 x 1080), Arria 10 FPGA still overcomes Nvidia GT 740 GPU in terms of power efficiency.

6.7 Conclusion

To reach a low energy consumption, an OpenCL-based FPGA SoC architecture for HOOFR feature extraction has been designed. The complexity of HOOFR algorithm was respected to ensure the robustness. Each block was designed so that the detection result on hardware is similar to that proceeded in software. This feature extraction system on FPGA respects bucketing method to warrant the homogeneous distribution of keypoints because it is aimed to use in SLAM applications. The design was evaluated for on Arria 10 SoC FPGA where the OpenCL design is 7x to 9x faster than the C++ implementation running completely on the on-chip ARM CPU. The throughput was 54 fps at 1226x370 or 14 fps at 1920x1080. Moreover, through the experiments, FPGA offers a better power efficiency comparing to a GPU implementation.

In our SLAM system, HOOFR feature extraction is the front-end part. Noting that the back-end part (SLAM kernel) has a high processing complexity and it is not suitable to be implemented on current FPGA due to the lack of logic elements. Therefore, we intend to propose a heterogeneous architecture based FPGA for SLAM applications where the font-end part runs on FPGA and the SLAM kernel runs on multi-core CPU. This kind of architecture will be dedicated to embed SLAM algorithm on mobile devices such as autonomous robots or intelligent vehicles.

Conclusion and Future Works

Conclusion

In this thesis, we have studied a visual SLAM system for large-scale autonomous vehicle applications. The visual SLAM system were considered as the combination of 2 principal parts: the image-processing (front-end task) and the SLAM-core (back-end task). With in-depth investigation and comparative analysis, corresponding proposals were presented for the two tasks to meet the requirement in this field:

For the front-end task, we have presented a method named HOOFR detector, which aims to address the problem of detecting, describing and matching image keypoints. Our detector is the combination of a modified ORB detector with a Hessian score, while our descriptor employs a human retina based descriptor consisting of a FREAK version with enhanced overlapping. Our proposal offers a better compromise between processing times and matching quality compared to others algorithms in the state-of-the-art such as SIFT, SURF and ORB. The experimental test shows that HOOFR is much faster than SURF, SIFT with competitive matching results. Besides, HOOFR exhibits comparably low computation cost as ORB and outperforms ORB matching performance in most real scenes. HOOFR extractor were also proved to be implemented efficiently on embedded platform such as ODROID-XU4 for computer vision applications.

For the back-end task, a novel estimation algorithm for feature-based stereo VSLAM has been presented. It integrates HOOFR features so that it is referred as HOOFR SLAM. The binary descriptor is employed for motion estimation and loop closure detection. Motion estimates are integrated over time following a hybrid filtering/key frame strategy. Position is estimated using a widowed weighted mean using previous neighbor frames.

Weights are computed from inter-frame robust feature matching. The localization accuracy was validated on six well-known datasets (KITTI, Oxford, Malaga, MRT, St-Lucia and New College).

Afterwards, the parallelized lightweight VSLAM framework on CPU-GPU architecture was then obtained as a result of a hardware-software mapping study addressing feature extraction, data processing, hardware implementation and benchmarking. The real-time algorithm implementation on high performance Intel-based PC architecture processes real-time frame rate at more than 20 Hz using sequences of KITTI dataset. On the Tegra TX1 embedded system, the processing time is 6 fps running rate which can be potentially improved with the emergence of embedded architectures with high performances.

Finally, to take advantage of FPGA architectures, especially in terms of energy consumption, an OpenCL-based FPGA SoC architecture for HOOFR feature extraction has been designed. The complexity of HOOFR algorithm was respected to ensure the robustness. This feature extraction system on FPGA respects bucketing method to warrant the homogeneous distribution of keypoints because it is aimed to use in SLAM applications. The FPGA implementation shows that the OpenCL design is 7x to 9x faster than the C++ implementation running on the on-chip ARM CPU. The obtained throughput is 54 fps at 1226x370 pixels or 14 fps at 1920x1080 pixels. Moreover, through the experiments, the FPGA offers a better power efficiency compared to a GPU implementation. This makes FPGAs potential candidates for designing a dedicated system based SLAM applications.

Future works

Despite of a fast running on a powerful Intel-based PC, real-time performance of the whole HOOFR SLAM algorithm on an embedded system that consumes few watts remains a perspective of this work. Moreover, the improvement of HOOFR localization in high-way environments will also be taken into account. In the near future, we would like to further improve the performance of HOOFR SLAM on both localization and speed and evaluate it using datasets of an instrumented vehicle (figure 6.8) of our SATIE laboratory. The emergence of new heterogeneous CPU-GPU architectures such as Xavier Nvidia (8

Figure 6.8: Instrumented vehicle of SATIE laboratory

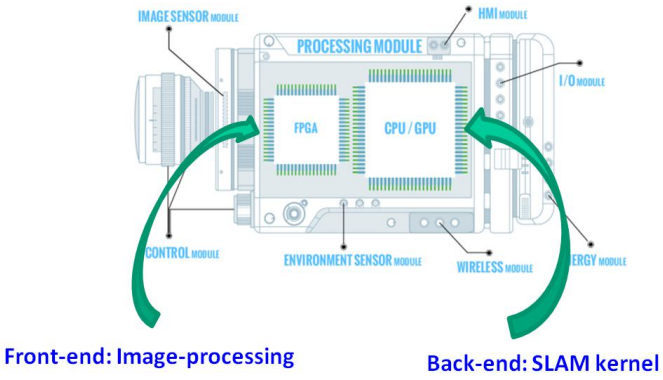


Figure 6.9: FPGA/GPU-CPU architecture

Core ARM64 CPU, 512 Core Volta GPU) should help to embed the HOOFR SLAM algorithm with real-time constraints. Furthermore, HOOFR SLAM is evaluated in this thesis using the HIL (Hardware In the Loop) approach and datasets. In the future, we also intend to test HOOFR SLAM with an online dataflow provided by a stereo camera.

