HAL
open science

Co-design of architectures and algorithms for mobile robot localization and model-based detection of obstacles

Daniel Tortei

To cite this version:

Daniel Tortei. Co-design of architectures and algorithms for mobile robot localization and model-based detection of obstacles. Robotics [cs.RO]. Université de Toulouse III, 2017. English. NNT : . tel-01477662v1

HAL Id: tel-01477662

https://laas.hal.science/tel-01477662v1

Submitted on 27 Feb 2017 (v1), last revised 16 Feb 2018 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*
Cotutelle internationale *Université de Novi Sad*

Présentée et soutenue le *02.12.2016* **par :**

Dániel TÖRTEI

CO-DESIGN OF ARCHITECTURES AND ALGORITHMS FOR MOBILE ROBOT LOCALIZATION AND MODEL-BASED DETECTION OF OBSTACLES

JURY

Branislav BOROVAC	Professeur d'Université	Président
Mohamed AKIL	Professeur d'Université	Rapporteur
Jonathan PIAT	Maître de Conférences	Examinateur
Đorde OBRADOVIĆ	Maître de Conférences	Examinateur

École doctorale et spécialité :
EDSYS : Systèmes embarqués 4200046
Unité de Recherche :
LAAS-CNRS (UMR 8001)
Directeur(s) de Thèse :
Michel DEVY et *Mirko RAKOVIĆ*
Rapporteurs :
Mohamed AKIL et *Branislav BOROVAC*

УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Даниел Тертеи

КОДИЗАЈН АРХИТЕКТУРЕ И АЛГОРИТАМА ЗА ЛОКАЛИЗАЦИЈУ МОБИЛНИХ РОБОТА И ДЕТЕКЦИЈУ ПРЕПРЕКА БАЗИРАНИХ НА МОДЕЛУ

ДОКТОРСКА ДИСЕРТАЦИЈА

Нови Сад, 2016

Abstract

English. An autonomous mobile platform is endowed with a navigational system which must contain multiple functional bricks: perception, localization, path planning and motion control. As soon as such a robot or vehicle moves in a crowded environment, it continously loops several tasks in real time: sending reference values to motors' actuators, calculating its position in respect to a known reference frame and detection of potential obstacles on its path. Thanks to semantic richness provided by images and to low cost of visual sensors, these tasks often exploit visual cues. Other embedded systems running on these mobile platforms thus demand for an additional integration of high-speed embeddable processing systems capable of treating abundant visual sensorial input in real-time. Moreover, constraints influencing the autonomy of the mobile platform impose low power consumption. This thesis proposes SOPC (System on a Programmable Chip) architectures for efficient embedding of vison-based localization and obstacle detection tasks in a navigational pipeline by making use of the software/hardware co-design methodology. The obtained results are equivalent or better in comparison to state-of-the-art for both EKF-SLAM based visual odometry: regarding the local map size management containing seven-dimensional landmarks and model-based detection-by-identification obstacle detection: algorithmic precision over execution speed metric.
Keywords: Robot navigation, visual SLAM, visual obstacle detection, hw/sw co-design, SoPC, heterogeneous embedded boards

Français. Un véhicule autonome ou un robot mobile est équipé d'un système de navigation qui doit comporter plusieurs briques fonctionnelles pour traiter de perception, localisation, planification de trajectoires et locomotion. Dès que ce robot ou ce véhicule se déplace dans un environnement humain dense, il exécute en boucle et en temps réel plusieurs fonctions pour envoyer des consignes aux moteurs, pour calculer sa position vis-à-vis d'un repère de référence connu, et pour détecter de potentiels obstacles sur sa trajectoire; du fait de la richesse sémantique des images et du faible coût des caméras, ces fonctions exploitent souvent la vision. Les systèmes embarqués sur ces machines doivent alors intégrer des cartes assez puissantes pour traiter des données visuelles en temps réel. Par ailleurs, les contraintes d'autonomie de ces plateformes imposent de très faibles consommations énergétiques. Cette thèse proposent des architectures de type SOPC (System on Programmable Chip) conçues par une méthodologie de co-design matériel/logiciel pour exécuter de manière efficace les fonctions de localisation et de détection des obstacles à partir de la vision. Les résultats obtenus sont équivalents ou meilleurs que l'état de l'art, concernant la gestion de la carte locale d'amers pour l'odométrie-visuelle par une approche EKF-SLAM, et le rapport vitesse d'exécution sur précision pour ce qui est de la détection d'obstacles par identification dans les images d'objets (piétons, voitures...) sur la base de modèles appris au préalable.
Mots clés: Navigation du robot, SLAM visuel, detection visuelle d'obstacles, adéquation algorithme architecture, SoPC, cartes embarquées hétérogenes

Srpski. Autonomna mobilna platforma opremljena sistemom za navigaciju podrazumeva više funkcionalnih blokova: percepciju, lokalizaciju, planiranje putanje i upravljački podsistem. U slučaju kada se robot kreće u sredini sa mnoštvom prepreka na putu, on kontinualno izvršava nekolicinu zadataka u realnom vremenu: šalje referentnu vrednost fizičkih veličina za upravljanje aktuatorima, računa svoju poziciju u odnosu na poznati referentni koordinatni sistem u prostoru i detektuje potencijalne prepreke na svom putu. Zahvaljujući kamerama koje pružaju bogatu semantičku informaciju u vidu slika u boji i njihovoj niskoj ceni, ovi zadaci se često baziraju na viziji. Prema tome, standardni namenski sistemi implementirani na ovim mobilnim platformama zahtevaju dodatno integrisanje brzih procesnih sistema sa kapacitetom obrade velike količine vizuelnih podataka u realnom vremenu. Štaviše, zarad što veće autonomije mobilne platforme potrebno je osigurati njenu minimalnu energetsku potrošnju. Ova teza bavi se dizajnom SoPC (engl. System on a Programmable Chip) arhitektura i algoritama za efikasnu implementaciju zadataka lokalizacije i detekcije prepreka baziranih na viziji u kontekstu autonomne navigacije koristeći metodologiju hardver/softver kodizajna. Ostvareni rezultati su ekvivalentni ili bolji od aktualnog stanja u oblasti pri lokalizaciji pomoću vizuelne odometrije bazirane na EKF-SLAM algoritmu i pri detekciju prepreka putem algoritama identifikacije bazirane na modelu. Za lokalizaciju, razvijena je efikasna računarska arhitektura za EKF algoritam, koja podržava skladištenje i obradu sedmodimenzionalnih orijentira u lokalnoj mapi. Za detekciju prepreka je predložena nova metoda prepoznavanja objekata u slici putem prozora detekcije fiksne dimenzije, koja omogućava veću brzinu izvršavanja algoritma detekcije.
Ključne reči: Robotska navgacija, vizuelni SLAM, detekcija prepreka u slici, kodizajn hardver-softver, SoPC, heterogene namenske kartice

Acknowledgements

This doctoral dissertation is the result of a rich past experience that spans far over last four years. It would have been an impossible feat, if not for my childhood education, early manufacturing-related working experience and most importantly the values taught at me at home: discipline, passion to quality work and the need for critical and focused thinking in this world characterized by evermore complex dynamics. I cannot express enough my gratitude to my grandparents: Nagytata (András), Nagymama (Mária), Baka (Ruža) and Deda (Josip), who, with their comprehension, patience and care, nurtured an early flourishment of my creative skills and enabled their very first embodiments. I am eternaly thankful to my parents, Mama (Erika) i Tata (Branislav), who aided me in my endeavors and pushed me into the unknown, while always finding a way to be by my side even when physically apart for several thousand kilometers. Not less worthy was my partner's support, which is proven to be inevitable for sharpening my communicational and social skills. Thank you Brigitta for your daily efforts and sacrifices.

This PhD thesis was conducted under joint supervision of Michel Devy at LAAS-CNRS, Toulouse, France and Mirko Raković at Faculty of Technical Sciences, University of Novi Sad, Serbia. Funds allocated in the 2012-2016 period are:

- *French Government Scholarship* for joint-PhD students,
- FUI-AAP14 project *AIR-COBOT*, co-funded by BPI France, FEDER and the Midi-Pyrénées region and
- Project *Robot Viticole* funded by the Midi-Pyrénées region.

In France, I was integrated in the Robotics, Action, Perception (RAP) group in LAAS-CNRS laboratory and affiliated to University Paul Sabatier (Toulouse III). In Serbia, I was registered at Computation and Control Engineering Department at Faculty of Technical Sciences (University of Novi Sad). During the joint thesis preparation I have had an opportunity to immerse myself largely into French and Serbian customs in terms of private and professionnal life, leaving my zone of comfort well behind me. That resulted in an exciting, multicultural and challenging adventure that improved greately my perception of both Academia and Industry, given these aforementioned research projects I was entrusted with. I would like to express my gratitude to Michel Devy who equivocally believed in my capacities and under whose supervision I was able to fully experience driving on the bulky road of a researcher's work constrained by industrial applications. Thanks to financial support from his behalf, I was able to attend and present my work at the International conference on reconfigurable computing and FPGAs, which had a huge impact on my understanding of the field of SoPC architectures design. I tend to thank Đorđe Obradović for his initial instructions, help and advices. Thanks to the genuine help of Mirko Raković, under whose supervision the dissertation came to its final shape, focus of the research content has been firmly formulated. Special thanks to Jonathan Piat and Bertrand Vandeportaele, for all the advices for both sw/hw conceptual and technical issues. I would like to thank Patrick Danès for his unreserved trust and support and foremost for his time devoted to me during hard times when the joint cooperation seemed almost unsustainable. Thank you Mohamed Akil and Branislav Borovac, and to other members of the jury of the defence of my thesis, for your invested time for reading, evaluating and synthesizing the dissertation. Your comments and critics are

unambiguously assertaining the value of this joint work and, to my great delight, give motivation for consequent research.

As part of RAP group, I had the opportunity to interchange ethical views and conceptions first hand with people originating from Middle East, India, Americas, Ethiopia, Germany and Far East. It left me with tremendeous appreciation towards muslim, latin and hindu traditions and cultural values and a much better understanding of the human being and its potential. Thank you for all cooperation, funny conversations and great babyfoot and table tennis games we had Ariel, Diego, Xavier, Alhayat, Ali, Mustaffa, Wassim, Nemanja, Igor, Dušan, Marko, Borislav, Mukunda, Nirmal, Ganesh, Harmish, Renliw, François, Phillippe-Antoine, Jean-Thomas, Justin, Marcus, Pierrick, Alexandre, Stanislas and Guido.

The technical, scientific and social skills needed to surmount such a joint PhD program are very well summarized in an old Hungarian proverb:

"Szemesnek áll a világ"

"World is upon those who see" (eng. trans.)

because, at the same time, it expresses the need of a physical ability of seeing the world as is with **eyes**, the hypothesis of perceiving the world in a right way by means of **vision**, and the skills requirement of one's **adaptability** and **cunning** to process the acquired knowledge and make something valuable out of it.

Contents

1 Introduction **11**
1.1 General Introduction . . . 12
1.2 Simultaneous Localization and Mapping . . . 14
1.3 Obstacle Detection and Tracking . . . 17
1.4 Organization of the manuscript . . . 20

2 Architectures and Algorithms for Robot Navigation **21**
2.1 Robot Navigation . . . 22
2.1.1 The navigation system for the AIR-COBOT robot . . . 23
2.1.2 Specifications of an embedded navigation system . . . 26
2.2 HW/SW co-design methodology . . . 27
2.3 Conclusion . . . 30

3 State-of-the-art **31**
3.1 Computer vision for object classification . . . 32
3.1.1 Interest Point Detectors and Descriptors . . . 32
3.1.1.1 Raw pixel patches . . . 32
3.1.1.2 Scale-invariant feature transform . . . 32
3.1.1.3 Speeded-up robust features . . . 34
3.1.1.4 Histogram of Gradients . . . 34
3.1.1.5 Other Descriptors . . . 35
3.1.2 Explicit Shape . . . 35
3.1.2.1 K-fans . . . 35
3.1.2.2 Implicit-shape model (ISM) . . . 36
3.1.2.3 Alphabets of visual words . . . 37
3.1.3 Without explicit shape . . . 37
3.1.3.1 Hierarchical object recognition . . . 38
3.1.4 Machine learning for classification . . . 38
3.1.4.1 Boosting . . . 40
3.1.4.2 Support vector machines . . . 41
3.1.4.3 Application in computer vision . . . 41
3.2 Complete real-time video analytics systems . . . 42
3.2.1 Non-embedded solutions . . . 42
3.2.2 Embedded solutions . . . 43
3.3 Visual EKF-SLAM . . . 44
3.3.1 Monocular SLAM implementations . . . 45
3.3.2 FPGA-based SLAM implementations . . . 45

4 EKF-SLAM HW/SW co-design: part I **47**
4.1 Modeling . . . 48
4.2 Hardware specification before partitioning . . . 50
4.2.1 System Cost Measure Definition . . . 50
4.2.2 Update Specifications . . . 52
4.3 Iterative prototyping . . . 53
4.3.1 First iteration: software-only prototype . . . 54
4.3.2 Second iteration: Front-end hardware accelerator . . . 55
4.3.2.1 Corner detector FAST . . . 55
4.3.2.2 Tesselation . . . 58
4.3.3 Third iteration: Back-end hardware accelerator . . . 59
4.3.4 Fourth iteration: Front-end hardware accelerator . . . 61
4.3.4.1 Correlation accelerator co-design . . . 61
4.4 Final hardware prototype . . . 64
4.4.1 Experimental results and discussion . . . 65
4.4.2 Comparison with state of the art . . . 66
4.5 Conclusion . . . 67
4.5.1 Towards EKF-SLAM at 100Hz . . . 68

5 EKF-SLAM HW/SW co-design: part II **69**
5.1 Algorithmic overview . . . 70
5.1.1 Computational bottleneck identification . . . 71
5.2 Design considerations . . . 72
5.2.1 Matrix multiplication tradeoffs on FPGAs . . . 72
5.2.2 Notion of systolic arrays on an FPGA . . . 73
5.2.3 Computational model mapping . . . 73
5.3 Cross-covariance matrix block accelerator . . . 76
5.3.1 HW/SW Integration . . . 79
5.4 Design evaluation . . . 81
5.4.1 Multiplier Latency . . . 81
5.4.2 Resource usage . . . 81
5.4.3 Power consumption . . . 82
5.5 Experimental results . . . 83
5.5.1 Measured multiplication latencies . . . 83
5.5.2 Design scalability . . . 84
5.5.3 Power per feature . . . 84
5.5.4 Future work . . . 84
5.6 Conclusion . . . 86

6 Embedding Algorithms for Visual Obstacle Detection **87**
6.1 Related work and motivation . . . 88
6.2 Proposed framework . . . 89
6.2.1 Appearance kernel . . . 90
6.2.2 Shape kernel . . . 90
6.2.3 Classifiers co-design . . . 94
6.2.4 Classifiers training . . . 97
6.2.5 Single scale decision making . . . 99

6.3 Conclusion 103

7 Towards Real-Time Visual Vehicle Detection 105

7.1 Software architecture 106

7.1.1 Concurrence 108

7.2 Algorithmic Evaluation 109

7.2.1 Estimation of vehicles detection on an embedded platform 112

7.3 Conclusion 114

8 Conclusion 115

List of Figures

1.1 Tasks in a SLAM process 15
1.2 2D SLAM with an obstacle 16

2.1 Robot OZ from Naio Technologies 22
2.2 Robot for pre-flight inspection used in AirCobot project 23
2.3 Functional architecture of navigational module in AIRCOBOT project . . 24
2.4 HW/SW co-design methodology for SoPC prototyping 29

3.1 SIFT keypoints: left image is mirrored and SIFT is applied on both images. The extracted keypoints's locations are invariant to image rotations. 33
3.2 An example of alphabet encoding with 4 cluster centers for intra-class classification: i) interest points are extracted and described with a decriptor in form of encoded feature vectors; ii) each of the feature vectors is assigned to closest visual word from codebook based on a distance measure; iii) histogram containing this combination of visual words is passed to a classifier which determines object instance. 37
3.3 i) AdaBoost's linear kernel vs ii) SVM's Gaussian kernel: an example over a complex, linearly non-separable dataset. 42

4.1 Example dataflow graph 48
4.2 The vision-based EKF-SLAM SDF model 50
4.3 A simple dataflow graph and its one processor schedule 51
4.4 The simple dataflow graph with re-timing (delay token denoted by a red dot) and its multi processor schedule 51
4.5 Re-timed application SDF model 52
4.6 Theoretical schedule of the SLAM aplication before and after re-timing . . 53
4.7 Schedule of the SLAM aplication after first iteration 54
4.8 FAST corner detection result 56
4.9 Image acquisition as a 1-D pixel stream 57
4.10 Fast segment test. Pixels from 1 to 16 form the Bresenham circle 57
4.11 FAST hardware architecture 58
4.12 Corner score module 58
4.13 Tesselation on FAST corner detection result 59
4.14 Schedule of the SLAM application after second iteration 60
4.15 Schedule of the SLAM application after third partitioning 61
4.16 Instantiation of a multicore correlator 63
4.17 Schedule of the SLAM application after fourth partitioning 63
4.18 Embedded SLAM's SoC architecture 64

4.19 Summary over the iterative prototyping 67
4.20 Expected schedule of the SLAM application after fifth partitioning 68

5.1 Our tiling of the 1-D systolic array for visual EKF-SLAM matrix multiplications 75
5.2 Tri-matrix multiplication architecture. 77
5.3 Our SoC implementation of the 3D EKF-SLAM application 78
5.4 EKF block throughput comparison #1 85
5.5 EKF block throughput comparison #2 85

6.1 Equal error rate of a set of classifiers for $N_SET = 300$ 93
6.2 Rule-based BoW classification. 96
6.3 Learning curves for $SVM(K_{raw})$ and $SVM(K_{HoG})$ 98
6.4 Single scale detection window non-maximal suppression scheme 99
6.5 Subtractive clustering in a template-based NMS inferring 101
6.6 Block diagram of the proposed framework. 103

7.1 Partial SIFT KEYMAP, after having applied K_{BoW} on extracted SIFT keypoints 107
7.2 Parallelization of stages I and II 108
7.3 Tasks timing in hierarchical detection pipeline 109
7.4 Tasks timing without linear SVMs computation 110

List of Tables

1.1 Visual SLAM implementations - an overview 18

3.1 Comparison of generative and discriminative classfiers 39

4.1 vSLAM tasks execution before iterative prototyping 53
4.2 Prototype's resource usage with IPs' performance 65
4.3 Execution time of SLAM functional blocks in [ms] on ZedBoard 66

5.1 Description and matrices dimensions according to our implementation of the EKF algorithm. 71
5.2 Number of floating-point operations in an EKF loop taking into consideration symmetry of the cross-covariance matrix. Equations 5.4 - 5.7 are executed c = 20 times, for each observation 71
5.3 Transfer analysis regarding P matrix in EKF equations 72
5.4 Control of *en* signals 76
5.5 1-D systolic array resource usage 81
5.6 Accelerator resource usage with on-chip memory 82
5.7 Visual EKF-SLAM accelerator's power distribution: comparison with the previous version 83
5.8 EKF equations cycles measured with 125 MHz counter 83
5.9 Device resources usage with $N_{max} = 70$ 84
5.10 Power per feature metric for visual EKF-SLAM with AHP points 85

6.1 Histogram of Gradients: an empirical study 92
6.2 SVM kernels: an empirical study 97

7.1 Stage II training 110
7.2 Stage I training 110
7.3 Stage I implementation 111
7.4 Stage II implementation 111
7.5 Stage III implementation 111

Chapter 1

Introduction

Contents

1.1 General Introduction 9
1.2 Simultaneous Localization and Mapping 11
1.3 Obstacle Detection and Tracking 14
1.4 Organization of the manuscript 17

1.1 General Introduction

Increasingly applications rely on mobility whatever the system - be it a robot, a vehicle, a drone or an operator with a tablet or a smartphone, equipped with several sensors. As soon as such a system moves in an environment, it must be endowed with several functions able to provide its position with respect to a known reference frame. It also needs to detect obstacles in order to avoid collisions (for robot navigation) or to recognize some known objects (for augmented reality). Main subject studied in this thesis concerns the localization and obstacle detection functionalities in the context of navigation of a mobile platform.

Many solutions exist in the literature, depending on the available sensory modalities, on the locomotion system, on the a priori knowledge learnt on the environment, and overall, on the available resources (energy, memory, computing power). Many robots (e.g. Automated Guided Vehicles (AGV) for industrial environments) and vehicles (e.g. the Google car) are equipped with laser range finders in order to perceive the environment; in this thesis it is proposed to use only vision, i.e. a low-cost sensor that could be embedded both on autonomous robots and on smart devices moved by humans.

A standard Visual Odometry (VO) algorithm is exploited for the localization of our system moving in an initially unknown environment. Such a function builds up a local map that corresponds to its environment in the form of positions of *landmarks* observed in images, estimates the local motion made by the system, and then, cumulating these motions, estimates the system position with respect to a specific reference frame, generally the initial position when the VO algorithm has been started. Landmarks are presumed to be static. Our mobile system must be equipped with a camera and besides it may also have proprioceptive sensors in order to give other measurements on the motions: wheel odometry for a robot, Inertia Measurement Unit (IMU) for a device worn by an operator or for a drone.

A VO algorithm is very similar to a Visual Simultaneous Localization and Mapping approach (VSLAM), but the localization in VO is only estimated from a local map. Such a map is built over a given horizon time, defined e.g. by the M last acquired images or by the N last detected landmarks. As the localization is computed from local information, it will be impacted by a drift growing over time. With a Visual SLAM algorithm, it is possible to build a global map, even on large environments: if the mobile system returns in a known area and succeeds in matching observed features with mapped landmarks, cumulative errors may be corrected using a loop closure function. VO and VSLAM could be based either on Filtering, e.g. EKF and Rao-Blackwellized Particle Filter, or on Optimization (global or incremental, iterative or direct). It is known [Strasdat et al., 2010] that the localization given by Optimization methods is more accurate than for Filtering ones, but it requires more computing power, and moreover, for large environments, the map size could be huge, so that some map management strategy has to be applied.

It is the reason why, for systems of limited size, an EKF-based VO algorithm is most often applied for position estimation; moreover it is easier to take into account measurements acquired from asynchronous multiple modalities (several cameras, gyros and accelerometers). Drifts could be corrected from time to time either fusing with a priori knowledge given to the system (e.g. Open Street Map in urban scenes, or a building map for indoor scenes) or with a global position provided by a GPS receiver in outdoor environments.

In spite of the increasing computing power on compact embedded system, EKF-based VO's computational complexity still may be too important with only a software implementation. It is so mainly due to image processing in order to extract features from images and due to large scale matrix-matrix multiplications when applying the filter on large state vectors. The real-time constraint for typical applications of drones, robots or augmented reality on smart devices, involves to update the system position at least at $30Hz(33.33ms)$. Moreover improving the frequency allows to enter a virtuous cycle: faster is the VO process, easier and more robust is the feature-landmark matching. Another issue is that of autonomy for a mobile system: standard processors, such as Intel's quad core i7, consume a lot of power, thereby limiting the level of autonomy.

The SLAM and obstacle detection functionalities are interconnected. First as the SLAM paradigm itself is used for both self-localization of the mobile platform and cartography of the static world around the moving platform, obstacle detection plays an important role in providing reliant visual cues only on static entities. Moreover estimating the speed of the mobile platform helps to characterize other mobile objects sharing the same world. In mobile robotics (including drones, micro-aerial vehicles (MAVs), terrestrial rovers, autonomous driving vehicles), the navigation performance in crowded environments depends heavily on efficient obstacle detection. In Advanced Driver Assistance Systems (ADAS) and Intelligent Transportation Systems (ITS) early collision avoidance systems play a crucial part in safe navigation. As vehicle speeds in urban areas may reach 50km/h, real-time execution of obstacle detection task at 10Hz implies that a single image is to be fully treated at each 1.39m of path traveled.

Cutting edge computer vision algorithms, benchmarked on public data-sets such as KITTI [Geiger et al., 2013] require huge amounts of processing power to deal with this issue: they most often rely on Graphical Processing Units (GPUs) to compute intensive image-related operations or multi-core personal computers with tens of gigabytes of RAM. Well-performing image indexation algorithms such as [Harzallah et al., 2009] cope with multiclass object detection by creating large computational kernels for non-linear Support Vector Machines (SVMs). In the literature this is known as "the curse of dimensionality". Calculating dense optical flow or depth map from stereo camera setup in real-time is unimaginable without a GPU or a dedicated hardware platform. Other known techniques for foreground/background segmentation are mainly used for visual monitoring of environments from static cameras, such as Gaussian Mixture Models (GMMs) which come with heavy computational load.

To overcome these difficulties, an embedded system ensuring high computational efficiency at low power consumption must be designed. Modern reconfigurable devices such as Field-Programmable Gate Arrays (FPGAs) may meet this requirement. Their reconfigurable fabric consists of a large number of configurable slices, and embedded Digital Signal Processing (DSP) blocks that can be used for floating-point applications. Compared to a standard processor, computationally extensive algorithms may be parallelized and thus executed several times faster on FPGAs in floating-point precision [Underwood and Hemmert, 2004]. While Graphical Processing Units (GPUs) also make use of the parallelization techniques, the main advantage of FPGAs over both GPUs and general purpose central processing units (GPCPUs) remains in significantly lower power consumption. Embedded GPUs have as well emerged on the market today with similar power consumption to an FPGA [NVIDIA, 2015b]. However, they are designed as to improve computational performances on highly demanding image processing applications.

Thus FPGAs with their algorithm-specific memory management capabilities owing to reconfigurability still remain a better choice over embedded GPUs in terms of computational efficiency of a custom application.

Hence, migrating SLAM and obstacle detection algorithms on systems of limited size, such as MAVs and intelligent vehicles, is most often done by developing dedicated hardware accelerator architectures on Field Programmable Gate Arrays (FPGAs) [Advani et al., 2015], [Chang et al., 2013], [Wilson et al., 2014], [Hahnle et al., 2013]. Their convenient size, powerful parallel processing capabilities and low power drain (˜5W vs ˜300W for an average GPU) make them an optimal tool for embedded systems development. Nowadays, the embedded industry is shifting towards issuing heterogeneous hardware architectures which port besides an FPGA (which already comes with a multi-core hard processor such as ARMv7) also an embedded GPU [Xilinx, 2015].

Contribution of this thesis concerns the localization and obstacle detection functionalities by identifying the underlying particularities within visual SLAM and computer vision algorithms: their complexity, inherent parallelisms and memory management specificities. It describes and evaluates an efficient FPGA-based hardware accelerator for solving the localization task and proposes a model-based obstacle detection framework to be executed on multi-core heterogeneous[1] hardware platforms.

1.2 Simultaneous Localization and Mapping

Simultaneous Localization and Mapping (SLAM) is a paradigm that intends to infer structure of an initially unknown environment from sensors moved by a mobile platform or by a person while performing self-localization of this platform or person by matching observations with the incrementally built map. In literature, this localization-mapping iterative process is known as "chicken or egg" problem, while in practice the answer to "where am I" question is given after an initial observation of the environment. To date, several thousand scientific and engineering papers have contributed to solving this intriguing puzzle, with the cornerstone being laid by [Cheeseman et al., 1987] who used landmarks in the environment in order to construct its map. SLAM may be divided into six separate tasks, as can be seen on Figure 1.1:

Initialization is the first step in establishing a priori information about the unknown world by identifying first landmarks in the vicinity of the robot via exteroceptive sensors. After having performed a motion, it calculates its new position based on proprioceptive sensors (odometers, accelerometers, gyros, magnetometers) and/or dynamic displacement model- it ***estimates*** its new position. Because of incertitude on data readings which eventually leads to a faulty pose estimation, it has to compare features observed from the current position with potential observations of known landmarks so it can estimate its pose relative to the point of departure (origin). For that it does ***feature detection*** anew, following by a posteriori ***data association*** with cues collected in initialization step. Upon successful data matching, the robot may ***update its state*** (localization) and ***update landmarks' positions in the map*** (mapping). As the process is iterative, a new loop initializes new landmarks and the SLAM goes on in same fashion.

A hypothetical two-dimensional SLAM process is depicted on Figure 1.2. Here, a holonomous robot is passing from state x_1 to x_5 while avoiding an obstacle. Initially, at

[1] multi-core CPU + FPGA

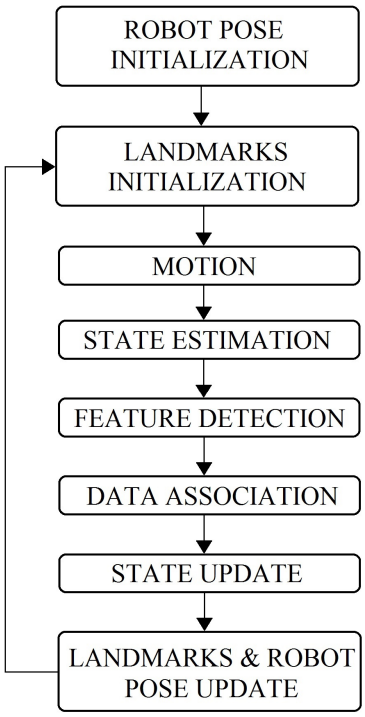

Figure 1.1: Tasks in a SLAM process

state x_1, it observes two landmarks L_1 and L_2. After having moved, it reobserves L_2 and updates its position along with the positions of both first two landmarks in its map. At x_2 three new landmarks are initialized: L_3, L_4, L_5 and at x_3 the fifth is reobserved, upon which it is able to adjust all the five landmarks in the map. At the final state x_5 robot's map contains nine landmarks.

All tasks involved in feature acquisition, data association and map management belong to so-called *front-end* part of SLAM while the tasks related to the state estimation and update both of the robot's state vector and of the map are alltogether known as *back-end* SLAM. Along past decades, the community has proposed different algorithmic solutions to these two distinct parts of the paradigm, treating several major problems arising from data uncertainty that eventually lead to loss of localization and mapping accuracy. Data uncertainty may be caused by unpredictive physical phenomena, environment conditions, material properties of objects and quality of the sensors.

A state-space based estimator, the Kalman filter [Kalman, 1960] - KF for short, was first used to model the dynamic system behavior, from imprecise sensory data. It functions well under two assumptions: the system is modelled as a linear system and transitional processes are affected by Gaussian-characterizable noise. For nonlinear system model, an Extended KF [Maybeck, 1979] - or simply EKF is used. The important particularity of an EKF-SLAM is the ability of updating the whole state-space vector based on a measure of its single element (observation) by means of convolving innovation with cross-correlation matrix. EKF-SLAM's algorithmic drawback comes after unsuccessful data association, due to linearization errors in state estimation and update phases, known as *filter inconsistency*. A study was conducted by [Castellanos et al., 1997] on impact of correlating mapping with state-space vector, where it is shown that tightly coupled systems contribute to more accurate mapping (*consistent maps*) than loosely coupled

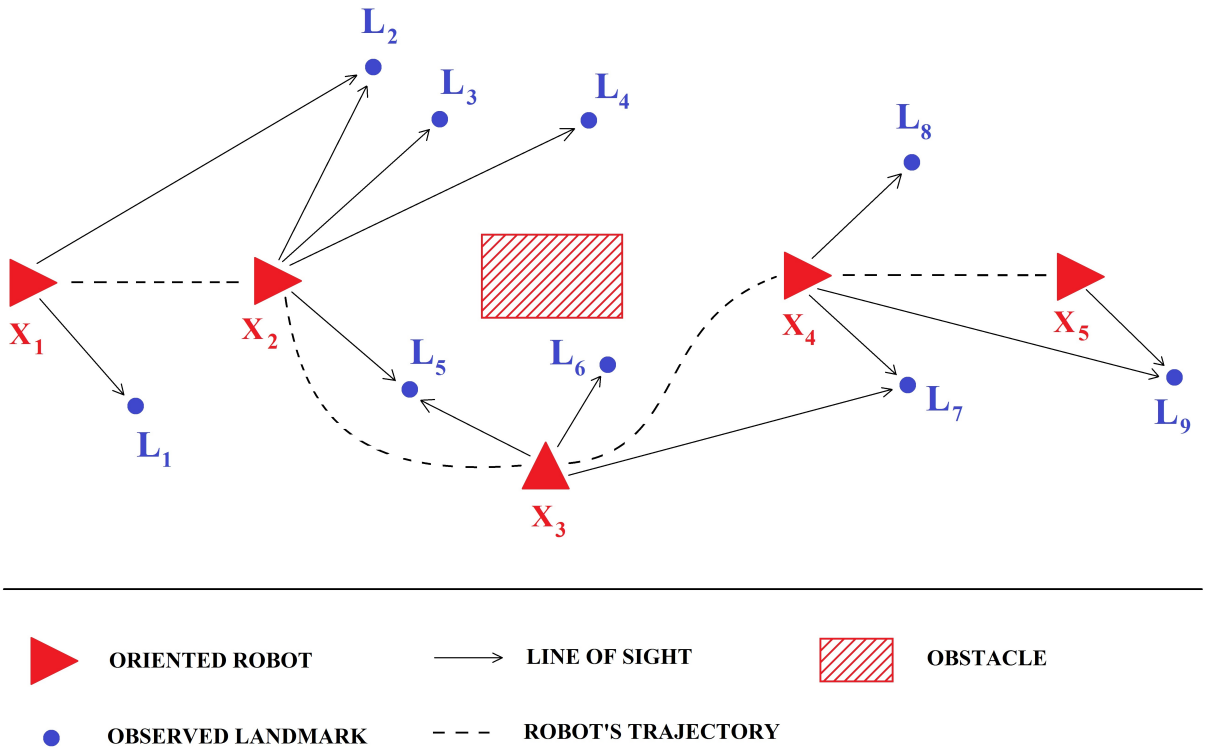

Figure 1.2: 2D SLAM with an obstacle

ones. In addition, the theoretical work of [Dissanayake et al., 2001] showed the need of having all the information regrouped in a single cross-covariance matrix for the sake of filter's good convergence.

First exteroceptive sensors to be used were laser rangefinders [Betge-Brezetz et al., 1996] [Csorba, 1998] - fast and providing confident 3D observations, but being too costly for larger-scale SLAM use. It is the reason why the robotics community has studied visual SLAM for almost fifteen years, taking profit of the existing "Structure from Motion" contributions of the Machine vision community. Use of a multi-camera setup with odometry for localization was proposed first by [Harris and Pike, 1987]. Three-dimensional point representation used in this work raised new questions about point parametrization during initialization phase due to perspective 3D (real-world) to 2D (camera plane) projection, as they didn't make use of stereo setup in their multi-camera configuration in order to acquire depth. There, several landmark observations are required to build landmark's 3D representation. Their work is equally valuable in presenting *active search* strategy, a technique used for limiting the search space in image plane during feature tracking phase, which greatly increases processing speeds in comparison to brute-force matching over the entire image.

The first ground-breaking work on monocular real-time SLAM was done by [Davison, 2003a]. From sensors, he used only a low-cost webcam which was connected to a those-days standard personal computer (PC). Mapping was done in three dimensions with a tightly-coupled back-end, using a delayed initialization of landmarks, based on a particle filter (PF). The depth uncertainty of each landmark is reduced over time while reobserving it during displacement. Once the uncertainty passes under a certain threshold, the point is added to the map. In order to keep real-time performance, a limited

amount of landmarks is held within the map.

In literature, based on environment representation (front-end) we may distinguish between different types of SLAM: semantic SLAM, topological SLAM, and geometric SLAM. Semantic SLAM, also known as *object-based* SLAM, uses object detection for self-localization. Here, the most common representative is grid-based SLAM (i.e. FastSLAM2.0 [Magnenat et al., 2010]) which uses laser telemetry to construct a map which is a discrete occupancy grid of robot's vicinity. Topological SLAM (i.e. TSLAM [Choset and Nagatani, 2001]) uses known topological maps (forms of structured graphs) of the environment for localization. Geometric SLAM uses low-dimensional spatial priors as landmarks: points, line segments, circular arcs.

On the other hand, based on back-end methods, the SLAM corpus may be divided into following categories: probabilistic, graph (optimization-based), and interval propagation SLAM. Probabilistic SLAM uses filtering techniques such as EKF and PF (i.e. FastSLAM [Montemerlo et al., 2002]) for state-space vector transformations. Graph SLAM [Kaess et al., 2011] [Kümmerle et al., 2011] makes use of energy optimization in a generalized graph whose vertices are ensembles of landmarks and robot poses through all past frames. The interval propagation SLAM methods transform the SLAM problem into a hybrid constraint satisfaction problem (CSP), which estimates a set-valued representation of the state [Jaulin, 2011].

This thesis deals with a monocular EKF SLAM that builds 3D maps of robot's environment for localization. An overview over existing monocular SLAM implementations may be seen on Table 1.1. Relevant methods will be cited and briefly described in Chapter 3. The more recent methods are based on optimization, but considering constraints related to embedded systems, filtering-based approaches remain interesting.

1.3 Obstacle Detection and Tracking

Obstacle detection, and overall Mobile Object Tracking (MOT), are critical functionalities in robotics or intelligent transportation systems, so many works have been devoted to this topics. As mentionned, SLAM and MOT are interconnected, mainly because features created by mobile objects in sensory data, could be considered as perturbations for the SLAM methods, as it was analyzed in [Márquez-Gámez, 2012]. Obstacles could be detected from many sensors, depending on the constraints upon minimal and maximal distances between the robot/vehicle and the obstacle to be detected. The more classical sensor used for this function on a terrestrial robot or for a vehicle, is a 2D laser range finder (LRF), scanning the environment on a plane parallel to the ground: it is known that it is very efficient, but it cannot detect obstacles under the laser plane (short objects on the ground) and the laser data are too poor in order to recognize the obstacle class (car, pedestrian, bicycle, etc.) with a good success rate. It is the reason why vision has been also used for obstacle detection for many years, either alone or tightly coupled with the 2D LRF. Here only obstacle detection and MOT are considered from vision.

In general, image indexation (content retrieval) is a two-stage hierarchical process which consists of

I. Hypothesis generation on object location within an image

II. Hypothesis verification

Table 1.1: Visual SLAM implementations - an overview

SLAM implementations	**Features**		**Mapping**		**Pose estimation**	
	Descriptor-based	Direct	Dense	Sparse	Optimization	Filtering
PTAM [Klein and Murray, 2007]	yes			yes	yes	
DTAM [Newcombe et al., 2011]		yes	yes		yes	
LSDSLAM [Engel et al., 2014]	At loop closure	yes	yes		yes	
ORBSLAM [Mur-Artal et al., 2015b]	yes			yes	yes	
SVO [Forster et al., 2014]	At keyframes	yes		yes	yes	
RTSLAM [Roussillon et al., 2011]	yes			yes		yes
CSLAM [Gonzalez et al., 2011]	yes			yes		yes
2D EKF-SLAM [Bonato et al., 2008]	yes			yes		yes
3D EKF-SLAM [Vincke et al., 2012b]	yes			yes		yes
3D EKF-SLAM w/ submaps [Lee and Lee, 2013]	yes			yes	yes	yes

In the first stage and based on sensor setup, obstacle detection methods in computer vision and image processing follow two main directions: monocular and stereo (binocular) camera detection. The latter takes advantage of the Inverse Perspective Mapping (IPM) [Mallot et al., 1991] to estimate the locations of obstacles. In case of vehicles, [Bertozzi and Broggi, 1998] IPM was computed from the left and right images and based on the comparison objects not belonging to ground plane (road presumed to be flat) were identified as vehicles. Another classical approach for obstacle detection from stereo-vision is based on the v-disparity [Labayrade, 2004], which requires also a flat ground. Dedicated architectures have been proposed both for IPM or v-disparity approaches [Botero-Galeano, 2012] [Irki et al., 2013]. Main problem with obstacles detection by stereo-vision is that it is very sensitive to the camera calibration parameters (extrinsics) and range of the detection is limited to about 10m upfront the sensor rig (depending on the stereo baseline - wider the baseline, higher the range).

Hypothesis generation in monocular setup may be further divided into three categories: (1) constrained IPM (2) motion-based and (3) knowledge-based. An IPM-based on-road obstacles detection system in urban areas is presented in [Itu and Danescu, 2014]. Executing on a smart-phone and making use of its main camera and accelerometer, the algorithm makes assumptions of an upfront flat road free from obstacles. Upon image transformation, color-based segmentation is used for obstacle areas identification. Motion-based approaches calculate optical flow, generating a displacement vector for each pixel - [Bruhn et al., 2005] [Stein, 2004]. High computational complexity of optical flow based approaches still remains a challenge for a straightforward implementation on an embedded platform. Knowledge-based methods employ various abstractions on object representation i.e. about shape, color, texture as well as contextual information about its environment: streets for pedestrians and vehicles, freeways for trucks .... Object location is searched for via sliding-windows (brute force) approach [Harzallah et al., 2009] or after some scene geometry-related assumption - i.e. vanishing point calculation [Hartley and Zisserman, 2004] or edge detection [Sun et al., 2002] for image search-space reduction.

In the second stage, methods that corroborate hypotheses on object location can be classified into two categories: (1) template-based and (2) appearance-based. Template-based approach defines an object class pattern. Each input sample (sub)image is correlated upon the class representative. Authors in [Betke et al., 1996] proposed a multiple-vehicle detection approach using deformable gray-scale template matching. Appearance-based methods detect the presence of an object in the scene based upon characteristics of the object class from a set of training image patches which capture the variability in object appearance. Local and/or global characteristics called features are extracted and most often learned by a binary classifier (e.g. convolutional neural network [Krizhevsky et al., 2012] or support vector machine (SVM)). These knowledge-based methods tend to be used alone, both for hypothesis generation and verification. They are now very popular, thanks to the massive distribution of low cost cameras on the market, to the success of the Viola&Jones's approach to detect faces on images using an AdaBoost classifier, and also, to the emerging use of Deep Learning for this topic.

In order to ease object redetection in successive frames, detected obstacles may be tracked. In this scenario a tracklet is initialized for each detected obstacle. Then, inter-frame object's position estimation is most often performed by a Kalman filter, following data association via a graph-search based algorithm (i.e. the Hungarian method). Only

tracklets that are at least three frames long are taken into further consideration.

1.4 Organization of the manuscript

The main body of this thesis is divided as follows.

Chapter 2 describes applicative domain, raises designing constraints and precises evaluation metrics of the research. Consequently it introduces and explains the methodology used for conception, emulation, design and validation steps of software and hardware components developped during my thesis.

Chapter 3 references state-of-the-art and relevant visual EKF SLAM applications and computer vision techniques used for model-based object recognition.

Chapter 4 describes iterative prototyping on the given visual EKF SLAM application ending with a functional SoPC model capable of executing at 100Hz.

Chapter 5 puts forward the co-design with evaluation metrics of FPGA based EKF co-processor's architecture.

Chapter 6 introduces a framework for model-based obstacle detection with focus on embedded target devices. A theoretical basis with relevant statistical studies has been profoundly explained.

Chapter 7 begins with a software implementation of previously described framework. Afterwards it presents results after a case-study evaluation on the exemple of *vehicles* type of obstacles.

Chapter 8 summarizes the thesis, exposes the remaining questions to be asked, and gives an outline of the subsequent work enabled by my research.

Chapter 2

Architectures and Algorithms for Robot Navigation

Contents

2.1 Robot Navigation **19**

2.1.1 The navigation system for the AIR-COBOT robot 20

2.1.2 Specifications of an embedded navigation system 23

2.2 HW/SW co-design methodology **24**

2.3 Conclusion **27**

2.1 Robot Navigation

In the context of this thesis the applicative domain of autonomous navigation includes following environment types:

- Indoor structured: popular off-the-shelf solution for these kinds of constrained environments is Robotics Operating System's ROS-nav-stack. It usually uses grid-based SLAM methods for localization.
- Outdoor structured (urban robotics) or large indoor, assuming robot motions on a flat terrain with possibly cluttered environment: service robots in factories, airports (i.e. in the context of AirCobot project[1]). Obstacles may be also dynamic (people, bikes, vehicles, other mobile robots) and this problem is too large for classical SLAM methods.
- Outdoor natural (typically agricultural robotics - Figure 2.1[2]): uneven and 3D terrain, with few dynamic obstacles, but many static obstacles on the terrain (ditches, puddles, tools on the ground).