For HOOFR extractor FPGA implementation, we would like to continue optimizing our design to reduce resource usage and execution times. Then, we intend to integrate in our system an interface to camera to complete the SLAM processing from raw sensor data. Furthermore, we would like to investigate the embeddability of the whole SLAM application on a combination of FPGA/GPU-CPU system as shown in figure 6.9 for autonomous vehicles, in particular, on the instrumented vehicle of the SATIE laboratory. The FPGA is aimed to be interfaced with sensor to process image from sensor data to HOOFR extraction. Otherwise, the SLAM kernel will be implemented on CPU-GPU heterogeneous architecture.

Appendix

Root Mean Square Error (RMSE)

The Root Mean Square Error (also called the root mean square deviation, RMSD) is a frequently used measure of the difference between values predicted by a model and the values actually observed from the environment that is being modeled. These individual differences are also called residuals, and the RMSE serves to aggregate them into a single measure of predictive power. The RMSE of a model prediction with respect to the estimated variable X_{model} is defined as the square root of the mean squared error:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(X_{obs,i} - X_{model,i})^2}{n}}$$

where X_{obs} is observed values andX_{model} is modeled values at time/place i. The calculated RMSE values is measured on the same scale, with the same units as X_{obs} and X_{model}. It expresses average model prediction error, can range from 0 to ∞ and are indifferent to the direction of errors. It is negatively-oriented scores, which means lower values are better.

We use RMSE to evaluate the performance of SLAM system, X_{model} is assigned to ground-truth provided by GPS-RTK while X_{obs} is camera position estimated from SLAM algorithms.

Essential matrix

Image points are represented by homogeneous 3-vectors q and $q^{'}$ in the first and second view, respectively. World points are represented by homogeneous 4-vectors Q .A perspective view is represented by a 3x4 camera matrix P indicating the image projection q ~PQ , where ~ denotes equality up to scale. A view with a finite projection center can be factored into $P = K[R|t]$, where K is a 3x3 upper triangular calibration matrix holding the intrinsic parameters and R is a rotation matrix. Let the camera matrices for the two views be $K_1[I|0]$ and $P = K_2[R|t]$. Let $[t]_x$ denotes the skew symmetric matrix.

$$[t]_x = \begin{bmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{bmatrix} \tag{6.1}$$

Then, the fundamental matrix is

$$F = K_2^T [t]_x R K_1^{-1} \tag{6.2}$$

The fundamental matrix encodes the well-known coplanarity or epipolar constraint $q^{'T} F q = 0$. The fundamental matrix can be considered without knowledge of the calibration matrices. Moreover, it continues to exist when the projection centers are not finite. If $K_1$and $K_2$are known, the cameras are said to be calibrated. In this case, we can always assume that the image points $q$and $q^{'}$ have been pre-multiplied by K_1^{-1} and K_2^{-1} , respectively, so that the epipolar constraint simplifies to

$$q^{'T} E q = 0 \tag{6.3}$$

where the matrix $E = [t]_x R$ is called the essential matrix.

Theorem 1. *A real nonzero 3x3 matrix, F , is a fundamental matrix if and only if it satisfies the equation:*

$$det(F) = 0 \tag{6.4}$$

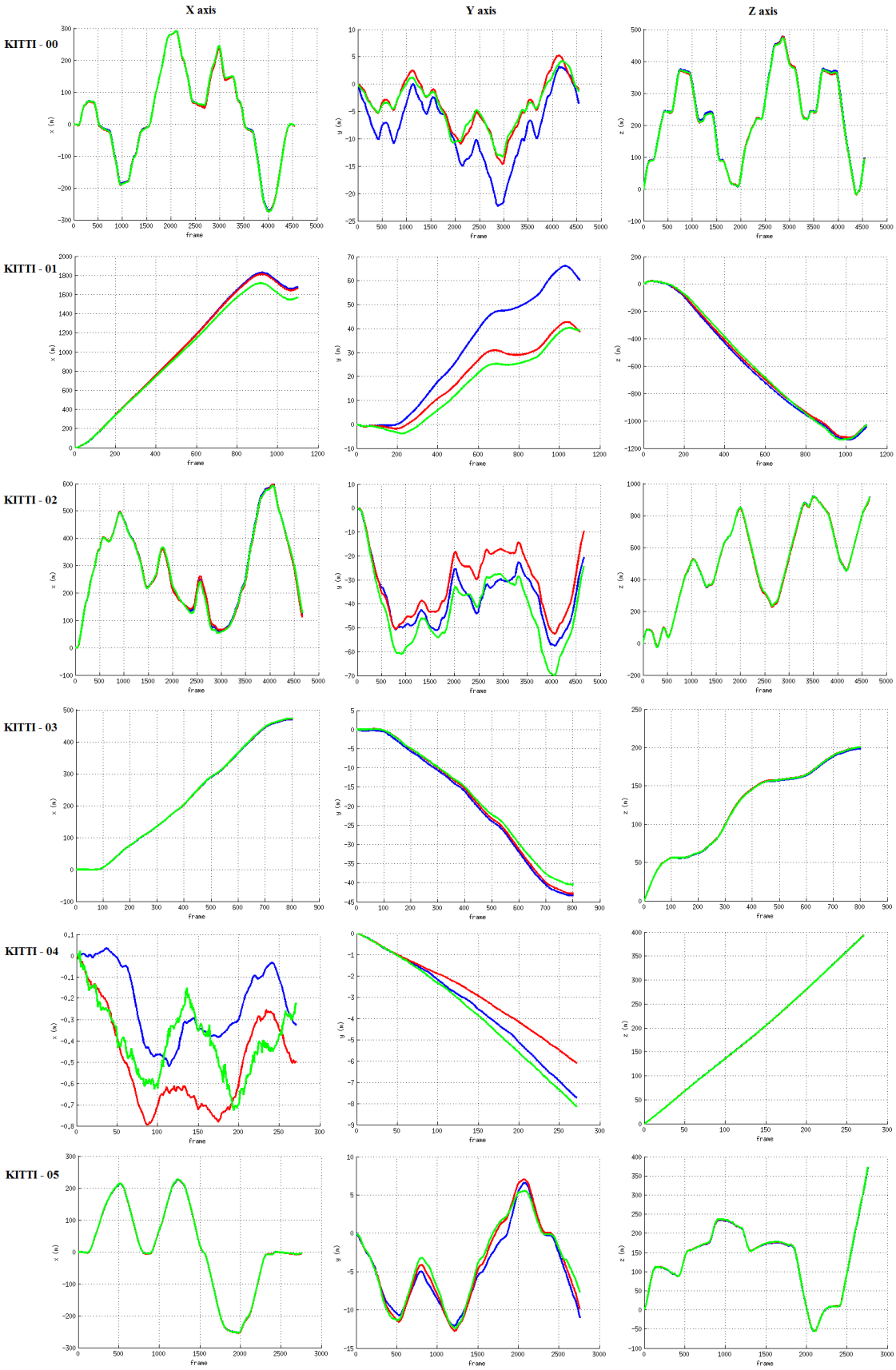
An essential matrix has the additional property that the two nonzero singular values are equal. This leads to the following cubic constraints on the essential matrix:

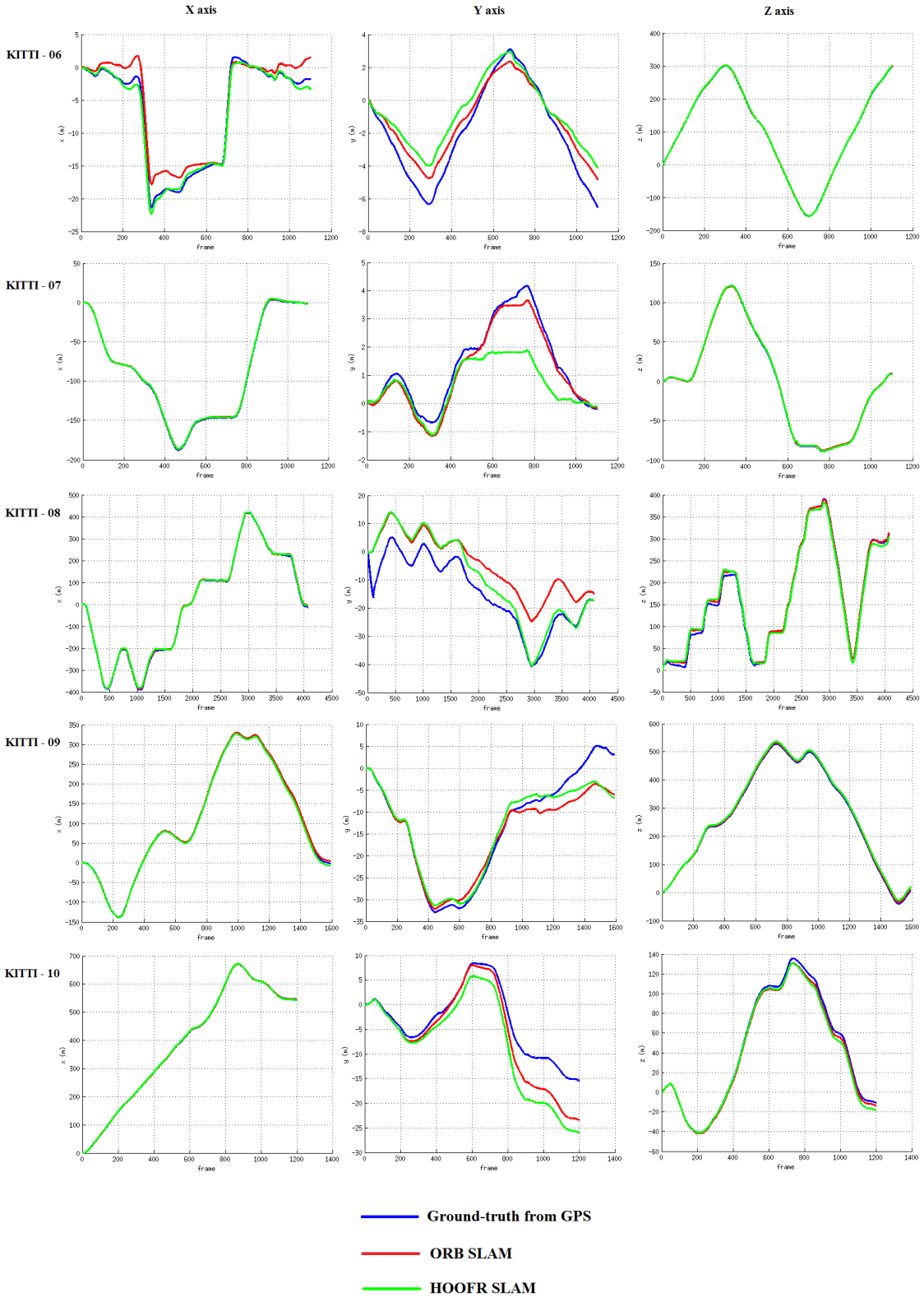
Theorem 2. *A real nonzero 3x3 matrix, E , is an essential matrix if and only if it satisfies the equation:*