Figure 2.1: Robot OZ from Naio Technologies

and different navigation methods:

- Reactive navigation: complementary to planned motions and used for obstacle avoidance i.e. the robot leaves the nominal trajectory if an obstacle is found on the path (i.e. in the context of AirCobot project).
- Sensor-based navigation: motions are defined as a sequence of sensor-based commands. Typically it is used when navigating in a field following successive raws, or navigating around an aircraft based on different targets.
- Trajectory-based navigation: robot learns the map of an obstacle-free environment and the motion planner finds a trajectory consistent with the robot's characteristics.

[1] image courtesy of Airbus Group
[2] image courtesy of Naio Technologies

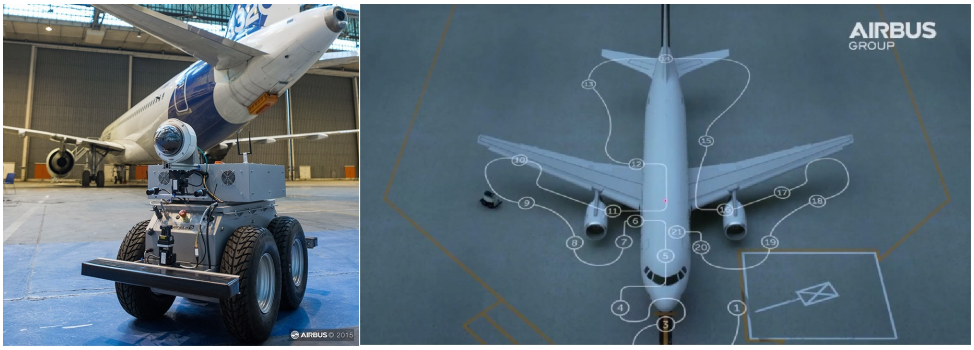

Figure 2.2: Robot for pre-flight inspection used in AirCobot project

It then executes this trajectory, based on a localization method (along with obstacle avoidance) in order to generate a control law.

Before giving some conceptual constraints for embedded navigational systems, in next subsection the system developed for the AirCobot robot by AKKA and LAAS will be presented. Here, the central role of the localization function on such a system will be accentuated.

2.1.1 The navigation system for the AIR-COBOT robot

During the preparation of this thesis, PhD candidate participated on the two projects illustrated on figures 2.1 (on agricultural robotics with the start-up NAIO Technologies) and 2.2 (on service robotics in airports with a consortium lead by AKKA Research and AIRBUS). Contribution on these projects was devoted to the evaluation of several visual SLAM methods for Visual Odometry (VO): the VO results are typically fused with motion estimates provided by wheel odometry and by inertial measurements, to improve the localization obtained from the integration of these motion estimates. It is well known that the odometry based localization drifts. So dead-reckoning navigation, based on these position estimates, can only work locally: this strategy is used only when other localization methods integrated on the robot fail.

By now, the VO function is executed in software, consuming important computational resources on the embedded system. A long term objective for these two projects is to integrate the VO method on a dedicated hardware platform, i.e. to generate a smart camera or stereo sensor specialized for visual odometry.

A simplified navigational schematic with its functional architecture, executed on the AIRCOBOT demonstrator (Figure 2.2 on the left) in a complex operational environment, is presented on Figure 2.3. This robot assists a pilot for the pre-flight inspection task, performed on the parking area before every departure. It must navigate autonomously around an airplane, going successively to every check point (from 1 to 21, presented on Figure 2.2 on the right). These motions must be executed on the taxiway, avoiding vehicles, trucks or operators executing other tasks (refueling, baggage loading, etc.), knowing that close to the airport buildings or to the airplane, GPS-based localization is not reliable. Moreover, airlines or AIRBUS prevent the robot navigating in some areas around the airplane, typically under the wings to avoid collisions with reactors. Before any mission, during an interactive initialization step, the robot learns several representations:

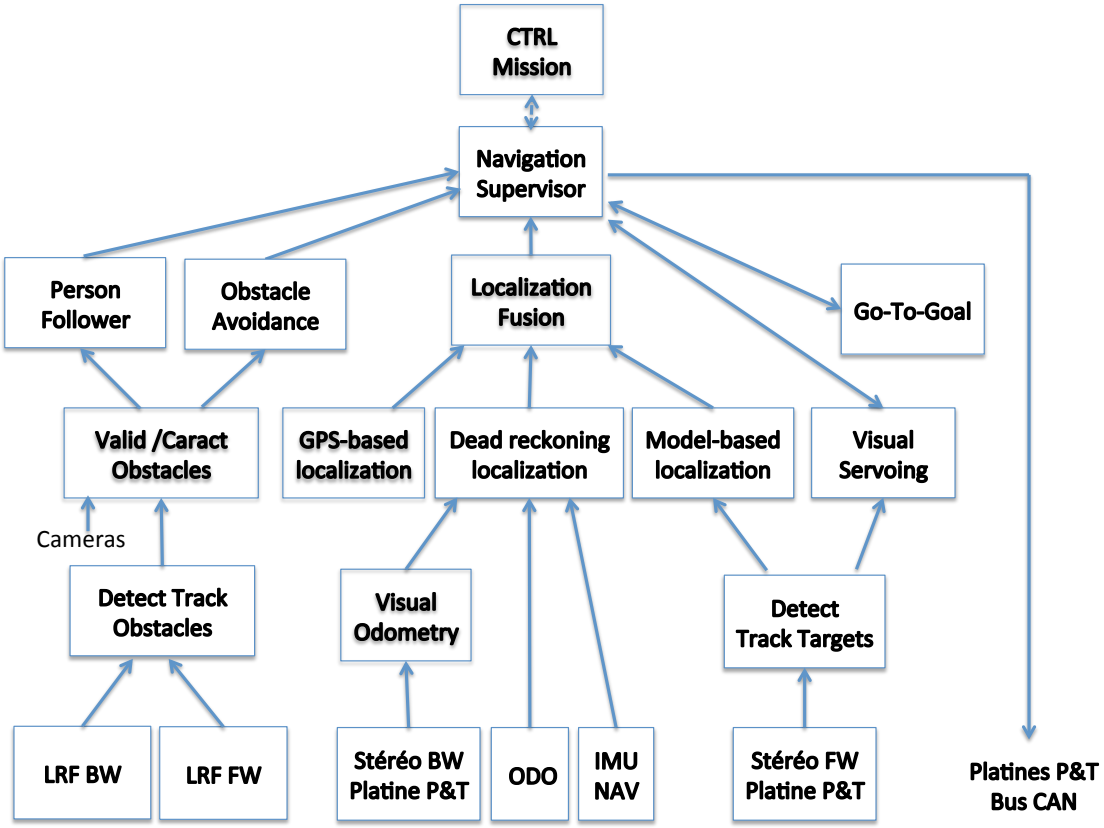

Figure 2.3: Functional architecture of navigational module in AIRCOBOT project

- first the airplane model (here an A320) expressed in the aircraft reference frame and the visual targets it could detect, track and use to control motions around the airplane,
- then the airport model expressed in the airport reference frame, mainly along the trajectories that the robot will have to execute between its docking station and the different parking areas where the pre-flight inspection must be performed,
- and finally the transform between the GPS geo-centered frame (WGS84 or ECEF) and the airport reference frame through a local orthographic projection.

Many modules have been integrated to deal with obstacle detection using laser and visual data (Figure 2.3 on the right), robot localization fusing dead-reckoning, model-based and GPS-based methods (Figure 2.3 in the middle) and motion control based on four modes:

- Visual Servoing activated typically to reach precisely a check point based on image-based control from visual targets detected and tracked on the airplane surface.
- Go-to-Goal used to execute trajectories defined as a sequence of way points expressed either in the airport or in the aircraft reference frame.
- Obstacle Avoidance in order to locally avoid an obstacle, maintaining the camera pointed towards the visual targets if required.

- Person follower: the AIR-COBOT robot may also be used as a cobot, i.e. executing collaborative tasks with an operator, especially following him until a given point where it could execute an inspection task, or following him in order to avoid an obstacle when the autonomous avoidance strategy fails.

Motions are planned by the navigation supervisor knowing the mission to be executed (typically, the next check point to be visited): it activates a navigation mode based on different events (obstacle detected close to the trajectory, detection of visual targets, failure of the tracking, failure of the obstacle avoidance, etc.); it combines low-level orders generated by these modes before sending them to the micro-controllers through the CAN bus. This architecture shows the central role of the localization functions. Whatever the navigation mode, the robot position with respect to the airport and the airplane reference frames are updated using available observations. This knowledge allows the navigation supervisor to plan trajectories, to select a better mode and to switch smoothly between modes . The dead-reckoning localization is only taken into account when GPS-based or model-based methods fail (either due to occlusions for GPS, or to failures of the feature detection or matching functions for vision); when a GPS or a model-based localization is available, it allows to correct the drifts and to maintain the localization error below a maximal value (typically 1.5m for GPS, 1m for vision), consistent with the other modules.

This navigation system is implemented on an embedded PC (2,3GHz quad core i7): it has been integrated using the middleware ROS and some ROS standard modules:

- the drivers used to acquire sensory data: LRF modules linked to the 2D laser range finders at 40Hz and image acquisition at 15Hz

- and the module Robot-Pose-EKF integrated to perform the dead reckoning localization from a loose fusion of three methods executed in order to estimate the robot motions (Visual odometry, Wheel odometry and Inertial measurements).

In this context we evaluated several VO methods: first CSLAM from LAAS, then ORB-SLAM proposed recently by authors from the University of Zaragoza:

- CSLAM will be described in detail in chapter 4; it performs a tight fusion between wheel odometry, inertial measurements and visual observations in order to update both the robot state (position and speed) and landmarks positions. It could replace the Robot-Pose-EKF module. In the AIR-COBOT context, it gave poor results, mainly because we could not guarantee a good synchronization between all sensory data.

- ORB-SLAM exploits only visual data in order to estimate the camera trajectory by an optimization method. It provides better position estimates as it makes use of a large number of landmarks (approx. 2000) and performs loop closure technique when revisiting known locations in order to correct for drifts. The cost of using it involves higher computational latencies and active memory requirements (its visual vocabulary takes only approx. 1GB of RAM while maximum operating frequency does not exceed 10Hz).

2.1.2 Specifications of an embedded navigation system

In the AirCobot project, all functionalities were implemented in software as ROS nodes. If an embedded device supporting these functionalities on a mobile operating platform is to achieve satisfactory behavior, some constraints must be taken into consideration:

- portability : one of the paramount conditions of an embedded device is that it should be small in dimensions and thus easily integrable into different systems,
- processing power : it should dispose of high processing power capabilities, as the problem we are treating is deeply rooted in computer vision and localization domains,
- power consumption : level of autonomy of a mobile platform is directly dependent of energy it consumes for its functioning. E.g. in mobile robotics and in automotive industry putting an Intel Core i3 processor (e.g. i3-4170 launched Q1'15 dissipating 54W at 3.70 GHz) or an Intel Core i7 processor (e.g. i7-6870HQ launched Q1'16 dissipating 45W at 3.6 GHz) in the mobile platform is not affordable due to power issues. So most of the processors that are used have an ARM core (Cortex-A8, A9 and nowadays A15 which dissipate roughly 1W at 1.2 GHz per core) plus a DSP/FPGA inside it. Thus, for higher levels of autonomy, the embedded design needs to take into consideration a logic that is both performing and minimalistic in terms of resource usage,
- accuracy : the most crucial characteristic of the device is to detect well the obstacles and to perform an accurate self-localization.

Real-time measure is based on 30 frames per second (fps) where the embedded system needs to provide thirty robot position responses in a second. Apart algorithmic complexity issues, second biggest and equally important challenge for this thesis is sw/hw integration. It is envisaged to independently provide an embedded solution for SLAM algorithm and for obstacle detection subsystem, respectively. Each subsystem needs to react in real-time, which brings up integration issues in each subsystem as well. Main problems that arise are:

- memory handling and custom data storage techniques,
- adapting the coding technique to "pixels on-the -fly treatment", which is different than the usual when storing images on disk,
- data synchronization of different sensors, namely of odometers, GPS (Global Positioning System), IMU (Inertia Measurement Unit) and camera frames,
- custom, application-specific, logic that accommodates processing power with energy consumption and resource usage,
- software behavioral simulation, functional hardware verification and insuring timing constraints satisfaction of hardware components.

An adequate sw/hw partitioning methodology, called sw/hw co-design, needs to be proposed in order to efficiently balance the aforementioned trade-offs. Thus this thesis will

mainly focus on co-design modeling of hardware architectures on SoC (System On a programmable Chip) and software coding techniques such as single-threaded, distributed and multi-threaded and alike stand-alone (so called bare-metal) and Linux-based applications.

Efficiency measures against which proposed designs for visual SLAM and obstacle detection and identification will be evaluated are:

- satisfaction of real-time constraints, that is how many frames per second an architecture is capable of processing,
- accuracy, DET (Detection Error Tradeoff [Hanley and McNeil, 1982]) curve for embedded classifiers and absolute difference in position for self-localization
- "performance per watt" [Afonso et al., 2011] measured in FLOPs (FLOating Point operations) over power usage of the embedded device,
- resource usage of the hardware components.

2.2 HW/SW co-design methodology

In literature, there is a plethora of works related to developing suitable methodologies to prototype an embedded system. A subset of these refers to heterogenous architectures and they can be summarized - [Shaout et al., 2009] - as: (A) Top-down approaches where from the behavioral system specification the target architecture is gradually generated by adding implementation details to the design; (B) Platform-based design where predefined system architecture is used for mapping the system behavior; and (C) Bottom-up approach where a designer assembles the final hardware architecture by inserting wrappers between functioning heterogenous components. In order to efficiently map system functionalities in a hw/sw co-design process, authors in [Sciuto et al., 2002] propose a methodology for defining *co-design analysis metrics* with the goal to statically associate to each task an affinity to be executed on a given processing element like a microprocessor, FPGA, DSP and alike. However, as authors report themselves, there could be no panacea refering to a single best-performing co-design methodology for all domains. In [Vincke et al., 2012b] a platform-based approach is used to develop a low-cost visual EKF SLAM application on a multi-core platform with a DSP. As authors conclude, a further step forward is implementing their work on a heterogenous architecture with an FPGA. Model-driven design flow is a long studied problem and proved worthy in a number of applicative domains where application complexity and computing architecture complexity may result in a delicate design flow. Some approaches use the data-flow model of application to ease the parallelism exploration to target heterogeneous multi-core architectures - [Bezati et al., 2014]. Others use a high level programming language that implements the data-flow model to be automatically transformed into a valid hardware implementation - [Sérot et al., 2014].

In our terms, applying hw/sw co-design methodology understands taking a defined system-level behavioral algorithmic model (it can also be a standalone software application) and transforming it to an application-specific integrated circuit (ASIC)-compliant design. The reason behind targeting a single-chip based hardware architecture is in possibility of achieving optimal performance/energy consumption trade-off given contempo-

rary technological traits. We apply a more general top-down methodology for hw[3]/sw co-design of visual SLAM (based on filtering techniques, such as EKF) on a heterogenous architecture - Fig. 2.4 - referred to in [Shaout et al., 2009]. Our approach is as follows[4]:

I find and accomodate the application to the global co-design analysis metric

II given system cost measure, identify bottlenecks

III repartition the design, build HW modules, integrate them back into design

IV if global cost measure is not achieved reiterate from step II, else stop.

Based on the level of parallelism (defined at the conceptual level) that the model/application may exhibit and in order to efficiently explore the design space, at *Modelling* phase our application is modelled in the form of a data-flow. Constraints such as power consumtion, memory footprint, speed, accuracy, time-to-market are set. They will influence HW/SW partitioning and condition whether the prototype is ready to be deployed for engineering a consumer-grade product based on its functional IPs and application's adapted execution flow. In *HW specifications* phase model-driven optimization is performed. The validated degree of parallelism is set as the static co-design analysis metric along with a heterogeneous development board on which we would be able to exerce that degree of parallelism. Then, tasks are being mapped to parallel executors and specifications are updated on them. Going through *CO-synthesis* phase, design gets incrementally partitioned [Balboni et al., 1996] into software and hardware modules with the accent on increasing number of hardware modules. During this phase, *Hardware in the loop* process is used to assure intellectual properties (IPs) compatibility with the rest of the system. In *Prototyping* phase multiple functional prototypes are dispatched until a final target-compliant prototype is issued.

In comparison with general co-design approaches, target hardware platform is chosen once in the *Update Specifications* block (i.e. the loop from *prototype N* never goes all the way back to *modelling* on Fig. 2.4). Given the maturity of the applicative domain i.e. taking into consideration all the previous scientific and engineering work in the community, it is argued that a well-chosen heterogenous development board porting an embedded processor with an FPGA device should suffice for small number of design loops.

For obstacle detection, according to Figure 2.4, this thesis will cover steps from *Application modeling* to *Allocation*. A model-based learning framework will be proposed with a case-study on *vehicle* obstacle class. For localization, the whole co-design methodology will be applied, starting with a working software code of SLAM.

[3]hw stands for FPGA

[4]In reference to Fig. 2.4, I: Modelling, Hardware Specifications until HW/SW Partitioning; II: Constraints verification in HW/SW Partitioning, III (we call it iterative prototyping): all blocks from HW/SW Partitioning to Constraints verification in Prototyping; IV: Target-Compliant Prototype

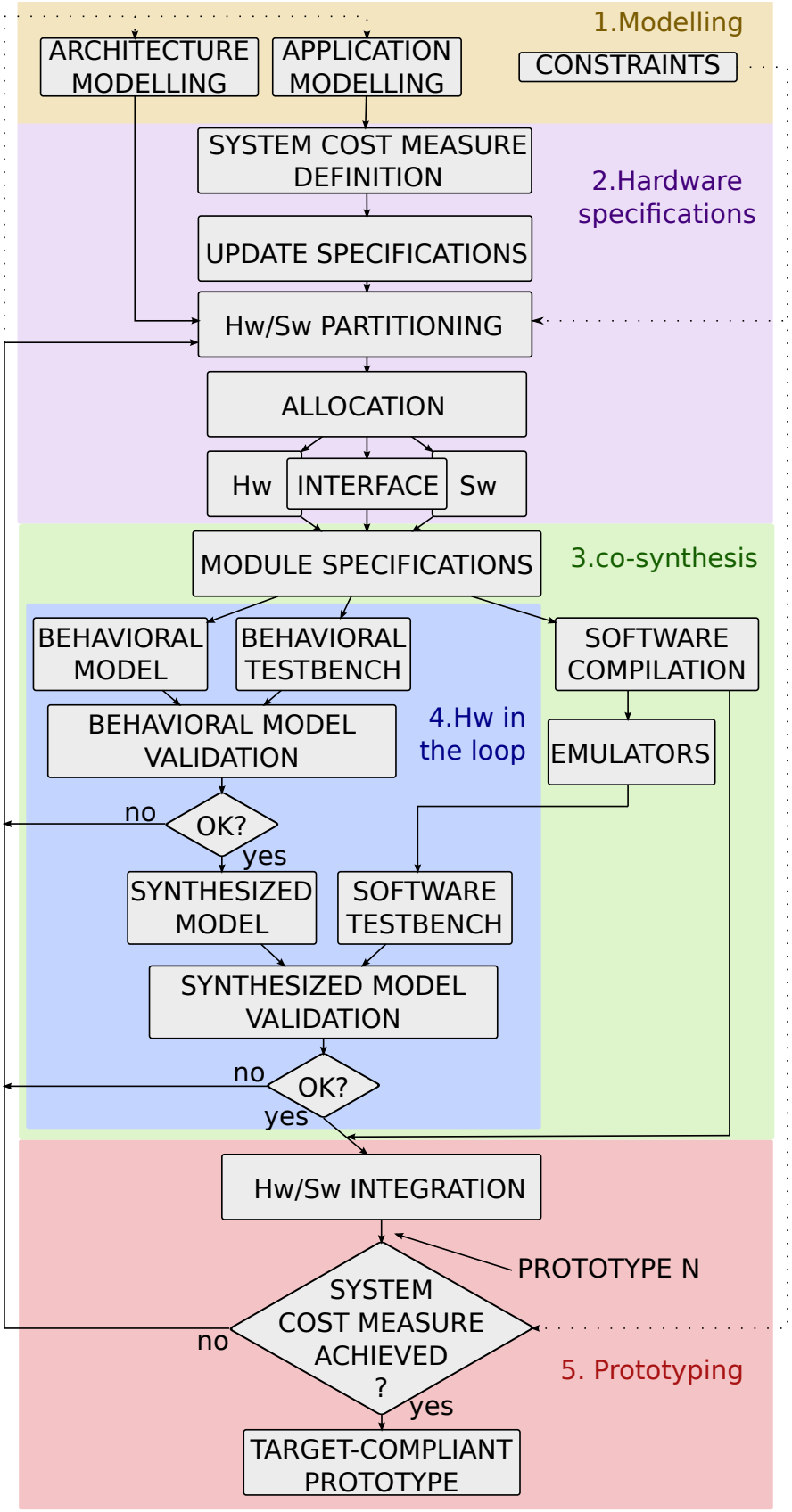

Figure 2.4: HW/SW co-design methodology for SoPC prototyping

2.3 Conclusion

After having described the context and constraints of the research in more detail in this chapter, its contributions relative to localization and obstacle detection tasks mentioned in previous chapter may be further refined in more detail as:

- Development of an FPGA-based scalable EKF block accelerator for Filtering-based visual SLAM applications. The accelerator's behavior is validated in *Hw in tho loop* process, as a bare-metal component in standalone application using Xilinx Software Development Kit and
- Single-scale obstacle detection framework proposition for embedded systems. In the executional pipeline (detailed in Chapter 6), most time-demanding tasks that should be executed on different physical threads are identified.

Using the hw/sw co-design methodology and by targetting embedded platforms, localization and obstacle detection tasks within a navigational framework are substantially accelerated and the power consumption during their execution is significantly lowered. As a result, this thesis offers efficient and real-time capable hardware and software architectures for mobile robots enabling them with evermore autonomy compared to equivalent state-of-the-art solutions.