$$EE^T E - \frac{1}{2} trace(EE^T)E = 0 \tag{6.5}$$

SLAM Error in KITTI dataset

This section shows the performance comparison of HOOFR SLAM and ORB SLAM with respect to ground-truth provided by GPS-RTK for the 11 first sequences of KITTI dataset. The camera position is presented on 3 axis separately, for all frames in each sequence.


X axis
Y axis
Z axis
KITTI - 00
KITTI - 01
KITTI - 02
KITTI - 03
KITTI - 04
KITTI - 05
frame
x (m)
y (m)
z (m)


X axis
Y axis
Z axis
KITTI - 06
KITTI - 07
KITTI - 08
KITTI - 09
KITTI - 10
x (m)
y (m)
z (m)
frame
Ground-truth from GPS
ORB SLAM
HOOFR SLAM

Bibliography

[1] Cyrill Stachniss. *Robotic mapping and exploration*, volume 55. Springer, 2009.

[2] Andreas Nüchter, Kai Lingemann, Joachim Hertzberg, and Hartmut Surmann. 6d slamâ3d mapping outdoor environments. *Journal of Field Robotics*, 24(8-9):699–722, 2007.

[3] Sebastian Thrun, Michael Montemerlo, and Andrei Aron. Probabilistic terrain analysis for high-speed desert driving. In *Robotics: Science and Systems*, pages 16–19, 2006.

[4] Paul Newman, John Leonard, Juan D Tardós, and José Neira. Explore and return: Experimental validation of real-time concurrent mapping and localization. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 2, pages 1802–1809. IEEE, 2002.

[5] Jan Steckel and Herbert Peremans. Batslam: Simultaneous localization and mapping using biomimetic sonar. *PloS one*, 8(1):e54076, 2013.

[6] Stephen Se, David G Lowe, and James J Little. Vision-based global localization and mapping for mobile robots. *IEEE Transactions on robotics*, 21(3):364–375, 2005.

[7] Thomas Lemaire, Cyrille Berger, Il-Kyun Jung, and Simon Lacroix. Vision-based slam: Stereo and monocular approaches. *International Journal of Computer Vision*, 74(3):343–364, 2007.

[8] Andrew J Davison. Real-time simultaneous localisation and mapping with a single camera. In *null*, page 1403. IEEE, 2003.

[9] Radu Bogdan Rusu, Aravind Sundaresan, Benoit Morisset, Kris Hauser, Motilal Agrawal, Jean-Claude Latombe, and Michael Beetz. Leaving flatland: Efficient real-time three-dimensional perception and motion planning. *Journal of Field Robotics*, 26(10):841–862, 2009.

[10] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, et al. Stanley: The robot that won the darpa grand challenge. *Journal of field Robotics*, 23(9):661–692, 2006.

[11] Matthew Johnson-Roberson, Oscar Pizarro, Stefan B Williams, and Ian Mahon. Generation and visualization of large-scale three-dimensional reconstructions from underwater robotic surveys. *Journal of Field Robotics*, 27(1):21–51, 2010.

[12] David Ribas, Pere Ridao, Juan Domingo Tardós, and José Neira. Underwater slam in man-made structured environments. *Journal of Field Robotics*, 25(11-12):898–921, 2008.

[13] Pedro Piniés, Juan D Tardós, and José Neira. Localization of avalanche victims using robocentric slam. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 3074–3079. IEEE, 2006.

[14] Christopher Mei, Eric Sommerlade, Gabe Sibley, Paul Newman, and Ian Reid. Hidden view synthesis using real-time visual slam for simplifying video surveillance analysis. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4240–4245. IEEE, 2011.

[15] Jorge Artieda, José M Sebastian, Pascual Campoy, Juan F Correa, Iván F Mondragón, Carol Martínez, and Miguel Olivares. Visual 3-d slam from uavs. *Journal of Intelligent and Robotic Systems*, 55(4-5):299, 2009.

[16] Clark F Olson, Larry H Matthies, John R Wright, Rongxing Li, and Kaichang Di. Visual terrain mapping for mars exploration. *Computer Vision and Image Understanding*, 105(1):73–85, 2007.

[17] Bastian Steder, Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Visual slam for flying vehicles. *IEEE Transactions on Robotics*, 24(5):1088–1093, 2008.

[18] 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. In *Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 1–4. IEEE Computer Society, 2007.

[19] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234. IEEE, 2007.

[20] Oscar G Grasa, Javier Civera, and JMM Montiel. Ekf monocular slam with relocalization for laparoscopic sequences. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4816–4821. IEEE, 2011.

[21] Lina M Paz, Pedro Piniés, Juan D Tardós, and José Neira. Large-scale 6-dof slam with stereo-in-hand. *IEEE transactions on robotics*, 24(5):946–957, 2008.

[22] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (6):1052–1067, 2007.

[23] Juan Manuel Sáez and Francisco Escolano. 6dof entropy minimization slam. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1548–1555. IEEE, 2006.