Chapter 3

State-of-the-art

Contents

3.1 Computer vision for object classification 29

3.1.1 Interest Point Detectors and Descriptors 29

3.1.2 Explicit Shape 32

3.1.3 Without explicit shape 34

3.1.4 Machine learning 35

3.2 Complete real-time video analytics systems 39

3.2.1 Non-embedded solutions 39

3.2.2 Embedded solutions 40

3.3 Visual EKF-SLAM 41

3.3.1 Monocular SLAM implementations 42

3.3.2 FPGA-based SLAM implementations 42

As the up-to-date Computer Vision field covers quite a broad spectrum of vision-based algorithms, geometry and image processing techniques, the present overview is incented to give both critical and characteristic descriptions of techniques that are widely used within computer vision community for 2D object recognition. Some of them are adopted in this research for hypotheses generation. Built upon these algorithms are video analytic systems, presented in the second section. The third part describes contemporary visual EKF-SLAM algorithm implementations, focusing on performance and power metrics.

3.1 Computer vision for object classification

The task of detecting an object is equivalent to extracting a region of interest (ROI) in an image and assigning a label to it. This ROI is a bounding box represented by 2D coordinates which is slided over the image. Some works reduce the complexity of the detection window sliding task from $\mathcal{O}(n^2m^2)$ to $\mathcal{O}(nm)$ [Lampert et al., 2008]. In order to confront scaling issues, either parts of the objects or the objects in the whole are searched for in these image patches. Either than working on the raw group of pixels, a local descriptor algorithm is used to describe the central pixel in relation to its neighbours. With the information issued from descriptors we feed a trained classifier that labelizes the ROI with a related classification-quality score.
As my thesis focuses on monocular obstacle detection, all references made on 3D models in this chapter assume 2D feature extraction in the background.

3.1.1 Interest Point Detectors and Descriptors

These descriptors search for information (local features) within an image patch. Information is provided by a *detector* which is an image processing algorithm performing some morphological and/or convolutional operation. The detected, consistent points are called *salient*. These points are searched for by a Harris corners detector [Harris and Stephens, 1988] (which is scale-variant), Hessian [Bay et al., 2006] (scale invariant) and Difference of Gaussians (DoG) [Lowe, 1999] (which is fast but less accurate). Interest point descriptors are compaired in [Mikolajczyk and Schmid, 2005a]. Here, the authors conclude that the quality of descriptors is independent of detectors.

3.1.1.1 Raw pixel patches

This is a grouping of neighboring image pixels values. In [Agarwal et al., 2004], authors use these groups as *raw visual words* in order to generate a patch-based visual alphabet model for object classification. The distance measure between words is defined by their cross correlation. However, the correlation function is quite sensitive to the size and brightness in the image, which encourages other more invariant feature transformations that can cope with changes in real-life conditions.

3.1.1.2 Scale-invariant feature transform

SIFT descriptor [Lowe, 1999] computes a histogram of local oriented gradients around the interest point and stores the bins in a 128-dimensional vector (8 orientation bins for each of the 4×4 regions in a 16×16 pixels neighborhood centered at the interest point). The

local features generated are invariant to image scaling, translation, and rotation and are partially invariant to changes in brightness and affine transform changes. Conceptually, the appearance of salient points in the image should remain salient, even if the image is resized, rotated, or the brightness levels are changed - Figure 3.1. SIFT features may be used also for point-to-point matching of the same object between different image frames in the context of tracking.

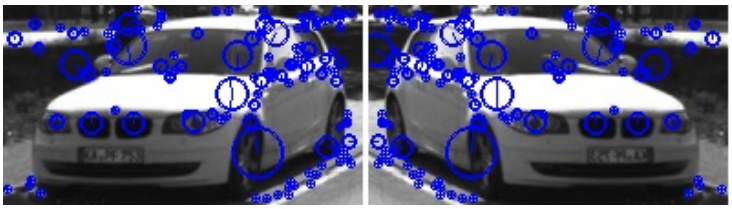

Figure 3.1: SIFT keypoints: left image is mirrored and SIFT is applied on both images. The extracted keypoints's locations are invariant to image rotations.

As reported in [Mikolajczyk and Schmid, 2005a], SIFT-based descriptors perform in general best among moment invariants [Mindru et al., 2004], differential invariants [Florack et al., 1994], shape context [Belongie et al., 2002], steerable filters [Freeman and Adelson, 1991], spin images [Lazebnik et al., 2003] and complex feature [Schaffalitzky and Zisserman, 2002] descriptors whereas moment invariants and steerable filters show the best performance among the low dimensional descriptors.
FPGA-based SIFT architecture is proposed in [Chang et al., 2013]. The architecture presented in this paper is able to detect interest points in a 320×240 resolution image in 11ms, which represents a speedup of 250× with respect to a software implementation.
Several SIFT variants can be found in literature:

- Authors in [Ke and Sukthankar, 2004] use Principal Component Analysis on the gradient image (**PCA–SIFT**). Descriptor returns a 36-dimensional vector which is fast for matching, but less discriminative than SIFT as reported in [Mikolajczyk and Schmid, 2005a]. Also it is more computationally demanding which reduces the speed-up effect of faster matching.

- *Mikolajczyk et al.*, also in [Mikolajczyk and Schmid, 2005a], propose Gradient Location and Orientation Histogram (**GLOH**). A larger feature vector with finer granularity than SIFT is extracted, and the dimensionality is reduced using PCA based on a large training set. The 128 strongest eigenvectors are used for description. However, the drawback when using dimensionality reduction techniques is always slower execution time to on-line classification tasks due to computational overkill.

- Modified SIFT descriptors with global edge features (extracted with a Canny detector [Canny, 1986]) are used in [Ma and Grimson, 2005] on vehicle images for intra-class classification. This type of classification differentiates between individual object instances in a given class i.e. between vehicle models in vehicle class. An evaluation on the test set from a mid-range surveillance camera shows small error rates, from 2% to 6%.

3.1.1.3 Speeded-up robust features

SURF descriptors are introduced in [Bay et al., 2006] in order to speed-up SIFT-like feature generation. The detector is based on the Hessian matrix [Lindeberg, 1998] on integral images. The descriptor, on the other hand, build the feature vector on a distribution of Haar-wavelet responses within the interest point neighborhood (on integral image). As its design focuses on computational speed, only 64 dimensions are used, which increases robustness but decreases distinctivity at the same time.
Recenlty, a Zynq-device based parallel architecture was implemented for SURF [Wilson et al., 2014]. It operates at 131fps with a 640×480 (VGA) resolution camera.

3.1.1.4 Histogram of Gradients

The concept of HoG was introduced in [Dalal and Triggs, 2005] for pedestrian detection. Descriptor uses a global feature to describe a large ROI that could contain a person rather than a collection of local features. Here, an entire person is represented by a single feature vector. To compute it, the input gradient image is divided into overlapping blocks (usually with a 50% rate of overlap) with each containing non-overlapping cells in itself. A histogram over given binning is calculated in each cell. Each block histogram represents a concatenated value from cell histograms. Final feature vector, representing the input gradient image, is obtained after having applied a contrast normalization scheme on each block and concatenated all the results. As gradient magnitudes vary over a wide range because of local variations in brightness and foreground-background contrast, so an eective local contrast normalization is essential for good performance. As a gradient-image based descriptor, HoG captures well the structure of the objects and is used most often in conjunction with Support Vector Machine (SVM) classifiers.

In [Negri et al., 2007] authors compare Haar (all-time famous face detector) to HoG features in vehicle detection. They conclude that HoG outperforms the Haar cascade and they make an interesting observation that when increasing the HoG feature space the detection rate increases while the number of false positives fluctuates at a stable level contrary to Haar where (by increase in Haar feature space) the number of false positives decrease with decrease in detection rate. The training contains multi-stage levels (11 and 12) as in a Viola Jones boosting algorithm [Viola and Jones, 2001].

A fused HoG-HCT (Histograms of Census Transform) feature set is created in [Li et al., 2012]. First, HoG and HCT features are extracted from different vehicle viewpoints and then fused together in a single block of matrix elements by PCA (Principal Component Analysis) algorithm. Afterwards they are introduced and built into a deformable parts model (front view, rear view, side view). Astonishing results based on "car" object class images from VOC 2007 show that this aproach outperforms HoG. However, they only present results on images of cars on roadways (not in urban environments). The quality of detector is always tied to specificity of the training set (data set).

One interesting study aimed to train HoG-based classifiers specific to vehicle orientation in comparison to a coarse (default) HoG-based classifier [Rybski et al., 2010]. The goal was to detect vehicle orientation and with it to further infer some relevant information. The conclusion was that, counter-intuitivelly, the coarse classifier outperformed the specific ones. There is also a study that unsuccessfully tried to modify the HoG descriptor to separate between edges (edge gradients) and shades ("diuse" gradients) because this descriptor at its current implementation cannot feed classifiers (such as SVMs) with in-

formation that is insensitive to texture [Doersch and Efros,]. Also, HoG exhibits aliasing problems where two image patches that are perceptually very different may end up with very similar HoG descriptors.

As a gradient-image based descriptor, HoG captures well the structure of the objects and is used most often in conjunction with Support Vector Machine (SVM) classifiers. Cell dimension, number of cells within a block, block overlap rate and number of blocks are all empirical values. In this thesis, a study over the optimal HoG parametrization for vehicle classification is described [Tertei, 2013].
A real-time version of this descriptor on Full HD images (1920×1080) is proposed in [Hahnle et al., 2013].

3.1.1.5 Other Descriptors

Boundary fragment model (**BFM**) for generic object recognition is introduced in the seminal paper [Opelt et al., 2006b]. Canny edge detector is used to generate object boundaries - fragmented lines. In the phase of model training, Chamfer distance measure [Borgefors, 1988] is used to generate a codebook of fragments. Each fragment has a geometric information to vote for an object centroid (in the manner of Leibe et al. [B. Leibe and Schiele, 2004]). Discriminative fragments are learned afterwards via the process of boosting. In general, part-based models are advantageous in dealing with partial object occlusions.
Edgelet features [Wu and Nevatia, 2007]. These local shape features represent short segment of a line or a curve. Extracted after a Sobel convolution, they are used for pedestrian detection. A strong edgelet-based classifier is learned after boosting. Authors in the aforementioned paper combine people detection and tracking (initialization and hypotheses on trajectories) which in that way shows improved robustness for both techniques.
Apart from aforementioned descriptors (SIFT, SURF and HoG) raw pixels may be used for binary edge extraction. In that way the information may be used as normalized input for feature descriptors, as varying brightness levels in pixel intensities are mostly removed during edge detection. Canny edge detector to generate features is used for:

- Traffic flow analysis by tracking vehicles on highway [Kim and Malik, 2003]. Edges are detected for generating lines of a 3D vehicle model. Then, connected-component analysis is applied for line grouping.
- Intra-class vehicle detection [Ma and Grimson, 2005] as described earlier in Subsection 3.1.1.2.
- Generic object recognition in [Opelt, 2006], which will be more detailed in Subsection 3.1.4.

3.1.2 Explicit Shape

Here it is presumed that parts of objects are detected. This Subsection focuses on direct modeling of the spatial relationship between them.

3.1.2.1 K-fans

The N parts of an object are modelled as nodes of a graph. K-fan model [Crandall et al., 2005] separates this nodes into k reference nodes and N-k regular nodes.

Each reference node is *N-regular*, having edges towards all other nodes. The spatial prior is thus conceptualised in this manner as number of *N-regular* reference nodes within a graph. Changing the value of k, the graph goes from null to complete, changing the spatial representation of an object from *no structure* to *full rigid structure*. This concept is introduced in order to improve understanding of tradeoffs between representational power and inherent computational complexity for part-based recognition. Authors conclude that for efficient object detection and localization (in this sense *k-fans* model combines the two) very often we don't have to use higher order fans.
Most shape models are related to *k-fans*:

- Authors in [Kim and Malik, 2003] use a 1-fan model to group edges of highway scenes into vehicles.
- A 1-fan constellation for vehicle detection based on SIFT features is used in [Ma and Grimson, 2005].
- Also, image gradient-based descriptors such as HOG and 3DHOG use a complete graph model.

3.1.2.2 Implicit-shape model (ISM)

This 1-fan model is introduced in [B. Leibe and Schiele, 2004] and in more detail in [Leibe et al., 2008a]. During training process, 25×25 pixel patches are extracted from images using Harris corner detector. After agglomerative clustering based on a defined similarity criterion, cluster centers are stored in a codebook. In the second training iteration, all codebook entries are assigned a probability relative to their location in regard to object centroid. That, in fact, defines the ISM model. During prediction, unknown image patches are matched against all codebook entries and each element gives a vote toward object centroid based on its spatial occurrence distribution, in a generalized Hough voting [Ballard, 1981] manner. Interesting and useful part of this procedure is what comes next: once the object centroid is found (mean-shift mode estimation [Cheng, 1995]), the most contributing codebook entries are backprojected with their surrounding in original image, giving us a segmented object.

Generalized Hough Voting. Similar approach for object voting is used in Lowe's seminal paper [Lowe, 1999], which has been already described in Subsection 3.1.1.2. Same method, extended using different features and distance metrics is reused in:

- [Leibe et al., 2007] and [Leibe et al., 2008b]; features, averaged pixel intensities matched in consecutive camera frames are fed into a Structure from Motion (SfM [Hartley and Zisserman, 2000]) pipeline. Cars and pedestrians are detected in an urban area from a moving camera.
- [Cornelis et al., 2008]; corners and edges are used for features,
- [Leibe et al., 2005]; with SIFT features fed into ISM codebook,
- [Opelt et al., 2006b]; already described in Subsection 3.1.1.5,

- Opelt et al. [Opelt et al., 2006c] and [Opelt et al., 2006a] where authors make use of the BFM model and a plethora of descriptors (SIFT, Harris corners, moment-invariant descriptors) - respectively, with AdaBoost.

3.1.2.3 Alphabets of visual words

Alphabets are used for reducing the number of training samples. Some clustering technique is used to identify similar feature vectors and to combine them. Visual vocabulary is then built upon the set of cluster centers. In this kind of codebook the spatial information is embedded in cluster centers as each holds a list of class labels and could represent several positions in a shape model - Figure 3.2. This idea is used in [Wijnhoven et al., 2008],[Creusen et al., 2009], [Ma and Grimson, 2005], [Opelt, 2006] and [Wijnhoven and de With, 2009].

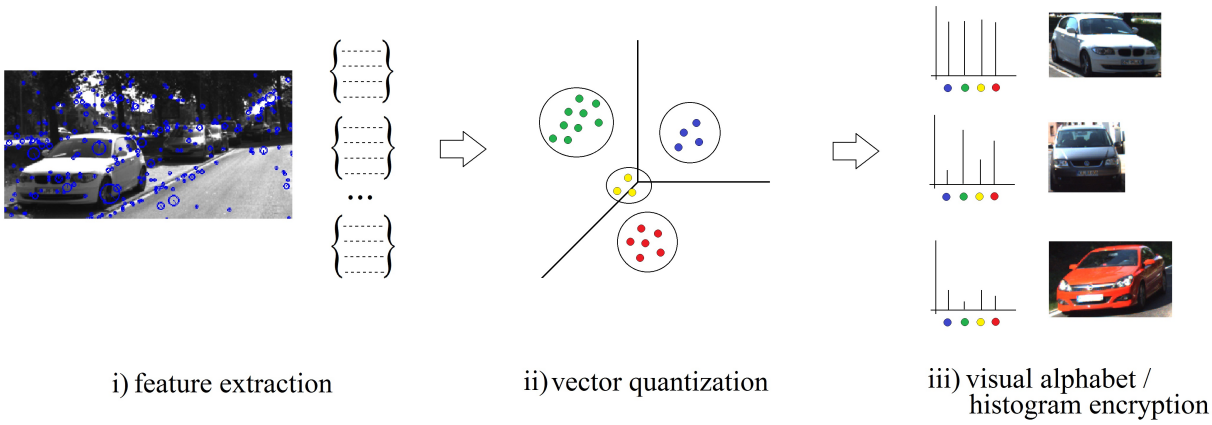

Figure 3.2: An example of alphabet encoding with 4 cluster centers for intra-class classification: i) interest points are extracted and described with a decriptor in form of encoded feature vectors; ii) each of the feature vectors is assigned to closest visual word from codebook based on a distance measure; iii) histogram containing this combination of visual words is passed to a classifier which determines object instance.

3.1.3 Without explicit shape

Also known in literature as *Bag of Words* (BoW [Sivic et al., 2005], [Fei-Fei and Perona, 2005], [Jurie and Triggs, 2005], [Nowak et al., 2006]). This 0-fan "shapeless" model is extensively used for generic object recognition. A set of extracted visual features is grouped into empirically chosen number of cluster centers (a visual dictionary). Each training instance (feature vector) here is represented as a histogram containing the number of appearances for each feature in a visual dictionary. Interesting comparison between alphabets and BoW as object categorization techniques is given in [Wijnhoven and de With, 2009]. Authors argue that when it comes to feature vector representation, the histogram-based BoW compares better to HMAX-based alphabet (also known in the literature as extended *standard model* [Serre et al., 2007]) for small dictionaries with up to 500–1000 features, making it more suitable for embedded applications. Further they conclude:

- *Dictionary quality.* Random BoW dictionaries outperform dictionaries created from Hessian-Laplacian salient points.
- *Interest point selection.* Hessian-Laplace point operators fail to match up against processing all image points.

3.1.3.1 Hierarchical object recognition

Hierarchical object recognition is mainly inspired by biological vision, more precisely on the structure of the human visual cortex [Ullman, 2007]. Inherent advantage of such a hierarchy lies in invariance to position and scale in lower layers and invariance in viewpoint angle and other transformations in hiher layers of the structure. Here object features are based on image patches. Every image patch is decomposed into smaller informative subparts. The procedure is repeatedly carried out until we obtain atomic patches. This leaves us at the end with a tree-style object hierarchy.

The *standard model* [Serre et al., 2007] presents a complete object recognition and segmentation framework using a visual cortex structure. Four layers are used, which perform:

- *Simple filtering.* Simple S_1 cells apply a battery of Gabor filters ([Gabor, 1946] and [Daugman, 1985]) over input image patches at multiple scales, and at each scale in $0°, 45°, 90°$ and $135°$) orientations,
- *Complex searching.* Cortical *C1* cells search for local maxima over scale and position.
- *Filtering.* *S2* units behave as Radial Basis Function (RBF) units. Each *S2* unit's response depends in a Gaussian-like way on the Euclidean distance between a new input and a stored prototype. Patch prototypes are learned during previous training. Response is kept for each orientation.
- *Likelihood response.* *C2* units compute a global maximum over *S2* units. That is thus a shift- and scale-invariant response. Final result is a vector of N *C2* values, where N corresponds to the number of prototypes.

This feature vector mat further be passed to a linear classifier, SVM or boost. This approach is used in traffic surveillance applications in [Wijnhoven et al., 2008], [Creusen et al., 2009], [Wijnhoven and de With, 2009] and [Wijnhoven and de With, 2007].

3.1.4 Machine learning for classification

A classification algorithm is defined as a mathematical function that maps input data to a category. If the categories labels are provided a priori, the *classifier* is said to be supervised, otherwise it is unsupervised[1]. Authors in [Bradski and Kaehler, 2008] separate classifiers as generative and discriminative. Simplest way of interpreting is that generative approach models the cause of data (likelihood) while the discriminative models what data cause (probability). In Table 3.1 an overview over popular discriminative and generative classifiers used in computer vision may be seen.

[1] in some cases semi-supervised i.e. lack of some input data and/or mislabeled data

Table 3.1: Comparison of generative and discriminative classfiers

Algorithm	Model type	Feature representation	Training type	Advantages	Disadvantages
K-means	Generative	Cluster regions over defined distance metric. Number of clusters is pre-defined by user/expert.	Unsupervised	Resulting regions can be used as sparse histogram bins to represent data. Good expertise may define the best K-number or *elbow* method. Guaranteed convergence.	Cluster regions are non-Gaussian, their formation is dependent on distance metric and adjusted weighting.
Naïve Bayes Classifier	Generative	Gaussian distributed and statistically independent from each other.	Supervised	Can handle multiple classes in contrast to other binary classifiers. Works very well on small training data sets.	In general, features are not statistically independent (i.e. presence of eye feature implies face feature)
Decision trees	Discriminative	Features are thresholds at tree nodes used to train the tree (left-right branch separation).	All training type is supported.	Very fast.	Additional variable performance measure techniques need to be used when having large false positive rates.
Boosting (group of K classifiers)	Discriminative	Each classifier uses a simple decision tree with thresholds as features in training.	All training type is supported.	Effective on large training datasets.	Tends to generate large classifiers.
Neural networks	Discriminative	Raw input data are transformed in hidden layers as features; synapses represent the weighted mapping of inputs.	Supervised	High performance. Can map any higher order polynomial equations. Incredibly fast if used offline for predictions.	Size of training set is a tradeoff to time needed to perform online learning – dependent on processing power of underlying hardware. Training does not always converge.
Support vector machines	Discriminative	Raw input data are transformed by kernel functions – aka similarity measures. That is a mapping in respect to already known landmarks (supervision points in data space).	Supervised	Supports very large training datasets. Has high performance and the solution is found (based on trained model) in polynomial time.	Has to have a balanced ratio of positive and negative input data (or balanced ratio of class representatives in multi-class training).

3.1.4.1 Boosting

Boosting is a method of constructing a robust classifier on the basis of a linear combination of simpler, possibly biased, classifiers. Discrete AdaBoost, a binary category-based boosting algorithm by Freund [Freund, 1995] [Freund and Schapire, 1999], uses a combination of T classifiers called hypotheses h_t in order to make weighted predictions upon samples x:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right) \tag{3.1}$$

Hypotheses are linear predictors, so called "weak" classifiers such as decision trees and the perceptron learning rule. They are chosen from a domen $\mathcal{H}$, which is given beforehand the training process as a finite set of predictors. During training, the user provides number of weak learners T. The training process proceeds iteratively from t = 1:T with a chosen hypothesis in each iteration which satisfies the equation

$$h_t = arg\min_{h_j \in \mathcal{H}} \epsilon_j = \sum_{i=1}^{m} D_t(i)\left[y_i \neq h_j(x_i)\right] \tag{3.2}$$

where the error ϵ is upper bounded as $\epsilon_j \leq 0.5$ meaning that h_t only needs to perform better than random. m is the number of training samples. In this way, AdaBoost builds sequentially a set of "better than random" classifiers. D_t is a discrete distribution function which starts as a uniform distribution in first iteration ($D_1(i) = 1/m$) and changes as:

$$D_{t+1}(i) = \frac{D_t(i)e^{-\alpha_t y_i h_t(x_i)}}{Z_t} \tag{3.3}$$

Here, newly chosen hypotheses compensate for misclassified samples from previous iterations as the exponential component assures increase (decrease) in weight of wrongly (correctly) classified examples:

$$e^{-\alpha_t y_i h_t(x_i)} = \begin{cases} < 1, \ y_i = h_t(x_i) \\ > 1, \ y_i \neq h_t(x_i) \end{cases} \tag{3.4}$$

$Z_t(\alpha)$ is a convex, differentiable and monotone function (in terms od a discrete function through iterations) which is set as:

$$Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)} \tag{3.5}$$

$$\epsilon_t = \frac{1}{2} \mid \{H(x_i) \neq y_i\} \mid \tag{3.6}$$

In an iteration $t > 1$, Z_t acts as a normalization factor so that D_{t+1} is a distribution. Finally, α_t is set to greedily minize Z_t:

$$\alpha_t = \frac{1}{2} log\left(\frac{1 + \sum_{i=1}^{m} D_t(i)h_t(x_i)y_i}{1 - \sum_{i=1}^{m} D_t(i)h_t(x_i)y_i}\right) \tag{3.7}$$

Authors in [Jones and Snow, 2008] use AdaBoost for pedestrian detection in road surveillance. Generic object recognition with a binary multilayer AdaBoost network is performed in [Zhang et al., 2005]. In [Opelt et al., 2006b] and [Opelt et al., 2006a], binary

AdaBoost is used for generic object recognition, where boosting automatically performs the feature selection. An extension to multiple classes and incremental learning is introduced in [Opelt, 2006] and [Opelt et al., 2006c]. Boosting of gradient features to detect vehicles in road scenes is used in [Khammari et al., 2005], and [Acunzo et al., 2007]uses a boosted classifier for illumination condition detection (e.g., day and night).

3.1.4.2 Support vector machines

Generative model, in order to infer well the probability of occurence of an object has to be provided with all (or close to all) instances of the object which may appear during prediction. However, given our applicative domain, training a generative model is hardly possible and that is the reason why a discriminative model such as SVM is chosen.

In order to model well the features of a given class versus all other classes (1 vs all classification), an SVM relies on calculating a separating hyperplane given by θ, which delimits n-dimensional space of m samples[2]. Thus during training, it has for its goal thus to minimize the objective function given as:

$$\min_{\theta} C \sum_{i=1}^{m} \left[y^i cost_1(\theta^T K(q, x^i)) + (1 - y^i) cost_0(\theta^T K(q, x^i)) \right] + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2 \tag{3.8}$$

Here C is the regularization parameter which influences the separating margin quality (bigger it is, bigger is the margin). Cost functions guarantee following relations:

$$cost_0 : \theta^T K(q, x) \leq -1 \tag{3.9}$$

$$cost_1 : \theta^T K(q, x) \geq 1 \tag{3.10}$$

and $K(q, x)$ stands for kernel trick function. In case of linear kernels $K(q, x) = x$. In this case the positive and negative examples in dataset are linearly separable. However, on more complex datasets a similarity function transformation known as *kerneltrick* has to be performed (i.e. an RBF like Gaussian is most often used - Figure 3.3[3]). SVM parameters will be described in more detail in subsection 6.2.3.

Although in literature AdaBoost and SVM classifiers tend to perform similarly, in this thesis linear SVMs are chosen for detection-by-identification tasks. Firstly, AdaBoost's decision making architecture breaks down to decision tree stumps, which can be efficiently implemented on a reconfigurable hardware by making use of LUTs. Main problem being here is that for complex datasets, such as obstacle detection in urban environments, the number of stumps is quite large and that leads to huge resource consumption of these configurable slices, which are majorily used for accelerator's logic implementation such as state machine. On the other hand linear SVMs's decision making is a dot-product matrix operation whose implementation in both embedded software and hardware is thoroughly studied in literature.

3.1.4.3 Application in computer vision

Classification techniques used in computer vision may be categorized as[4]:

[2]each sample x_i contains its label y_i

[3]image courtesy of Czech Technical University, Prague

[4]text in italics denotes I/O data

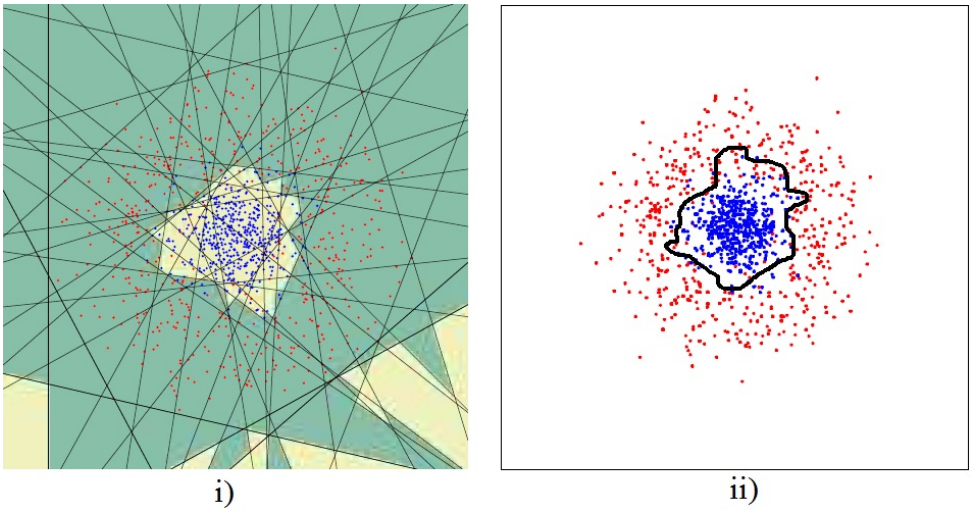

Figure 3.3: i) AdaBoost's linear kernel vs ii) SVM's Gaussian kernel: an example over a complex, linearly non-separable dataset.

- Top-down classification
  Frame → Foreground estimation → *Mask* → Features grouping → *Silhouette list* → Classifier → *Object list*

- Bottom-up classification *Frame* → Patch detector (descriptor) → *Patch list* → Patch classifier → *identified object* → Grouping (Hough voting) → *Object list*

In this thesis we will focus our efforts on bottom-up approach.

3.2 Complete real-time video analytics systems

3.2.1 Non-embedded solutions

3D convex hull matching. An RT system called *SCOCA* is introduced in [Messelodi et al., 2005] to track and classify 8 vehicle classes at road intersections. 3D conveh hull models are used to initialize an object list for every fifth frame based on the convex hull overlap of model projection and motion map. Camera calibration is required for this operation. A feature tracker follows the detected objects along some frames before a new initialization takes place. The tracker is used to speed up the operation, because the 3-D operation would not be fast enough to operate on every frame in real-time. The objects are classified into eight classes based on a two-stage classifier. The first stage evaluates the convex hull, whereas the second layer uses pixel appearance (color) for classes with similar convex hull. The performance is evaluated on 45 min of video data from two different sites. The total classification rate is given with 91.5% for the test data of the authors.

3D model matching against GMM silhouettes. 89.8% classification accuracy on 1 hour video. The work in [Buch et al., 2008], [Buch et al., 2010], and [Buch et al., 2009c] builds on this approach and uses a GMM to compare motion silhouettes with 3-D models for the classification of five types of road users. An appearance model of 3-DHOG is

introduced in [Buch et al., 2009a] and [Buch et al., 2009b]. This approach integrates local patch features with 3-D models.

Perspective normalised frame difference. 94.7% accuracy on 24 hour video to detect parking events. A system for detecting parked vehicles is introduced in [Park et al., 2007]. Camera homography is used to generate a normalized ground plane view. Based on a frame-differencing motion map, parking-in and parking-out conditions are calculated. A state machine is used to track the speed changes of vehicles until stopping to generate these conditions. The system is evaluated with 24 h of video data from two different sites. A detection rate of 94.7

3D model matching against motion mask from multiple camera with calibration [Atev et al., 2005] and [Atev and Papanikolopoulos, 2008]. 85% of 273 vehicles succesfully detected at 35fps with road coordinates. Multiple cameras, calibrated according to [Masoud and Papanikolopoulos, 2004] with road primitives, are used to identify 3-D ground plane locations of vehicles by projecting all foreground masks to the road plane.

3.2.2 Embedded solutions

Embedded solutions to road obstacles detection are found mainly in automotive industry, under *Collision Avoidance* projects of leading car manufacturers as Audi, BMW, VW, Volvo, ... However, in order to better their performance, these hazzard avoidance systems integrate multiple sensors along with a camera: radar, laser, ... In this industry, MobilEye [MobilEye, 2009] has developed its leading camera-based only obstacle avoidance solution.

MobilEye's **SeeQ2™** is a stand-alone camera and vision processing module for automotive applications. It offers an embedded processing platform for computationally intensive real-time visual recognition and scene interpretation applications. The SeeQ2™ board includes the MobilEye EyeQ2™, a powerful vision processor, a high dynamic range CMOS image sensor, Flash and SDRAM memories, CAN and GPIO interfaces for the peripherals. This vision system on a chip (VSoC) supports Lane Detection and Road Geometry estimation, Vehicle Detection, Intelligent Headlight Control, Traffic Sign Recognition and Pedestrian Detection. The Advance Warning System (AWS) is designed to alert the driver of potential collision situations and unintentional lane departure and increase awareness to dangerous situations during driving.

EyeQ2™ vision processor, manufactured by STMicroelectronics, processes up to two monochromatic images from 640x480 (VGA) CMOS image sensor at a rate of 60fps. It includes two floating point, hyper-thread (4 threads) 64bit RISC 34KMIPS CPUs running at 324MHz. And five ASICs called Vision Computing Engines (VCE):

- Classifier Engine
  - Image scaling and pre-processing units
  - Pattern classifier units
  - Tracker Engine
  - Image warping and motion analysis unit

- Pre-process Window
  - Image Convolver and image pyramid units
  - Computes vertical and horizontal edge maps
- Filter Engine
  - Features based classifier unit
- Disparity Finder Engine
  - Stereo engine
  - Programmable search, 2 pixel/clock
- Three Vector Microcode Processors (VMPs)
  - A generic Very Long Instruction Word (VLIW), Single Intruction Multiple Data (SIMD) vector processing unit without cache.

This smart-camera solution, operating at 5 VDC input voltage and dissipating only 3W, is encassed within a 65mm x 33mm x 10mm box weighing 20g (nominally). A performance evaluation is given in [Raphael et al., 2011]. Camera-only detection error was overall 9% and 11% for vehicles detected within 0-60 meter and 60-90 meter ranges, respectively.

3.3 Visual EKF-SLAM

As described in Subsection 1.1, methods that use Visual SLAM for localization may be classified according to several criteria. Here, only Filtering-based (and specifically EKF-based) methods are considered, like in [Zhang and Vela, 2015b, Bresson et al., 2014], because they are more adapted for embedded systems with limited resources. Several real-time optimization-based methods have been proposed, but either they are not robust [Klein and Murray, 2007] (indoor use, i.e. augmented reality), or they require a huge computing power [Mur-Artal et al., 2015a] or they are specific for drone applications [Forster et al., 2014]. Then according to features extracted from images, most methods exploit only interest points (Harris, SIFT, SURF, Speeded Up Robust Features (SURF), Binary Robust Independent Elementary Features (BRIEF), Oriented FAST and Rotated BRIEF (ORB) to name but a few); [Zhang and Vela, 2015a] compares several methods proposed for point detection and characterization, in this thesis only Harris points are used.

Finally according to sensor configuration, the SLAM function could acquire images from monocular, bi-cam or stereovision sensors. The latter allows to acquire depth images, but only in the sensor's vicinity e.g. ∼8m, for a 40cm stereo baseline. Three-dimensional (3D) landmarks are directly created as Euclidean points. Points further away from the sensor are never considered. Monocular or bearing-only SLAM uses a single camera, so that only 2D features can be observed. Such features cannot be added directly in a 3D-parameterized map. Either landmark initialization is delayed [Davison, 2003b], while waiting for other observations from different viewpoints so that

a 3D point could be triangulated, or it is undelayed using a specific parametrization [Solà et al., 2011, Civera et al., 2006]. The latter allows us to exploit landmarks further from the camera, e.g. on top of the horizon line. Bicam-SLAM combines both approaches, allowing landmarks to be created using features obtained from the two images, and from one camera only. In this thesis only monocular SLAM is used with undelayed landmark initialization , so that this method could be executed on an embedded system with limited size and resources - such as a smartphone.

3.3.1 Monocular SLAM implementations

RT-SLAM [Roussillon et al., 2011] is a generic Open Source software implementation of our state of the art visual bearing-only SLAM algorithm, based only on one camera to observe the environment, and on an IMU to estimate motions. The map contains Anchored Homogeneous Point (AHP) landmarks [Solà et al., 2011] which is a seven-dimensional landmark parametrization. AHP landmark parameterization is shown to increase filter's consistency, achieving more robust and accurate localization and mapping. A predefined number of landmarks from the map is observed and positions are corrected after each frame acquisition. The choice is made by applying the one-point Random Sample Consensus (RANSAC) [Fischler and Bolles, 1981]. Active search strategy is used for initialization of new ones. Authors [Roussillon et al., 2011] report real-time performance at $60Hz$ when having 20 consecutive landmark corrections and 5 initializations per EKF loop on an Intel i7 processor (only one core is used).

C-SLAM [Gonzalez et al., 2011] presents a C programming language version of RT-SLAM that implements a simplified SLAM application respecting the DO-178 norm required on airborn system. It uses a constant speed robot motion function and Inverse-Depth Point landmark parameterization [Montiel, 2006]. C-SLAM was implemented in [Botero et al., 2012], on an embedded architecture based on a Virtex5-Virtex6 FPGA couple [Xilinx, 2011]; it runs at 24Hz when having 20 IDP landmarks in the map and correcting 10 of them in each frame. The advantage gained by this architecture lies on the co-design possibility, with a hardware-level implementation of image processing functions at pixel frequency: Harris point extraction and active search strategy.

In a recent study [Vincke et al., 2012a], a visual EKF-SLAM algorithm is analysed in-depth and a multiprocessor-based embedded solution is proposed on Texas Instrument's SoC OMAP4430 platform. It contains dual core ARM Cortex-A9 integrating two SIMD NEON coprocessors, a GPU PowerVR, and 1GB DDR2 RAM. For peripherals an exteroceptive sensor (a low cost webcam) and an ATMegaI68 microcontroller which controls two odometers are connected. Frames have been grabbed at 30fps with 640×480 resolution. Odometers data were sampled at 30Hz. Map contains 50 IDP landmarks, all of which are observed and corrected. Number of initialized new landmarks in a loop is also set to 50. Their sw/hw co-design methodology is explained in great detail in [Vincke, 2012]. Reported mean processing frequency per frame acquired is 38Hz, which makes their embedded SLAM run in real time.

3.3.2 FPGA-based SLAM implementations

A power-efficient FPGA-based embedded EKF block accelerator is also proposed in [Bonato et al., 2008]. Their SLAM is a 2D EKF-SLAM application that satisfies real-

time constraints ($14Hz$ execution frequency in this case) with approximately 1.8k landmarks contained in their map which are represented in Euclidean parametrization (two-dimensional state size). All of them are observed and corrected. They do not report on initialization and matching techniques and neither mention whether they use sensors besides a camera.

A recent work on front-end feature extraction implementation for SLAM on an FPGA using bundle adjustment for localization may be found in [Nikolic et al., 2014]. Comparison with our work will be given in Subsection 4.4.1. To the best of our knowledge, however, there is no single FPGA implementation of both front-end and back-end parts of a visual SLAM application. Finally, some authors proposed FPGA implementations of matrix multiplication or inversion, using different strategies [Sarraf et al., 2007, Qasim et al., 2010, Tiwari et al., 2013]; the Parallel input/Multiple Output strategy proposed in this last paper, is very similar to our systolic array architecture.

Chapter 4

EKF-SLAM HW/SW co-design: part I

Contents

4.1 Modeling **45**

4.2 Hardware specification before partitioning **47**

4.2.1 System Cost Measure Definition 47

4.2.2 Update Specifications 49

4.3 Iterative prototyping **50**

4.3.1 First iteration: software-only prototype 51

4.3.2 Second iteration: Front-end hardware accelerator 52

4.3.3 Third iteration: Back-end hardware accelerator 56

4.3.4 Fourth iteration: Front-end hardware accelerator 58

4.4 Final hardware prototype **61**

4.4.1 Experimental results and discussion 62

4.4.2 Comparison with state of the art 63

4.5 Conclusion **64**

4.5.1 Towards EKF-SLAM at 100Hz 65

SLAM application used in this thesis is an upgraded version of C-SLAM. It is a monocular SLAM fusing IMU data for motion prediction, correcting 20 AHP interest points. Active search strategy is used as well to match the observed points and initialize new ones (5 per frame). This number is obtained by experimentation for one application on a robotic platform: C-SLAM behaviour was satisfactory when having set parameters in this manner. IMU data are used in order to reduce uncertainty on landmarks tracking during interest points matching. With this algorithmic optimization and for speed issues, one-point RANSAC is replaced by random choice when initializing new points in the map.

This chapter is organised as follows: application modelling along with design constraints is presented in Section 4.1 after which an explanation of the model optimization is given in Section 4.2. Afterwards, co-design iterations are described in Section 4.3. The final prototype is shown in Section 4.4 where we compare it with state of the art.

4.1 Modeling

The dataflow model of computation aims at describing applications regarding data dependencies. The application is represented as a network of actions in a direct acyclic graph (DAG) allowing to get knowledge of intrinsic parallelism and memory requirement. The dataflow paradigm was formally introduced by Kahn in [Kahn, 1974]. A Kahn process network (KPN) is made of actors connected by unbounded first-in first-out (FIFO) queues carrying data tokens - Fig. 4.1. A data token is an atomic (can not be divided) chunk of data. An actor can thus be described as a functional process that maps a set of tokens sequence into another set of tokens sequence. The dataflow process network (DPN) - [Lee and Parks, 1995] - inherits its formal underpinning from Kahn process networks but associates a set of firing rules to each actor to give the necessary tokens input for an actor to be triggered. This allows further analysis of the network using a set of properties. For further compile-time analysis, the synchronous dataflow model (SDF) - [Lee and Messerschmitt, 1987] - takes the dataflow process network semantic and restricts its expressiveness by specifying the data production/consumption rate in an integer form for each actor interconnection. This additional property allows to compose the topology matrix of the SDF graph T that can be analyzed to extract scheduling informations. This topology matrix allows to ensure the application to be deadlock free, to compute a statical schedule of the application and to compute memory requirements at compile time.

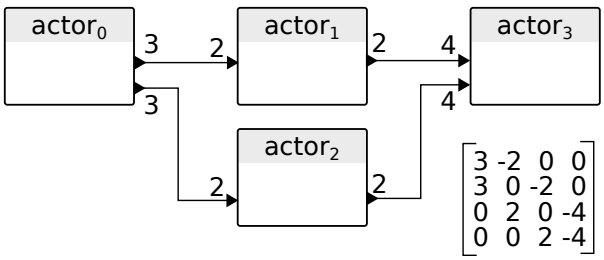

Figure 4.1: Example dataflow graph

The EKF-SLAM problem can be described as the following pseudo-code:

```
for n = 0; ; n++ do
    image = acquire_frame()
    predict(slam, robot_model);
    i = 0;
    j = 0;
    lmk_list = select_lmks();
    correl_list = correl_lmks(lmk_list, frame);
    for i = 0; i < nb_correl and j < nb_correct; i++ do
        if score(correl_list[i]) > correl_threshold then
            correct_slam(correl_list[i]);
            j++;
        end if
    end for
    feature_list = detect_features(frame);
    j = 0;
    for i = 0; i < grid_size and j < nb_init; i++ do
        if init_lmk(feature_list[i]) then
            j++;
        end if
    end for
end for
```

with collections:

- *lmk_list*, a list of landmarks (in the SLAM terminology)
- *correl_list*, a list of observations in the image for the SLAM landmarks
- *feature_list*, a list of detected features in the image

and functions :

- *predict*(), a function that performs the prediction step of the EKF filter
- *select_lmks*(), a function that returns a list of landmarks to be observed in the image frame (landmarks selection)
- *correl_lmks*(), a function that observes (matches) the selected landmarks in the image (landmarks correlation)
- *correct_slam*(), a function that performs the correction step of the EKF filter based on observations of landmarks
- *detect_features*(), a function that detects a set of features in the image (features detection)
- *init_lmk*(), a function that initializes detected image features as landmarks in the EKF-SLAM map (landmarks initialization)

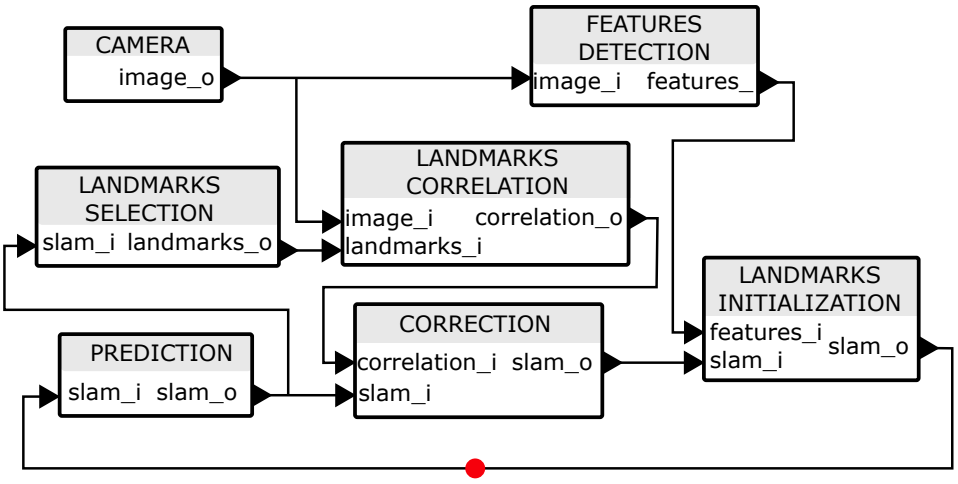

Figure 4.2: The vision-based EKF-SLAM SDF model

that translate into the following SDF representation (Figure 4.2). This SLAM algorithm is adapted to run in a SDF manner, explaining why the *correction* and *landmarks initialization* are executed in a worst-case for loop while the pseudo-code description would only execute these steps for a variable number of times.