[24] Pedro Piniés and Juan D Tardós. Large-scale slam building conditionally independent local maps: Application to monocular vision. *IEEE Transactions on Robotics*, 24(5):1094–1106, 2008.

[25] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[26] Raúl Mur-Artal and Juan D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *arXiv preprint arXiv:1610.06475*, 2016.

[27] Hyon Lim, Jongwoo Lim, and H Jin Kim. Real-time 6-dof monocular visual slam in a large-scale environment. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 1532–1539. IEEE, 2014.

[28] Joan Sola. Multi-camera vslam: from former information losses to self-calibration. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), San Diego, CA, USA*, 2007.

[29] Eagle S Jones and Stefano Soatto. Visual-inertial navigation, mapping and localization: A scalable real-time causal approach. *The International Journal of Robotics Research*, 30(4):407–430, 2011.

[30] Michael Montemerlo and Sebastian Thrun. Fastslam 2.0. *FastSLAM: A scalable method for the simultaneous localization and mapping problem in robotics*, pages 63–90, 2007.

[31] Michael J Milford and Gordon F Wyeth. Mapping a suburb with a single camera using a biologically inspired slam system. *IEEE Transactions on Robotics*, 24(5):1038–1053, 2008.

[32] Richard I Hartley and Peter Sturm. Triangulation. *Computer vision and image understanding*, 68(2):146–157, 1997.

[33] Stephen Se, David Lowe, and Jim Little. Mobile robot localization and mapping with uncertainty using scale-invariant visual landmarks. *The international Journal of robotics Research*, 21(8):735–758, 2002.

[34] Clark F Olson, Larry H Matthies, Marcel Schoppers, and Mark W Maimone. Rover navigation using stereo ego-motion. *Robotics and Autonomous Systems*, 43(4):215–229, 2003.

[35] Kurt Konolige and Motilal Agrawal. Frameslam: From bundle adjustment to real-time visual mapping. *IEEE Transactions on Robotics*, 24(5):1066–1077, 2008.

[36] Kurt Konolige, James Bowman, JD Chen, Patrick Mihelich, Michael Calonder, Vincent Lepetit, and Pascal Fua. View-based maps. *The International Journal of Robotics Research*, 29(8):941–957, 2010.

[37] Christopher Mei, Gabe Sibley, Mark Cummins, Paul M Newman, and Ian D Reid. A constant-time efficient stereo slam system. In *BMVC*, pages 1–11, 2009.

[38] Michael Kaess and Frank Dellaert. Probabilistic structure matching for visual slam with a multi-camera rig. *Computer Vision and Image Understanding*, 114(2):286–296, 2010.

[39] Gerardo Carrera, Adrien Angeli, and Andrew J Davison. Slam-based automatic extrinsic calibration of a multi-camera rig. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2652–2659. IEEE, 2011.

[40] Andrew J Davison, Yolanda Gonzalez Cid, and Nobuyuki Kita. Real-time 3d slam with wide-angle vision. *IFAC Proceedings Volumes*, 37(8):868–873, 2004.

[41] Davide Scaramuzza and Roland Siegwart. Appearance-guided monocular omnidirectional visual odometry for outdoor ground vehicles. *IEEE transactions on robotics*, 24(5):1015–1026, 2008.

[42] Albert S Huang, Abraham Bachrach, Peter Henry, Michael Krainin, Daniel Maturana, Dieter Fox, and Nicholas Roy. Visual odometry and mapping for autonomous flight using an rgb-d camera. In *Robotics Research*, pages 235–252. Springer, 2017.

[43] David Nistér. An efficient solution to the five-point relative pose problem. *IEEE transactions on pattern analysis and machine intelligence*, 26(6):756–770, 2004.

[44] Hauke Strasdat, J Montiel, and Andrew J Davison. Scale drift-aware large scale monocular slam. *Robotics: Science and Systems VI*, 2, 2010.

[45] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[46] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22, 2000.

[47] Azra Fetić, Davor Jurić, and Dinko Osmanković. The procedure of a camera calibration using camera calibration toolbox for matlab. In *MIPRO, 2012 Proceedings of the 35th International Convention*, pages 1752–1757. IEEE, 2012.

[48] Reg Willson. Tsai camera calibration software. *C code for Tsai calibration available online at< URL: http://www-2. cs. cmu. edu/afs/cs. cmu. edu/user/rgw/www/TsaiCode. html*, 1995.

[49] D Scaramuzza and R Siegwart. Ocamcalib toolbox: Omnidirectional camera calibration toolbox for matlab. *Google for âocamcalibâ.*, 2006.

[50] Tomas Svoboda, Daniel Martinec, Tomas Pajdla, Jean-Yves Bouguet, Tomas Werner, and Ondrej Chum. Multi-camera self-calibration. *Czech Technical University, Prague, Czech Republic.\tt http://cmp. felk. cvut. cz/˜ svoboda/SelfCal*, 2003.

[51] Olivier Koch, Matthew R Walter, Albert S Huang, and Seth Teller. Ground robot navigation using uncalibrated cameras. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2423–2430. IEEE, 2010.

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

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

[54] Sparsh Mittal. A survey of techniques for managing and leveraging caches in gpus. *Journal of Circuits, Systems, and Computers*, 23(08):1430002, 2014.

[55] Philipp Michel, Joel Chestnutt, Satoshi Kagami, Koichi Nishiwaki, James Kuffner, and Takeo Kanade. Gpu-accelerated real-time 3d tracking for humanoid locomotion and stair climbing. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 463–469. IEEE, 2007.

[56] Haiyang Zhang and Fred Martin. Cuda accelerated robot localization and mapping. In *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, pages 1–6. IEEE, 2013.

[57] Lu Ma, Juan M Falquez, Steve McGuire, and Gabe Sibley. Large scale dense visual inertial slam. In *Field and Service Robotics*, pages 141–155. Springer, 2016.

[58] Mikael Persson, Tommaso Piccini, Michael Felsberg, and Rudolf Mester. Robust stereo visual odometry from monocular techniques. In *Intelligent Vehicles Symposium (IV), 2015 IEEE*, pages 686–691. IEEE, 2015.

[59] Siddharth Choudhary, Shubham Gupta, and PJ Narayanan. Practical time bundle adjustment for 3d reconstruction on the gpu. In *European Conference on Computer Vision*, pages 423–435. Springer, 2010.

[60] Changchang Wu, Sameer Agarwal, Brian Curless, and Steven M Seitz. Multicore bundle adjustment. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3057–3064. IEEE, 2011.

[61] Diego Rodriguez-Losada, Pablo San Segundo, Miguel Hernando, Paloma de la Puente, and Alberto Valero-Gomez. Gpu-mapping: Robotic map building with graphical multiprocessors. *IEEE Robotics & Automation Magazine*, 20(2):40–51, 2013.

[62] Adrian Ratter, Claude Sammut, and Matthew McGill. Gpu accelerated graph slam and occupancy voxel based icp for encoder-free mobile robots. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 540–547. IEEE, 2013.

[63] Luigi Nardi, Bruno Bodin, M Zeeshan Zia, John Mawer, Andy Nisbet, Paul HJ Kelly, Andrew J Davison, Mikel Luján, Michael FP O'Boyle, Graham Riley, et al. Introducing slambench, a performance and accuracy benchmarking methodology for slam. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 5783–5790. IEEE, 2015.

[64] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011.

[65] M Zeeshan Zia, Luigi Nardi, Andrew Jack, Emanuele Vespa, Bruno Bodin, Paul HJ Kelly, and Andrew J Davison. Comparative design space exploration of dense and semi-dense slam. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 1292–1299. IEEE, 2016.

[66] Luna Backes, Alejandro Rico, and Björn Franke. Experiences in speeding up computer vision applications on mobile computing platforms. In *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2015 International Conference on*, pages 1–8. IEEE, 2015.

[67] Vanderlei Bonato, José A de Holanda, and Eduardo Marques. An embedded multi-camera system for simultaneous localization and mapping. In *International Workshop on Applied Reconfigurable Computing*, pages 109–114. Springer, 2006.

[68] Grigorios Mingas, Emmanouil Tsardoulias, and Loukas Petrou. An fpga implementation of the smg-slam algorithm. *Microprocessors and Microsystems*, 36(3):190–204, 2012.

[69] Sérgio Cruz, Daniel M Muñoz, Milton Conde, Carlos H Llanos, and Geovany A Borges. Fpga implementation of a sequential extended kalman filter algorithm applied to mobile robotics localization problem. In *Circuits and Systems (LASCAS), 2013 IEEE Fourth Latin American Symposium on*, pages 1–4. IEEE, 2013.

[70] Daniel Törtei Tertei, Jonathan Piat, and Michel Devy. Fpga design of ekf block accelerator for 3d visual slam. *Computers & Electrical Engineering*, 55:123–137, 2016.

[71] Mengyuan Gu, Kaiyuan Guo, Wenqiang Wang, Yu Wang, and Huazhong Yang. An fpga-based real-time simultaneous localization and mapping system. In *Field*

Programmable Technology (FPT), 2015 International Conference on, pages 200–203. IEEE, 2015.

[72] Janosch Nikolic, Joern Rehder, Michael Burri, Pascal Gohl, Stefan Leutenegger, Paul T Furgale, and Roland Siegwart. A synchronized visual-inertial sensor system with fpga pre-processing for accurate real-time slam. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 431–437. IEEE, 2014.

[73] Biruk G Sileshi, Juan Oliver, Ricardo Toledo, Jose Gonçalves, and Pedro Costa. Particle filter slam on fpga: A case study on robot@ factory competition. In *Robot 2015: Second Iberian Robotics Conference*, pages 411–423. Springer, 2016.

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

[75] Will Maddern, Geoff Pascoe, Chris Linegar, and Paul Newman. 1 Year, 1000km: The Oxford RobotCar Dataset. *The International Journal of Robotics Research (IJRR)*, 36(1):3–15, 2017.

[76] Jose-Luis Blanco, Francisco-Angel Moreno, and Javier Gonzalez-Jimenez. The malaga urban dataset: High-rate stereo and lidars in a realistic urban scenario. *International Journal of Robotics Research*, 33(2):207–214, 2014.

[77] Frank Moosmann and Christoph Stiller. Velodyne SLAM. In *Proceedings of the IEEE Intelligent Vehicles Symposium*, pages 393–398, Baden-Baden, Germany, June 2011.

[78] Michael Warren, D. McKinnon, H. He, and Ben Upcroft. Unaided stereo vision based pose estimation. In Gordon Wyeth and Ben Upcroft, editors, *Australasian Conference on Robotics and Automation*, Brisbane, 2010. Australian Robotics and Automation Association.

[79] Mike Smith, Ian Baldwin, Winston Churchill, Rohan Paul, and Paul Newman. The new college vision and laser data set. *The International Journal of Robotics Research*, 28(5):595–599, 2009.

[80] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[81] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Computer vision–ECCV 2006*, pages 404–417. Springer, 2006.

[82] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: an efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571. IEEE, 2011.

[83] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *Computer Vision–ECCV 2006*, pages 430–443. Springer, 2006.