This data-flow description was translated into a multi-threaded functionnal model of the application to validate the general behavior of the application.

Scheduling of these models clearly shows that the application exhibits a single degree of parallelism (features detection can be run in parallel with the other tasks) with minimum impact on the application latency. Therefore, running the application in real-time requires oversized computational resources and keeping it that way it is difficult to take advantage of heterogeneous architectures.

Constraints intended to be imposed through design loops are: (A) software execution time - it needs to be under $33.33ms$, (B) minimal power consumption and (C) minimal time-to-market.

4.2 Hardware specification before partitioning

This Section begins with our work concerning structural-level optimization of the application in order of exposing an additional degree of parallelism and ends with specifying the chosen hardware platform.

4.2.1 System Cost Measure Definition

In SDF applications, more parallelism can be exposed at the structural level by applying re-timing techniques. Re-timing consists in adding delay tokens on the graph edges in order to modify the application pipeline.

This modification does not affect the structural aspect of the model (way that nodes are connected) and the latency of the schedule as the repetition vector of the graph is preserved. Re-timing only allows to modify the schedule pipeline to take advantage of architectures with more degree of parallelism than what the application initially exposed.

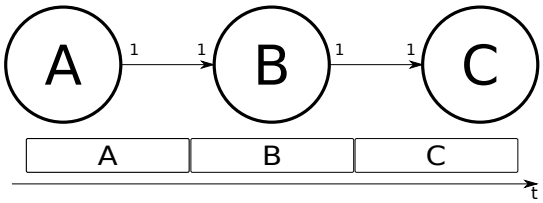

Figure 4.3: A simple dataflow graph and its one processor schedule

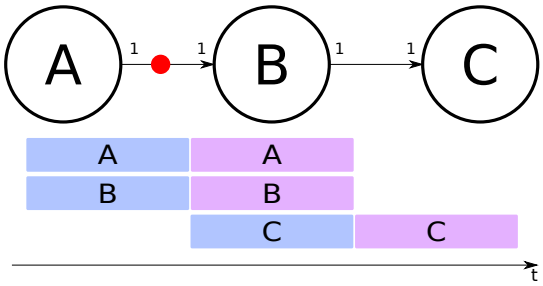

Figure 4.4: The simple dataflow graph with re-timing (delay token denoted by a red dot) and its multi processor schedule

In that way we optimize the application's throughput. Let's consider a simple application with only three nodes: Fig. 4.3.

The initial schedule of the model does not expose parallelism. This data-flow model can be altered by adding delay tokens (pictured with a dot on the arc in the graphical representation) to generate a schedule with a higher level of parallelism: Fig. 4.4. Adding a single token on the edge between A and B means that B can fire before A even produced a token, thus consuming the delay token.

This re-timing clearly exposes an application pipeline with more parallelism and allows to optimize the throughput of the application for a multi-processor architecture. Re-timing only affects the structural level of the application but still needs to be taken into account when implementing the behavior of the graph's actors.

In our 3D EKF SLAM application, re-timing was achieved by delaying the *prediction*, *correction* and *landmarks initialization* tasks (from Fig. 4.2 these tasks are delayed in Fig. 4.5).

After the re-timing, the application exposes three-degree parallelism as landmarks correlation and feature detection can be performed in parallel of the Kalman filtering steps. This allows to create a clear partitioning between the filtering tasks (landmarks selection, prediction, correction, landmarks initialization) and the vision tasks (feature detection, landmarks correlation). Re-timing may also affect SLAM's performance but it is necessary to meet the execution time constraint on a heterogeneous architecture. In this case the designer should take into account the behaviour of *landmarks correlation* task as its function is finding landmark correspondencies between past and current frame. In cases of e.g. sudden changes in orientation it is highly likely that we lose large number of landmarks. Given *correction* operates on landmarks from precedent frame, quality of the prediction may be corrupted by low (or none) landmark correspondency. Thus, a larger number of landmarks needs to be correlated.

The data parallelism inherent to the image processing could also be explored in the case where the complete image is available for the vision processing tasks. In our case

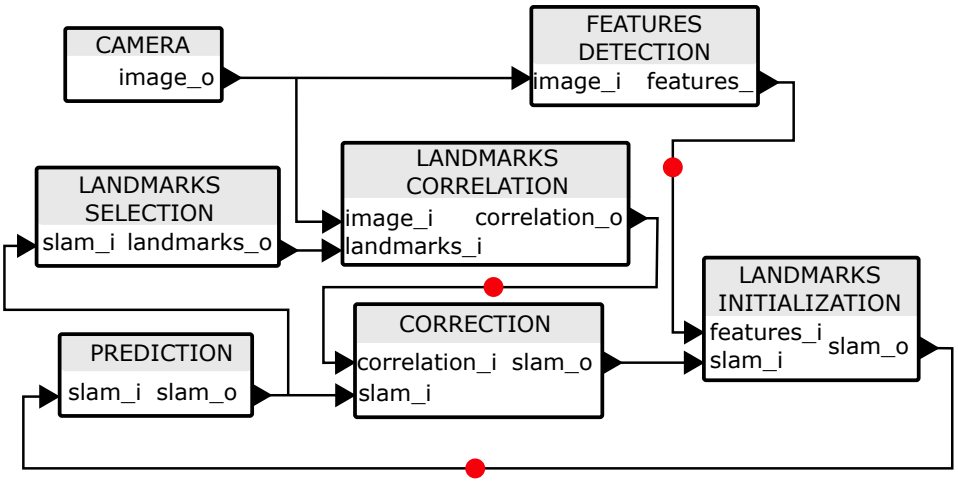

Figure 4.5: Re-timed application SDF model

we consider the image to be a one-dimensional stream of pixels (i.e. when interfacing directly to the image sensor, which is the case in embedded systems). In conclusion to this subsection, we generalize this re-timing method as:

- *correction* task may be any filtering technique
- *features detection* and *landmarks correlation* may be any feature detector with a suitable correlation technique

in a filtering-based visual SLAM application.

4.2.2 Update Specifications

Avnet's ZedBoard is chosen for prototyping our 3D EKF visual SLAM SoC because it comes with:

- an energy-efficient multi-core ARMv7 processor (667MHz),
- a Zynq-7020 FPGA device enabling an additional degree of parallelism,
- 512MB DDR3 RAM
- a Xillinux ([Xillinux, 2010]) distribution which comes quite handy for our needs as it allows memory-mapped and streaming interfaces between FPGA and the host processor(s) at no additional engineering effort.

Relevant execution times of a single-threaded application on ZedBoard are presented in Table 4.1, obtained by means of instrusive software profiling.

Timing performance of these tasks is explained in more detail in Section 4.4. A theoretical scheduling onto three processing elements (PEs) may be seen on Fig. 4.6: with the usual sequential schedule on top and with re-timed schedule after structural-level optimization on bottom[1]. By enabling clear partition between vision and filtering tasks, we realize the potential of real-time execution of our application on embedded architecture with limited resources.

[1]Length of the bars is indicative

Table 4.1: vSLAM tasks execution before iterative prototyping

vSLAM task	time [ms]
landmarks selection	0.8
prediction	2.5
correction	33
landmarks initialization	0.3
camera	13
landmarks correlation	12.78
features detection	219

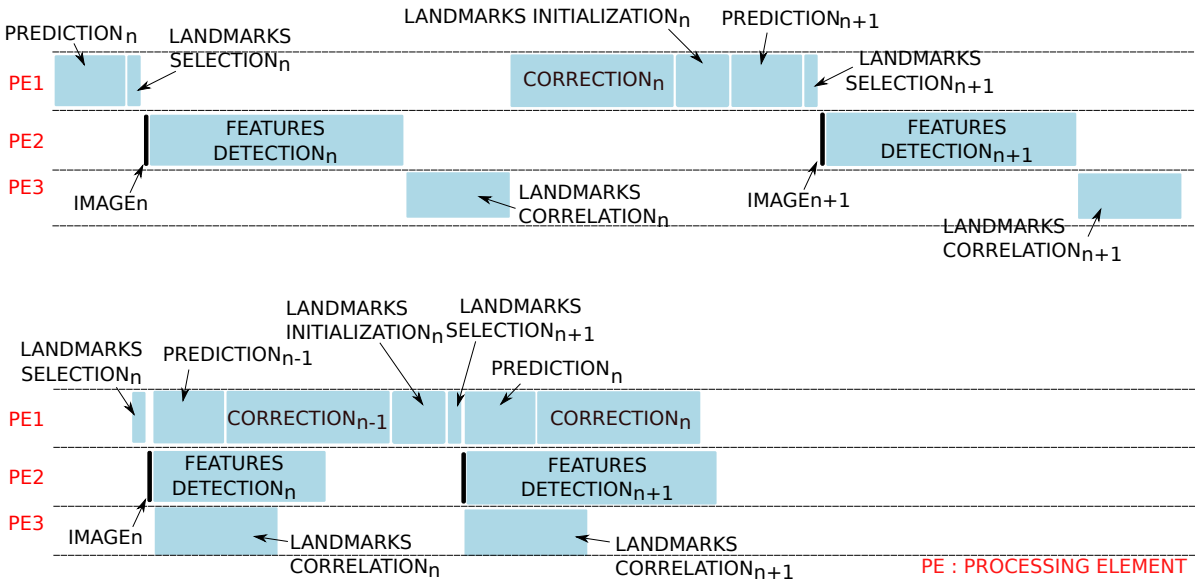

Figure 4.6: Theoretical schedule of the SLAM aplication before and after re-timing

The following Section will explain Iterative Prototyping, the process we intend to follow in order get visual EKF-SLAM's performances on the ZedBoard close to theoretical re-scheduled optimum.

4.3 Iterative prototyping

During hw/sw partitioning constraints defined in subsection 4.1 are revisited. In first prototyping iteration the re-timing technique is applied on software. During co-synthesis, some software modules which regroup vSLAM's tasks functionalities may be further optimized according to CPU's characteristics on heterogeneous platform (e.g. SIMD NEON optimizations for ARM for matrix multiplication and image processing). If the constraints are not met, new HW/SW partitioning is decided upon code profiling of the software: tasks inducing timing bottlenecks are identified and they are next to be implemented in hardware in the form of hw accelerators during Co-synthesis. Careful theoretical analysis is performed before Hardware Description Language (HDL) coding takes place: they must not change the behavioral characteristic of the functional block they replace and at the same time they should have high throughput capacities and low execution times

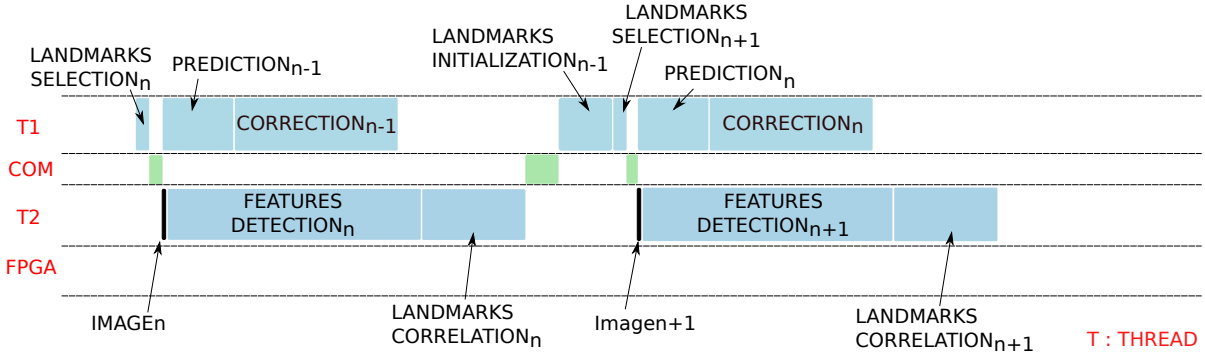

Figure 4.7: Schedule of the SLAM aplication after first iteration

(latencies). Efficient communication capability with the rest of the system (loading and storing data) is also taken into design consideration. From software development point of view, during *SW compilation* we optimize it with respect to processor's characteristics. Chunks of code which need to be accelerated in hardware are rewritten in form of software emulators and passed to *hardware in the loop* block to test the synthesized hardware component in a code-coverage manner.

From hardware design point of view, Hardware In the Loop (HIL) technique is used to build efficient accelerators:

- *Behavioral model* and *Behavioral testbench* are HDL codes conforming to updated hw specifications. They are used to build a valid HDL model of accelerators (*Behavioral model validation*).

- *Synthesized model* is a model that meets timing and area constraints of the FPGA device. In *Synthesized model validation* step we migrate the *Sw testbench* as a standalone bare-metal application with wrapper functions to the *Synthesized model*. If the model successfully passes extensive code coverage techniques, we have a fully functional intellectual property (IP).

After HIL, in hw/sw integration the partitioned design gets intergrated. Here we make use of Xillinux communication interfaces. In order to be able to use them, one must instantiate *Xilly_vga*, *Xillybus_lite*, *Xillybus* IPs on FPGA for display, memory mapped and streaming interfacing between Xillinux and FPGA logic. This process is repeated afterwards with each iteration yielding new reusable IPs, until a target-compliant functional prototype has not been reached.

4.3.1 First iteration: software-only prototype

The first partition is comprised only of software. Application is rescheduled on heterogeneous development board as described in previous Section (allocation of vSLAM tasks) - Fig. 4.7.

Vision and filtering tasks are being allocated (mapped) onto two different threads, as ZedBoard comes with a dual-core ARM host. *COM* stands for First In First Out (FIFO) communication implemented between software threads. Currently there is no task mapped on reconfigurable hardware device (FPGA), entire application is running on the host processor. However, constraints were not met in our case after this iteration.

Thus we reiterate anew: as the *features detection* is the most time-consuming task, the application is partitioned into hardware (feature detection on FPGA) and software components (rest of the application).

4.3.2 Second iteration: Front-end hardware accelerator

Features from accelerated segment test (FAST) is a corner detection method that uses basic arithmetic operations such as additions and comparisons. Hence it is an algorithm suitable to be implemented on FPGA. It is presented in [Rosten and Drummond, 2005] and reviewed in [Mohammad Awrangjeb and Fraser, 2012]. An efficient[2] FAST implementation is given in [Marek Kraft and Kasinski, 2008]. Most often the extracted corners are non-uniformly distributed within the image - Fig. 4.8.

Also the number of corners extracted in similar scenes to outdoors structured (unstructured, natural) remains high (∼3000). However, an EKF-SLAM makes use of sparse features for localization. For a good overall pose estimation, a limited number of uniformly spaced FAST corners (∼50) is preferred. Thus, besides the corner detector, a second hardware acceleration of this front-end vision task is required. The technique of dividing an image frame in a grid of cells (tiles) is called tesselation. It allows to select a feature with highest score within a tile. C-SLAM uses a 16×12 grid in order to insure this scattering.

4.3.2.1 Corner detector FAST

FAST corner detection is composed of two main steps:

- Corner score attribution of the candidate pixel p
- Corner validation of pixel p.

As here an FPGA-based implementation is considered, in further algorithmic analysis the image acquisition process is considered to be a 1-D stream of pixels as on Fig. 4.9:

This streaming representation comes from the fact of actual image acquisition from global shutter cameras connected directly onto the ZynQ-7020 FPGA on ZedBoard. Thus, in first step, the candidate pixel will be referred to as the **current pixel**. Second step uses a so-called *Bresenham's circle* of 16 neighboring pixels to determine whether a candidate pixel p is actually a corner or not - Fig. 4.10.

The corner score function V is thus defined as the sum of absolute differences between the intensity of the central pixel and the intensities of the pixels forming the Bresenham's circle:

$$V = max(\sum_{x \in Sbright} |I_{p \to x} - I_p| - t, \sum_{x \in SDark} |I_p - I_{p \to x}| - t) \tag{4.1}$$

where t is a threshold, $I_{p \to x}$ is the intensity of the segment test pixel and I_p is the pixel's intensity under test. Pixels forming the circle are identified by numbering from 1 to 16, clockwise.

Corner validation step compares each pixel in the circle with the current pixel p: if a set of N contiguous pixels in the circle are all brighter than the candidate pixel p (denoted by Ip) plus an integer-valued threshold t or all are darker than the candidate pixel p minus

[2]In terms of resource consumption and performance

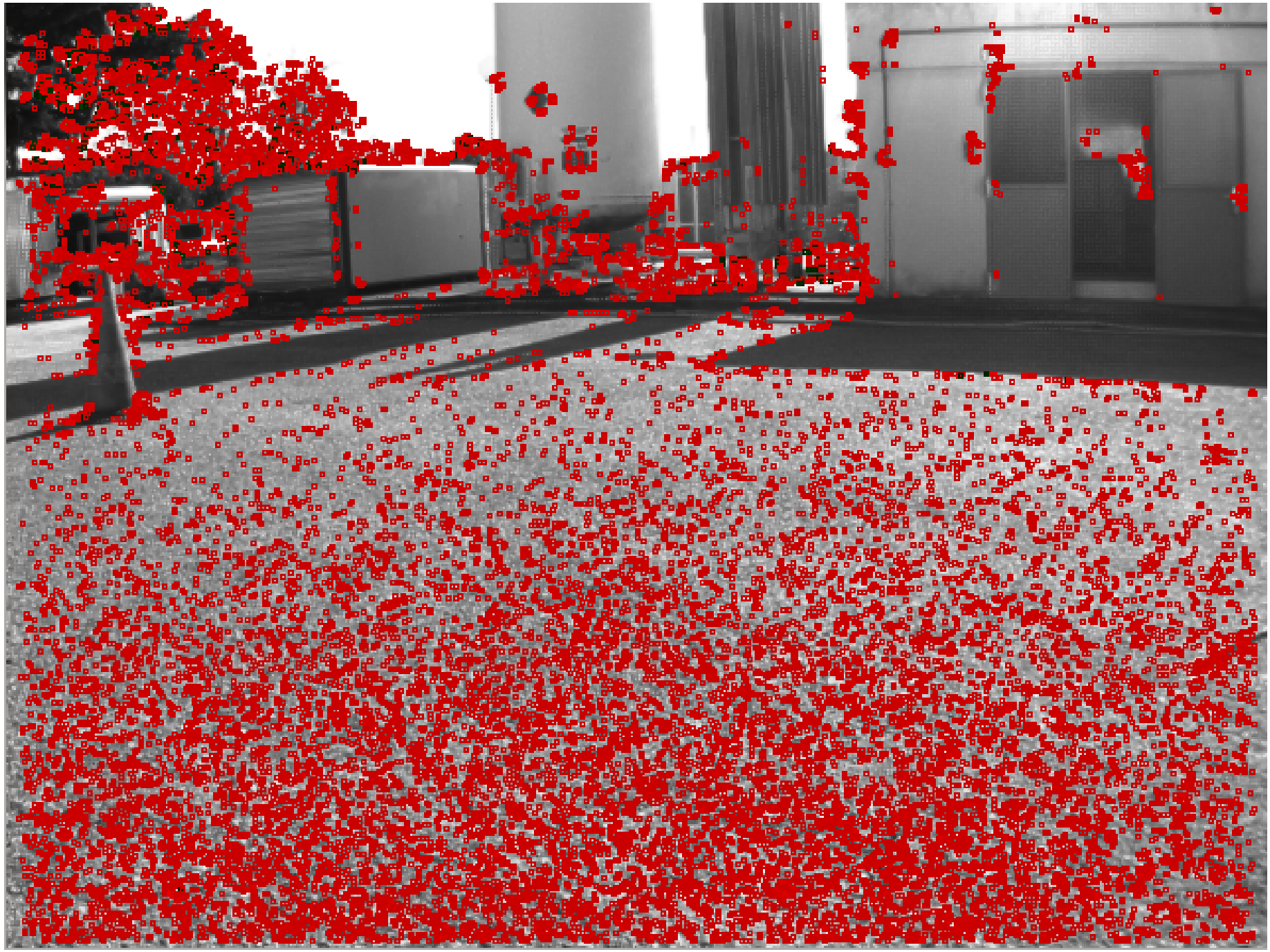

Figure 4.8: FAST corner detection result

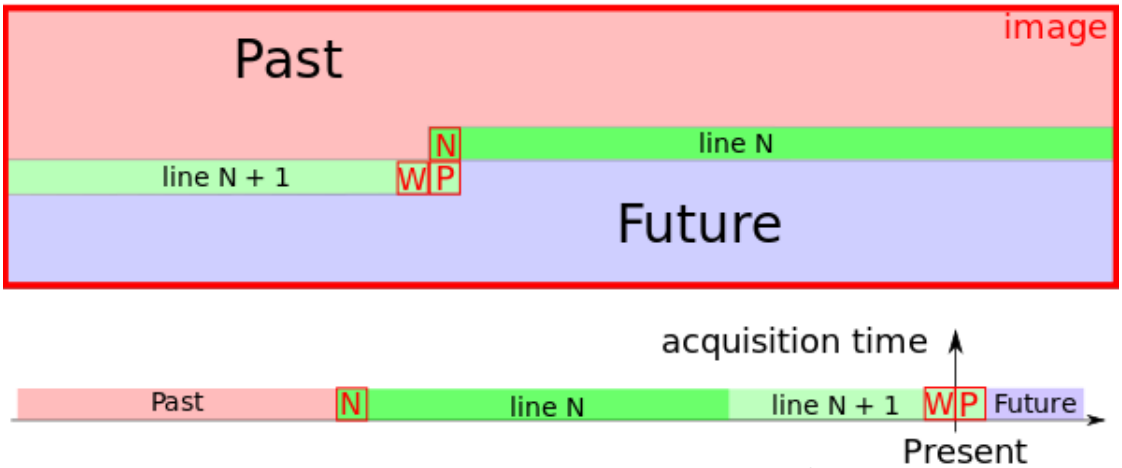

Figure 4.9: Image acquisition as a 1-D pixel stream

the threshold t, then p is a corner. In example given on Fig. 4.10 pixels 1 to 5 have the same value as pixel p. Pixels 6 to 16 are brighter than p. As they are all contiguous, it means that pixel p is a corner and its score is validated.