[84] Michael Calonder, Vincent Lepetit, Mustafa Ozuysal, Tomasz Trzcinski, Christoph Strecha, and Pascal Fua. Brief: Computing a local binary descriptor very fast. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(7):1281–1298, 2012.

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

[86] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. Brisk: Binary robust invariant scalable keypoints. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2548–2555. IEEE, 2011.

[87] Alexandre Alahi, Raphael Ortiz, and Pierre Vandergheynst. Freak: Fast retina keypoint. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 510–517. Ieee, 2012.

[88] Taihú Pire, Thomas Fischer, Javier Civera, Pablo De Cristóforis, and Julio Jacobo Berlles. Stereo parallel tracking and mapping for robot localization. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 1373–1378. IEEE, 2015.

[89] Krystian Mikolajczyk, Tinne Tuytelaars, Cordelia Schmid, Andrew Zisserman, Jiri Matas, Frederik Schaffalitzky, Timor Kadir, and Luc Van Gool. A comparison of affine region detectors. *International journal of computer vision*, 65(1-2):43–72, 2005.

[90] Engin Tola, Vincent Lepetit, and Pascal Fua. Daisy: An efficient dense descriptor applied to wide-baseline stereo. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(5):815–830, 2010.

[91] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[92] Francesco Bianconi and Antonio Fernández. Evaluation of the effects of gabor filter parameters on texture classification. *Pattern Recognition*, 40(12):3325–3335, 2007.

[93] Krystian Mikolajczyk and Cordelia Schmid. Scale & affine invariant interest point detectors. *International journal of computer vision*, 60(1):63–86, 2004.

[94] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[95] Dai-Duong Nguyen, Abdelhafid El Ouardi, and Samir Bouaziz. Enhanced bio-inspired feature extraction for embedded application. In *Control, Automation, Robotics and Vision (ICARCV), 2016 14th International Conference on*, pages 1–6. IEEE, 2016.

[96] Raul Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[97] Luis Riazuelo, Javier Civera, and JMM Montiel. C 2 tam: A cloud framework for cooperative tracking and mapping. *Robotics and Autonomous Systems*, 62(4):401–413, 2014.

[98] Guillaume Bresson, Thomas Féraud, Romuald Aufrère, Paul Checchin, and Roland Chapuis. Real-time monocular slam with low memory requirements. *IEEE Transactions on Intelligent Transportation Systems*, 16(4):1827–1839, 2015.

[99] Geoffrey Pascoe, Will Maddern, Michael Tanner, Pedro Piniés, and Paul Newman. Nid-slam: Robust monocular slam using normalised information distance. In *Conference on Computer Vision and Pattern Recognition*, 2017.

[100] Andrew J Davison and David W Murray. Mobile robot localisation using active vision. In *European Conference on Computer Vision*, pages 809–825. Springer, 1998.

[101] Andrew J Davison and David W Murray. Simultaneous localization and map-building using active vision. *IEEE transactions on pattern analysis and machine intelligence*, 24(7):865–880, 2002.

[102] Andrew J Davison and Nobuyuki Kita. 3d simultaneous localisation and map-building using active vision for a robot moving on undulating terrain. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE, 2001.

[103] Lina María Paz, Patric Jensfelt, Juan D Tardos, and José Neira. Ekf slam updates in o (n) with divide and conquer slam. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 1657–1663. IEEE, 2007.

[104] Sebastian Thrun, Michael Montemerlo, Daphne Koller, Ben Wegbreit, Juan Nieto, and Eduardo Nebot. Fastslam: An efficient solution to the simultaneous localization and mapping problem with unknown data association. *Journal of Machine Learning Research*, 4(3):380–407, 2004.

[105] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Nonlinear constraint network optimization for efficient map learning. *IEEE Transactions on Intelligent Transportation Systems*, 10(3):428–439, 2009.

[106] Giorgio Grisetti, Rainer Kummerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010.

[107] Hauke Strasdat, José MM Montiel, and Andrew J Davison. Visual slam: why filter? *Image and Vision Computing*, 30(2):65–77, 2012.

[108] 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*, 94(2):198–214, 2011.

[109] Jakob Engel, Jörg Stückler, and Daniel Cremers. Large-scale direct slam with stereo cameras. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 1935–1942. IEEE, 2015.

[110] Eckart Michaelsen, Wolfgang von Hansen, Michael Kirchhof, Jochen Meidow, and Uwe Stilla. Estimating the essential matrix: Goodsac versus ransac. *Photogrammetric Computer Vision*, pages 1–6, 2006.

[111] Dai-Duong Nguyen, Abdelhafid El Ouardi, Emanuel Aldea, and Samir Bouaziz. Hoofr: An enhanced bio-inspired feature extractor. In *Pattern Recognition (ICPR), 2016 23rd International Conference on*, pages 2977–2982. IEEE, 2016.

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

[113] Takeo Kanade and Masatoshi Okutomi. A stereo matching algorithm with an adaptive window: Theory and experiment. *IEEE transactions on pattern analysis and machine intelligence*, 16(9):920–932, 1994.

[114] Mark Cummins and Paul Newman. Highly scalable appearance-only slam-fab-map 2.0. In *Robotics: Science and Systems*, volume 5, page 17. Seattle, USA, 2009.

[115] Thomas Whelan, Stefan Leutenegger, R Salas-Moreno, Ben Glocker, and Andrew Davison. Elasticfusion: Dense slam without a pose graph. Robotics: Science and Systems, 2015.

[116] Bastien Vincke, Abdelhafid Elouardi, and Alain Lambert. Real time simultaneous localization and mapping: towards low-cost multiprocessor embedded systems. *EURASIP Journal on Embedded Systems*, 2012(1):1–14, 2012.

[117] Mohamed Abouzahir, Abdelhafid Elouardi, Rachid Latif, Samir Bouaziz, and Abdelouahed Tajer. Embedding slam algorithms: Has it come of age? *Robotics and Autonomous Systems*, 2017.

[118] Lifan Yao, Hao Feng, Yiqun Zhu, Zhiguo Jiang, Danpei Zhao, and Wenquan Feng. An architecture of optimised sift feature detection for an fpga implementation of an image matcher. In *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, pages 30–37. IEEE, 2009.

[119] Dimitris Bouris, Antonis Nikitakis, and Ioannis Papaefstathiou. Fast and efficient fpga-based feature detection employing the surf algorithm. In *Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on*, pages 3–10. IEEE, 2010.

[120] Liang-Chi Chiu, Tian-Sheuan Chang, Jiun-Yen Chen, and Nelson Yen-Chung Chang. Fast sift design for real-time visual feature extraction. *IEEE Transactions on Image Processing*, 22(8):3158–3167, 2013.

[121] KY Lee. A design of an optimized orb accelerator for real-time feature detection. *International Journal of Control & Automation*, 7(3), 2014.

[122] Josh Weberruss, Lindsay Kleeman, David Boland, and Tom Drummond. Fpga acceleration of multilevel orb feature extraction for computer vision. In *Field Programmable Logic and Applications (FPL), 2017 27th International Conference on*, pages 1–8. IEEE, 2017.

[123] Rongdi Sun, Peilin Liu, Jun Wang, Cecil Accetti, and Abid A Naqvi. A 42fps full-hd orb feature extraction accelerator with reduced memory overhead. In *Field Programmable Technology (ICFPT), 2017 International Conference on*, pages 183–190. IEEE, 2017.

[124] Yuliang Pu, Jun Peng, Letian Huang, and John Chen. An efficient knn algorithm implemented on fpga based heterogeneous computing system using opencl. In *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*, pages 167–170. IEEE, 2015.

[125] 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*, 5:2747–2762, 2017.

[126] Michał Fularz, Marek Kraft, Adam Schmidt, and Andrzej Kasiński. A high-performance fpga-based image feature detector and matcher based on the fast and brief algorithms. *International Journal of Advanced Robotic Systems*, 12(10):141, 2015.

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

Titre : Un système de vision pour la localisation et cartographie temps réel

Mots clés : Conception conjointe matérielle/logicielle, traitement d'image, systèmes embarqués, SLAM, GPU, FPGA.

Résumé : L'objectif principal de cette thèse est de proposer un algorithme visuel SLAM et l'étude de la portabilité de cet algorithme sur des architectures hétérogènes. La conception du système nécessite des phases de proposition et de validation de la fonctionnalité de l'algorithme vSLAM, tandis que l'étude de la portabilité inclut l'analyse de la complexité de l'algorithme et des contraintes d'architecture dans une approche de adéquation algorithm architecture. Cette adéquation vise à réduire le temps d'exécution et donc à obtenir des performances en temps réel.

La première contribution consiste à proposer un algorithme intitulé "extracteur HOOFR" qui vise à adresser la partie frontale d'un système visuel SLAM pour la détection, la description et la mise en correspondance des primitives dans l'image. Sur la base d'expériences, notre proposition offre un meilleur compromis entre vitesse et qualité correspondante par rapport à d'autres algorithmes dans l'état de l'art. La deuxième contribution est une nouvelle méthode pour la partie dorsale d'un système SLAM visuel stéréo. L'algorithme proposé utilise des points d'intérêt détectés par l'extracteur HOOFR de sorte qu'il soit désigné par HOOFR SLAM. La complexité de traitement est réduite afin de convenir au système embarqué tout en maintenant une précision de localisation élevée. La troisième contribution de notre travail est la possibilité de mettre en œuvre HOOFR SLAM sur des architectures hétérogènes CPU-GPU où un PC puissant et un système embarqué sont pris en compte. De plus, nous présentons également nos recherches sur l'intégration de la partie frontale sur une architecture SoC embarquée CPU-FPGA.

Title : A vision system based real-time SLAM applications

Keywords : Hardware-Sofware mapping, image processing, embedded systems, SLAM, GPU, FPGA.

Abstract : The main objective of this thesis is to propose of a visual SLAM algorithm and the study of the portability of this algorithm on heterogeneous architectures. The system design requires phases of proposing and validating the functionality of the vSLAM algorithm while the study of the portability includes the analysis of the algorithm complexity and the architecture constraints in a software-hardware mapping approach. This mapping is aimed to reduce the execution time and hence to have real-time performances.

The first contribution consists in proposing an algorithm called "HOOFR extractor" which aims to address the front-end part of a visual SLAM system for detecting, describing and matching image features. Based on experiments, our proposal offers a better compromise between speed and matching quality against others algorithms in state-of-the-art. The second contribution is a new method for back-end part of a stereo visual SLAM system. The proposed algorithm uses key-points detected by HOOFR extractor so that it is denoted as HOOFR SLAM. The processing complexity is reduced so as to be suitable to embedded system while maintaining a high localization accuracy. The third contribution of our work is presenting a capability of implementing HOOFR SLAM on CPU-GPU heterogeneous architectures where a powerful PC and an embedded platforms are considered. Moreover, we also present our researches on emdedding the front-end part on a CPU-FPGA embedded SoC architecture.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France