The FAST hardware implementation follows the algorithmic pattern: it is composed of the corner score and the corner validation modules - Fig. 4.11. Both can be computed in parallel. A block of $N \times N$ pixels stores $(N-1)lines + (N-1)pixels$ in BRAM. In that way, the modules are provided all necessary elements from current Bresenham's circle each clock cycle. Figure 4.12 summarizes the hardware architecture of the corner score computation: The main hardware acceleration is provided by the adder tree instantiation which computes pipelined additions. It allows to perform, in our case, additions of eight 8-bit inputs per clock cycle instead of 8 clock cycles using the host processor. The validation module computes a contiguity test by performing 8 comparisons in parallel. Thus, eight 8-bit comparators are instantiated in order to test arcs of 9 contiguous pixels (the *segment test*. The 9-pixels input segment tests are given by $Inputs[1+m \text{ to } 9+m]$ with $m \in [0 \text{ to } 7]$. The proposed global hardware architecture for the FAST algorithm is generic. The circle size can be easily parameterized and the design can also be implemented on a Xilinx or Altera FPGA. The size of the Bresenham's circle is set to 16 pixels and the contiguity test to an arc of 9 pixels, which is the most efficient configuration according to [Rosten and Drummond, 2005].

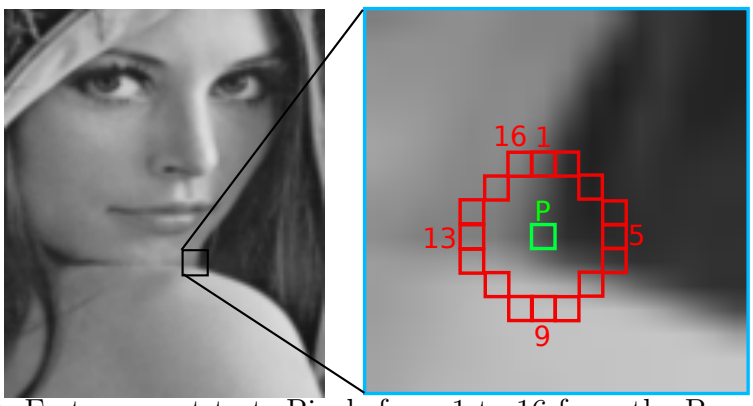

Figure 4.10: Fast segment test. Pixels from 1 to 16 form the Bresenham circle

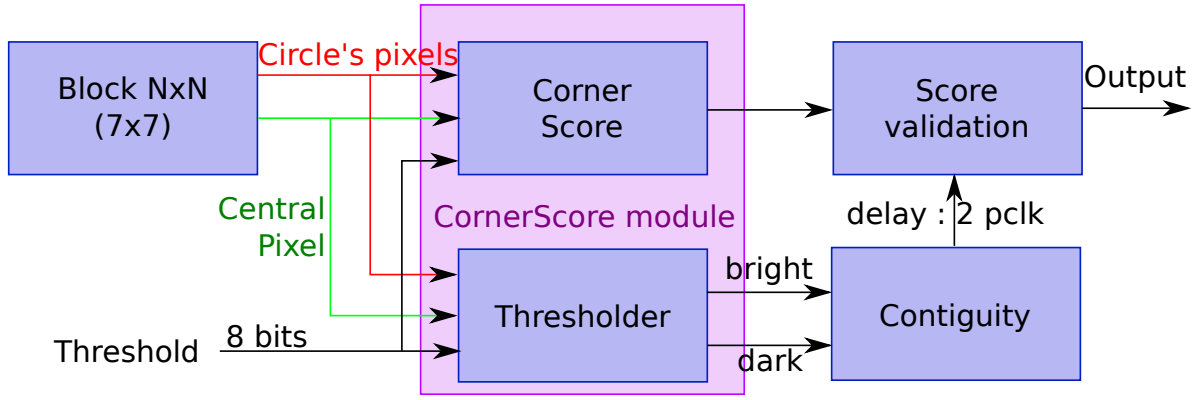

Figure 4.11: FAST hardware architecture

4.3.2.2 Tesselation

Hardware architecture of the tesselation module is mainly based on BRAM usage for storing the best score of each tile. As in case of the FAST implementation, tesselation is also based on a streaming pipeline. The memory consumption (Mem) is given by the following equation:

$$Mem = Cols * (nbrbitY + nbrbitX + nbrbitscore) \tag{4.2}$$

where $Cols$ denotes number of columns in the grid, $nbrbitY/nbrbitX$ stands for number of bits for coding the Y/X tile coordinate and $nbrbitscore$ is the number of bits for coding the score value. In our case these values are as follows: $Cols = 16$, $nbrbitY = 10bit$, $nbrbitX = 10bit$ and $nbrbitscore = 12bit$. Tesselation requires only Mem=512 bit.

Two counters are implemented in order to be able to keep track to which tile the current pixel belongs to. This pixel position allows to forbid the selection of corners near the edge of the image. It also avoids to select points that BRIEF descriptor cannot describe (neighboring pixels that are outside of the frame) - which is set to be at 4 pixels

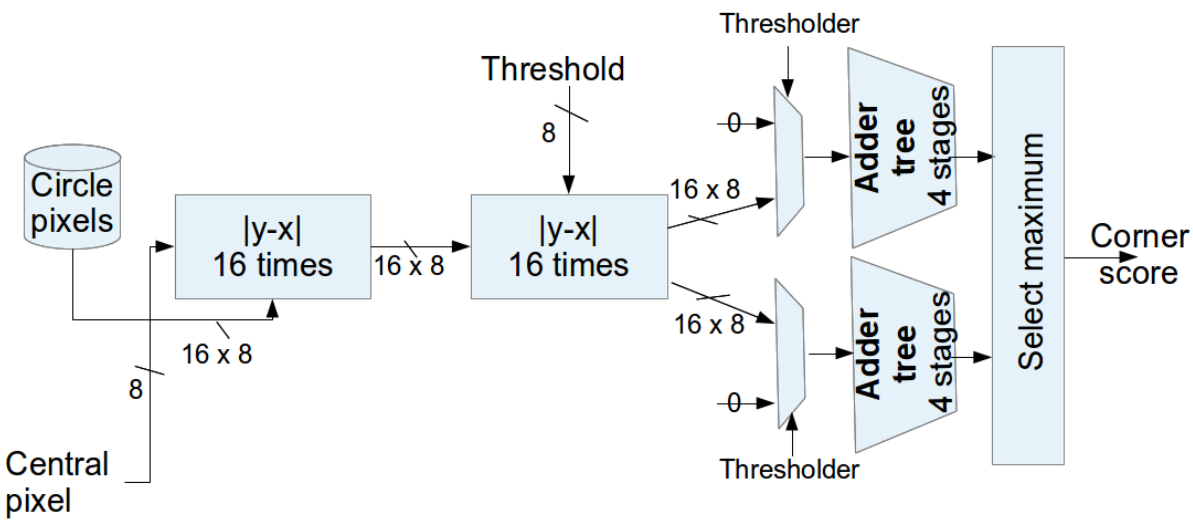

Figure 4.12: Corner score module

on each side of the image. Figure 4.13 shows the grid in black with selected FAST corners having highest scores in a particular tile. Retained corners are painted in blue. Moreover, the latency (L) of this module is fixed and can be calculated as:

$$L = (IMGW/Rows + (FASTrad))_{lines} + 1_{pixel} \tag{4.3}$$

where $IMGW$ is image width, $Rows$ is the number of rows in grid and $FASTrad$ represents Bresenham's circle radius. In our case: $IMGW = 640$, $Rows = 12$ and $FASTrad = 4pixels$ which leaves us with $L = 44_{lines}\ and\ 1_{pixel}$. This chain also provides a fixed amount of data D:

$$D = (Col * rows) * (nbrbitY + nbrbitX + nbrbitscore) \tag{4.4}$$

Settings used for the design are as follows: $Col = 16$, $Rows = 12$, $nbrbitY = 10$, $nbrbitX = 10$ and $nbrbitscore = 12$ bits. It always returns to the host processor $D = 768B$ per frame. That amount accounts for 0.25% of the amount of data that is transferred from a global shutter camera to host processor per standard VGA frame. After integration of this front-end hardware accelerator, 3D EKF SLAM tasks' timings change as on Fig. 4.14. In conclusion to the second iteration it may be said that $FAST$ and $Tesselation$ hardware modules once instantiated (with a fixed grid size) return always the same amount of data to the host processor, which is suitable for minimum bus bandwidth calculation - bus width and clock frequency, given the imposed FPS rate and latency of the rest of the system.

4.3.3 Third iteration: Back-end hardware accelerator

The EKF-SLAM algorithm, relevant for *correction* task, is going to be introduced in this subsection only briefly. Detailed theoretical overview with hardware design implementation ad experimental results will be given in next chapter.

Here, the term *landmarks* replaces selected *FAST corners* from previous subsection in order to be coherent with SLAM's terminology. A state-space based predictive algorithm based on EKF [Maybeck, 1979] runs in a three-steps loop:

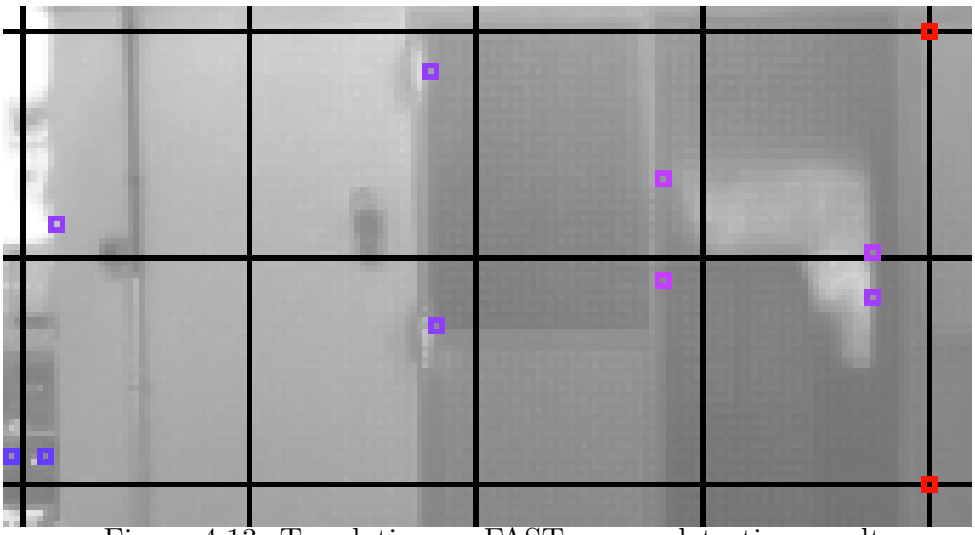

Figure 4.13: Tesselation on FAST corner detection result

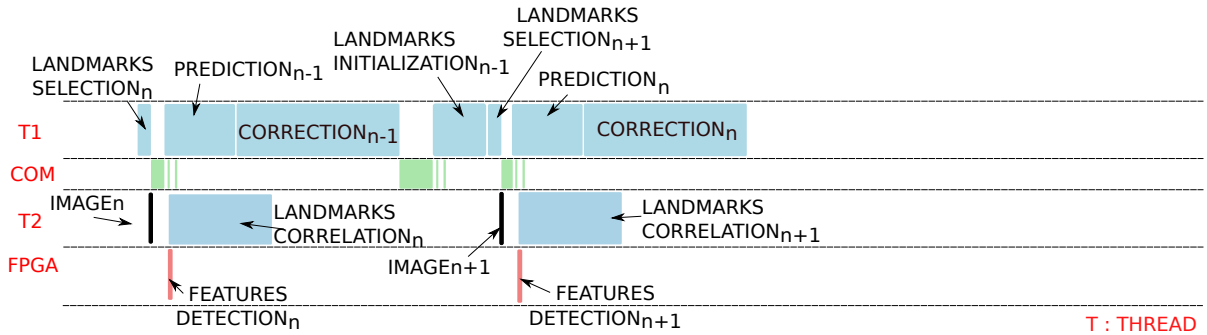

Figure 4.14: Schedule of the SLAM application after second iteration

- Prediction of robot's position from the motion estimates,
- Correction of the robot's and landmarks' positions with respect to the world reference frame (by tracking chosen landmarks) and
- Initialization of landmarks [Solà et al., 2005].

The robot is represented by a vector composed of 3D position, quaternion-based orientation, 9D IMU data, 3D GPS data, and 7D landmark positions (one 7D vector per landmark). During prediction, given a dynamic displacement model of the robot, its pose is being estimated and relevant parts of the cross-covariance matrix (matrix of incertitudes on the state-space related variables of the robot model) are updated. Afterwards, for each observation, a correction loop is run where the part of cross-covariance matrix relative to landmark's position in reference to robot's state vector is updated. Maximum number of observations is chosen empirically[3]. An important particularity of an EKF-SLAM is the ability of updating the whole state-space vector of the robot based on a single observation (single landmark detection after having acquired a new frame) by means of convolving innovation (new landmarks' coordinates perceived from camera after the observation) with cross-covariance matrix.

Let us recall here that our SLAM is implemented without loop closure, so the system position is updated from a local map, with a limited size. Landmarks are removed and replaced by new ones in several situations. (1) They are never matched, due to environmental changes (illumination, occlusions). (2) They have been selected on the surface of a dynamic object. (3) Due to the camera motion, they are overpassed, so they could not be observed during the next motions. Possible losses of landmarks may severely influence the quality of motion estimation. Thus, it is preferable to have many landmarks memorized in the map after initialization and to choose a subset of these upon which position corrections are going to be made. Also, parametrization (dimensioning) of landmarks plays a significant role in their quality [Solà et al., 2011], yielding robuster behaviour with higher landmark state size.

Introducing larger number of higher-dimensional landmarks increases visual EKF-SLAM's computational complexity, having as a consequence in *correction* task larger scale matrix-matrix multiplications. Thus the work in third iteration of the iterative prototyping will focus on designing an efficient FPGA-based matrix-matrix multiplier.

[3]in our EKF-SLAM implementation it is set to twenty

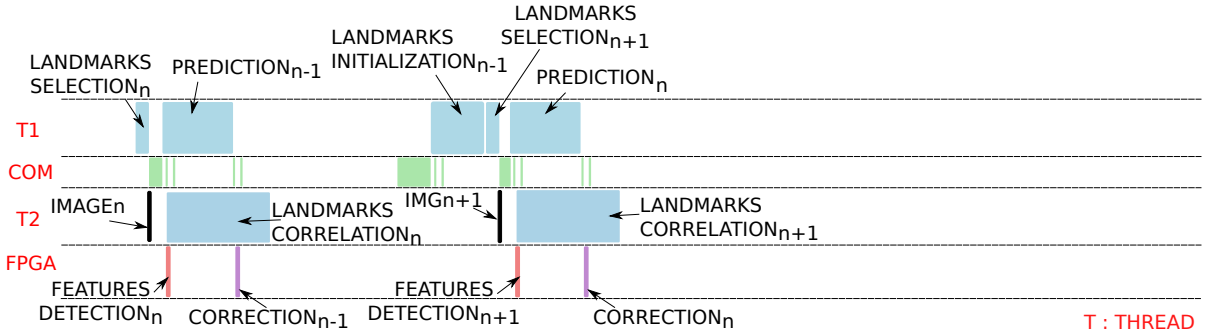

Figure 4.15: Schedule of the SLAM application after third partitioning

After integration of this back-end accelerator, 3D EKF SLAM's tasks' timing changes as on Fig. 4.15.

4.3.4 Fourth iteration: Front-end hardware accelerator

The goal of the fourth iteration is to compute the correlation on reconfigurable hardware. This task is the last vision-related operation. The benefit of its integration lies in avoiding image transfers between FPGA and host processor.

4.3.4.1 Correlation accelerator co-design

In the SLAM process, the correlation task measures the similarity between two features. Choosing BRIEF descriptor to describe a pixel leads to using the Hamming distance in order to correlate the two features' descriptors, as it is a binary feature descriptor. The first vector input is the feature's BRIEF descriptor that we are trying to match in the current frame ($BRIEF_{ref}$). The second vector represents the current BRIEF ($BRIEF_{current}$). Computing Hamming distance (HDist) is shown in following equations:

$$tmp = BRIEF_{ref}\ xor\ BRIEF_{current} \tag{4.5}$$

$$HDist = nOnes(tmp) \tag{4.6}$$

where $nOnes()$ counts the number of bits equal to '1' in tmp vector. This function is implemented by instantiating registers in a static array where the 6-bit Hamming Distance vector is stored (tmp is divided into $n * 6$-bit vectors). The results are summed together in a pipeline in order to provide the total Hamming distance. This induces a latency of 4 clock cycles in our design which in return consumes fewer logic cells than a direct Hamming distance instantiation. The following pseudo-code explains how the hardware implementation executes this task in a given ROI (Region Of Interest):

```
1: if Xpos AND Ypos ∈ ROI then
2:     HDist=nOnes(BRIEF_current xor BRIEF_ref)
3:     if HDist > distance_latched then
4:         distance_latched = HDist
5:         result_available ='0'
6:     end if
7: else
```

```
    if Xpos AND Ypos = endROI then
        result_available ='1'
    end if
end if
```

where *distance_latched* is set at maximum at the beginning of the ROI and then updated at each smaller Hamming distance. Figure 4.16 shows a correlator core and the principle of instantiating multiple cores as to be able to process n correlations in parallel (in case of ROI overlaps). In our case, $n = 20$.

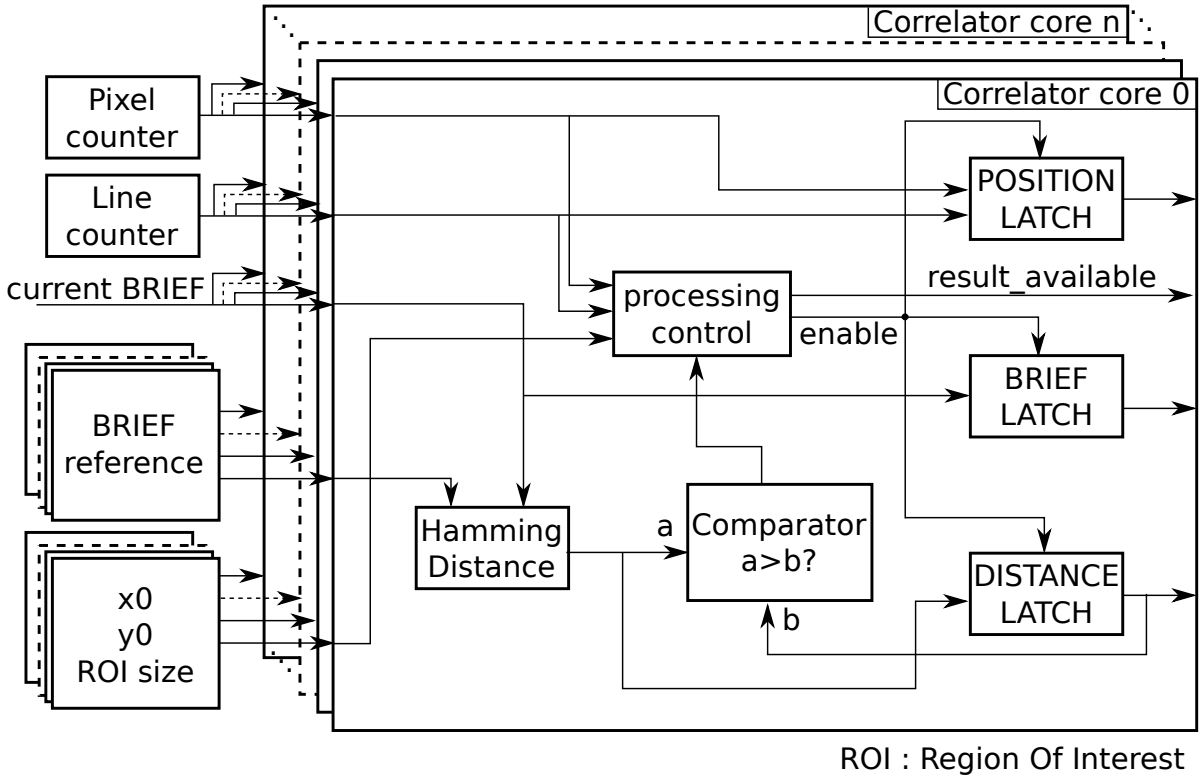

Figure 4.16: Instantiation of a multicore correlator

Data for correlation ($BRIEF_{ref}$, $ROIsize$ and $X0/Y0$) and the correlation results (X,Y, $BRIEF$ and $HammingDistance$) are stored in a same BRAM which can be read/written to both by hardware and ARM host. A signal $result_available$ is asserted to notify the end of the computation.

After integration of this front-end accelerator, 3D EKF SLAM tasks' timing changes as on Fig. 4.17.

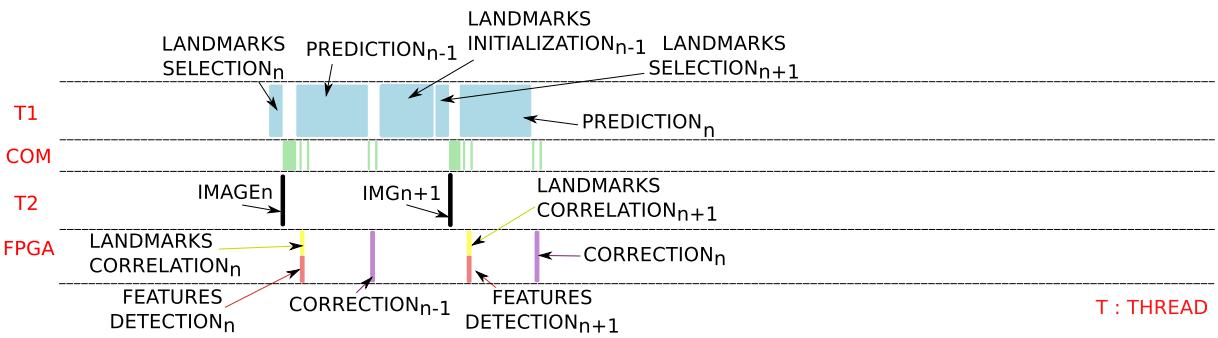

Figure 4.17: Schedule of the SLAM application after fourth partitioning

4.4 Final hardware prototype

This section begins with a description of our visual 3D EKF SLAM's SoC architecture after iterative partitioning process. Afterwards, the obtained experimental results are being discussed and put into perspective with state of the art.

The developped hardware architecture consists of three IPs instantiated in reconfigurable fabric of a ZynQ-7020 device: Fast+Tesselation, EKF and Correlator. All vision tasks and the most time-consuming task in EKF are executed on FPGA, while the rest is scheduled on ARM under Xillinux (user space). Hardware-accelerated functions are integrated into the embedded system in form of co-processors. In order to be able to communicate with our IPs via Xillinux middleware (kernel space), *Xilly_vga*, *Xillybus_lite* and *Xillybus* IPs are also instantiated on the FPGA - (Fig. 4.18).

FAST+Tesselation accelerator is interfaced with Xillinux through Xillybus FIFOs *camera_fifo* and *FAST_fifos* which are accessed in user space through files. The first is used for frame pixels reception, which are sent from image files in Xillinux. All the image sequence, IMU and GPS data used by our application are pre-registered and stored in external memory. This *features detection* logic runs at 89MHz, limitted by maximal throughput

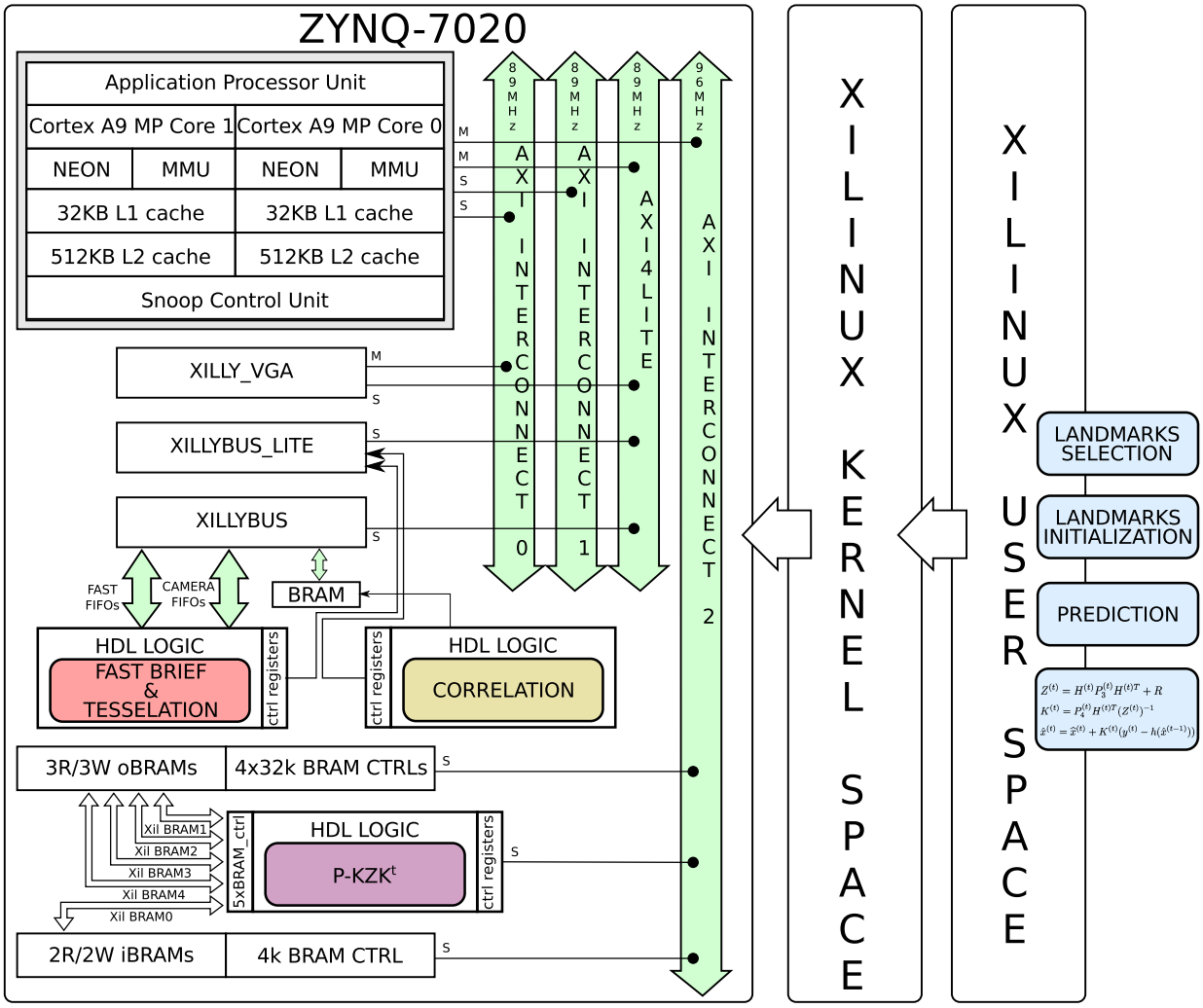

Figure 4.18: Embedded SLAM's SoC architecture

Table 4.2: Prototype's resource usage with IPs' performance

Resources	instantiated	%FPGA
LUTs	27256	50
Slice registers	34127	31
DSP48E	40	18
BRAM [36Kb]	51	36
BRAM [18Kb]	15	6
ARM Cortex A9	2(cores)	100
F_{MAX} [MHz]	individually	integrated
FAST+Tesselation	166	89
Correlator	204	89
Eq5.7	198	96
Power consumption [W]		
ZedBoard	-	4.56 (0.671)

supported by *Xillybus* IPs. Control signals are communicated from host through *Xillybus_lite*, which serves as a memory-mapped (host to FPGA BRAM) interface. Correlator communicates with middleware as a memory-mapped device.

Our modification to this prototyping of-the-shelf architecture was introduced in third *CO-sythesis* loop by changing the *devicetree.dts* file so that this architecture could support addressing through an additionally instantiated AXI4 stream bus *axi_interconnect_2* for the EKF accelerator (*Eq5.7*). In this way we were able to set-up a new clock domain (96MHz) and to facilitate the HIL process. An individual bus is also an imperative because of P matrix related transfer rates - Table 5.3.

4.4.1 Experimental results and discussion

In Table 4.2 we can see the summary over Zynq-7020 device's resource usage. BRAM's usage is high because apart from instantiating it for P matrix' storage, it is also used for implementing Xillybus FIFOs (6×BRAM36K). Maximal frequency of individual accelerators is higher than actually set, because of the timing issues of the integrated bare-metal design with Xillybus IPs. Xilinx Power Analyzer is used to mesure power consumption in reconfigurable logic (0.671W) - this measure does not include ARM's power consumption. We used a multimeter in order to measure current during application execution (380mA), and as ZedBoard consumes 12V we deduce its maximum consumption to be 4.56W. The value is high because of other active peripherals on the board (which are not used by SLAM application): Ethernet controller, LEDs, OLED display...

Performance measures in frames-per-second of different versions of our SLAM application are given in Table 4.3. First column presents the multi-threaded SLAM before having applied re-timing techniques, First column presents SLAM tasks execution with re-timing, second, third and fourth columns the impact of these timings after relevant hardware acceleration. Total execution time of functional blocks is measured using software-intrusive code profiling. FIFO communication between software threads takes a lot of time (12ms).

Table 4.3: Execution time of SLAM functional blocks in [ms] on ZedBoard

Iteration no.	I	II	III	IV
COM	12	12	12	12
landmarks selection	0.8	0.8	0.8	0.8
prediction	2.1	2.1	2.1	2.1
correction_slam	33	33	5	5
landmarks initialization	0.3	0.3	0.3	0.3
camera	13	13	13	13
landmarks correlation	8.8	8.8	8.8	0.6
features detection	140	15	15	15
FPS rate [Hz]	6	20	21	24

Also does loading frames from external memory(13ms). Although *features detection* task is much faster now, huge percentage of execution time goes on FIFO communication : $15 - (640 \times 480)/89MHz = 11.55[ms]$. The accelerated task *correction* takes only $20 \times 0.25 = 5[ms]$, where 0.25 ms is the time needed for single landmark correction. Thanks to multicore Correlation blockm, matching of observed landmarks takes minimal time now.

As the target-compliant prototype lacks real-time execution because of Xillinux communication interfaces, FIFO inter-thread communication and latency to fetching images from storage medium, we conclude our approach to be validated after four iterations - Fig. 4.19.

4.4.2 Comparison with state of the art

Three embedded FPGA-based SLAM systems exist to our knowledge: monocular 2D EKF SLAM of [Bonato et al., 2008], works previously led in LAAS [Gonzalez et al., 2011] on monocular 3D EKF SLAM and 3D bundle adjustment SLAM of [Nikolic et al., 2014]. However, these works do not present complete SLAM systems as they don't implement vision and pose estimation related accelerators on a single reconfigurable device. The first paper does not mention what algorithm for features detection the authors use and besides their map is 2D, which significantly downsizes the complexity of their problem compared to ours. On the other hand, authors of the second paper use a ZynQ device for preprocessing vision tasks - features detection and correlation, from multiple cameras along with synchronization of IMU data. Moreover they mention themselves in conclusion that their future work envisages a "SLAM in a box" module on a single embedded device, as the computationally demanding bundle adjustment is ported onto power-consuming Core2Duo host. Lastly, authors in [Gonzalez et al., 2011] implement monocular SLAM functionality similar to ours, but they achieve same performances (24Hz) with same amount of observations and corrections although they dissipate more power (two FPGAs) and they use lesser dimensioning of landmarks (IDP).

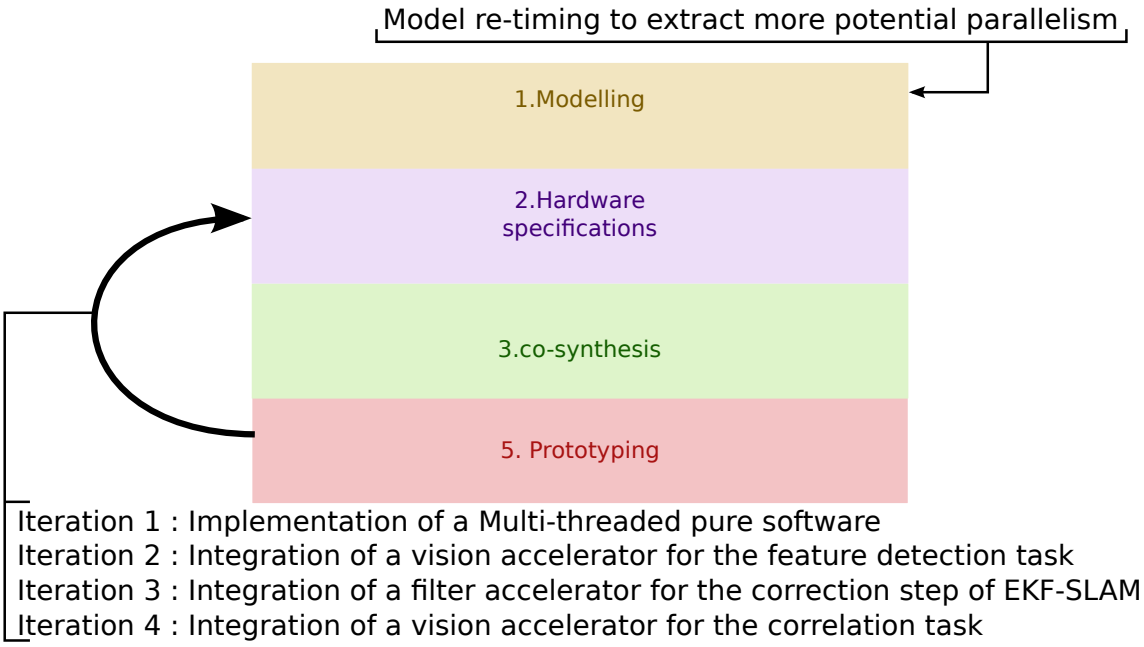

Figure 4.19: Summary over the iterative prototyping

4.5 Conclusion

In this chapter an application of a hw/sw co-design methodology on our visual 3D EKF SLAM application is presented. At modelling phase, model-driven modelling helped us to refine the application model before starting the work on implementation. Moreover this development flow permits to validate the application behavior for every step before moving to the next, resulting in a know-to-work final prototype. The performance constraints defined for the application were hard to meet with a naive implementation of the EKF vision-based SLAM (especially in terms of latency) running on an embedded system. The model-driven flow allowed to take performance into account early in the design flow thus leading to model-level rework. It limited the number of design iterations (4 design iterations) where an implementation-only flow would have required major rework for each iteration and a large number of iterations before meeting the constraints. It is showed that a SLAM paradigm may be mapped in general on three parallel executors: position estimation, corellation and feature detection. Based on that fact, our application is chosen to be implemented on a heterogenous (processor + FPGA) hardware development board resulting in a SoC prototype. Performing four iterative hw/sw partitionings followed by co-syntheses, we integrated three performing FPGA accelerators into the 3D EKF SLAM SoC prototype: Fast feature detector with Tesselation, EKF-accelerator and landmarks Correlator, without any algorithmic changes. The application is accelerated from initial 5Hz to 24Hz. Real-time constraint of 30Hz is not met because of timing delays in communication between software and hardware modules. Our prototype presents a behavioraly verified and fully functional 3D EKF SLAM SoC with high processing power and low power consumption of merely $4.56(0.671)W$.

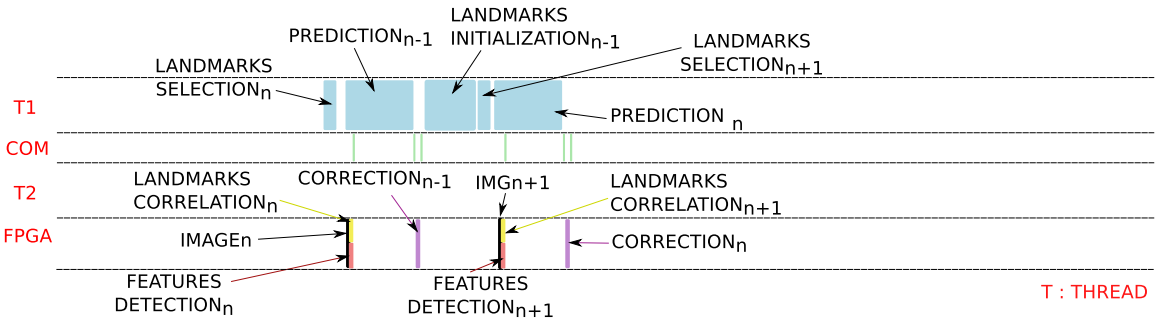

Figure 4.20: Expected schedule of the SLAM application after fifth partitioning

4.5.1 Towards EKF-SLAM at 100Hz

The prototyped architecture is target-compliant and satisfies real-time constraints in terms of processing throughput. Further work takes on interfacing the ZynQ-7020 device directly with an embedded camera - Fig. 4.20, as we presumed that *camera* task outputs a 1D pixel flow in our application model (data-flow). That would remove the 13ms delay, which would yield an over-performing prototype.

Once we have the embedded camera wired with the ZynQ, our prototype may be evolved into a consumer-grade product. We intend to exploit the three developped IPs with rescheduled EKF SLAM application in a bare-metal design where the reconfigurable fabric would permit higher bandwidth (at least 150MHz instead of 89MHz). On the other hand, once freed from the delays induced by FIFO software communication and Xillinux interfaces, we estimate[4] that vision tasks and filtering tasks would roughly take at most 4.1ms and 8.2 ms respectively, with filtering part being the new bottleneck. That would roughly bring us to a monocular EKF SLAM functionality with local maps containing 20 AHP points (and observing all of them) over 100Hz rate at very low power consumption, which even outperforms RTSLAM's - [Roussillon et al., 2011] execution (when same parameters used) on an Intel i7 core.

[4]estimation is made upon timings in Table 4.3

Chapter 5

EKF-SLAM HW/SW co-design: part II

Contents

5.1 Algorithmic overview **67**

5.1.1 Computational bottleneck identification 68

5.2 Design considerations **69**

5.2.1 Matrix multiplication tradeoffs on FPGAs 69

5.2.2 Notion of systolic arrays on an FPGA 70

5.2.3 Computational model mapping 70

5.3 Cross-covariance matrix block accelerator **73**

5.3.1 HW/SW Integration 76

5.4 Design evaluation **78**

5.4.1 Multiplier Latency 78

5.4.2 Resource usage 78

5.4.3 Power consumption 79

5.5 Experimental results **80**

5.5.1 Measured multiplication latencies 80

5.5.2 Design scalability 81

5.5.3 Power per feature 81

5.5.4 Future work 81

5.6 Conclusion **83**

This chapter will start off with co-design of the back-end hardware accelerator for *correction* task. Afterwards, an extensive evaluation of the developed accelerator will be given in terms of resource usage, latency and power consumption.

5.1 Algorithmic overview

Equations of the EKF are given below[1].
Prediction loop:

$$\hat{x}^{(t)} = f(\hat{x}^{(t-1)}, \varpi^{(t)}) \tag{5.1}$$

$$P_1^{(t)} = F_x^{(t)} P_1^{(t-1)} F_x^{(t)T} + F_\omega^{(t)} Q F_\omega^{(t)T} \tag{5.2}$$

$$P_2^{(t)} = F_x^{(t)} P_2^{(t-1)} \tag{5.3}$$

Correction loop:

$$Z^{(t)} = H^{(t)} P_3^{(t)} H^{(t)T} + R \tag{5.4}$$

$$K^{(t)} = P_4^{(t)} H^{(t)T} (Z^{(t)})^{-1} \tag{5.5}$$

$$\hat{x}^{(t)} = \widehat{x}^{(t)} + K^{(t)}(y^{(t)} - h(\hat{x}^{(t-1)})) \tag{5.6}$$

$$P^{(t+1)} = P^{(t)} - K^{(t)} Z^{(t)} K^{(t)T} \tag{5.7}$$

After each frame acquisition, the front-end process furnishes landmarks found by FAST corner detector. Active search algorithm is used to find correspondences with observed landmarks in the past frame. In our case, we observe and search for correspondencies between 20 interest points. Matched points are passed through in the camera reference frame. They are afterwards reparameterized to richer AHP representation, which is the data type upon which our Kalman filter works. This information is stored in robot state vector $\hat{x}$ and is updated in the process within Jacobian (H) and cross-covariance (P) transformations - Table 5.1.

An EKF loop is executed as follows: first, a prediction step is performed in order to update the corresponding elements in P : 5.1 - 5.3. Matrices P_1, P_2, P_3 and P_4 are sub-matrices of the cross covariance matrix P. Secondly, multiple correction loops are run, each with respect to a single observed landmark: equations 5.4 - 5.7 (in our case 20 times). In correction loop the filter's gain is calculated, upon which landmarks' and robot's positions are being updated (along with the the entire cross-covariance matrix). It is done in equation 5.6 where IMU odometry and GPS data are integrated in measurement process y^t (localization). Upon observed, matched and corrected landmarks all landmarks in the map are reparameterized from AHP to three-dimensional Euclidean representation (3D mapping). After correction loops, we are able to reduce the search space for landmarks which we are observing in the following frame.

In Table 5.1, N stands for the total number of landmarks retained in the local map, d is the dimension of AHP feature state size (which is seven), six is the number for GPS and IMU related variables and r is the state-space size of our robot (which is nineteen). All twenty landmarks in the map are observed. From that table we deduce Table 5.2 which shows the total amount of floating-point operations that are executed in an EKF block in function of the number of retained landmarks N in the visual map.

[1]In equations 5.2 and 5.7 we perform matrix calculations only on P_1 and P matrices' upper triangles respectively, as the cross-covariance matrix P is symmetrical

Table 5.1: Description and matrices dimensions according to our implementation of the EKF algorithm.

Symbol	Dimension	Description
$\hat{x}$	$(dN + r) \times 1$	Robot and feature positions
f	-	Prediction function
ϖ	r	Control law
P	$(dN + r) \times (dN + r)$	Cross covariance matrix
P_1	$r \times r$	Cross covariance matrix with respect to robot position
P_2	$r \times dN$	Cross covariance matrix with respect to all landmarks
P_3	$2d \times 2d$	Cross feature-robot covariance matrix
P_4	$(dN + r) \times 2d$	Cross feature-feature and robot-robot covariance matrix
F_x	$r \times r$	Jacobian to system state
F_ω	$r \times 6$	Jacobian to system state perturbation
Q	6×6	Permanent perturbation
Z	2×2	Covariance innovation
H	$2 \times 2d$	Jacobian
R	2×2	Measurement noise
K	$(dN + r) \times 2$	Filter gain
y	2×1	Measured output
h	-	Observation function

5.1.1 Computational bottleneck identification

It is shown that computational requirements for an EKF algorithm depend on the number of features N retained in the map: $L_{EKF} = O(N^2)$ [Thrun et al., 2005]. Highest share of the execution time belongs to equation 5.7. Here, the upper triangle of the square P matrix is updated during correction. This fact is confirmed after having run a code profiling tool [Spivey, 2003] over the implemented EKF-SLAM code (which will be more detailed in section 5.5). Thus, a significant speed-up can be made by leveraging the $P - KZK^T$ operation in hardware.

Knowing that square matrix P is the largest data structure in SLAM algorithm in terms of memory usage and that it is being frequently evoked for pose estimation and state update, it is an imperative to ensure efficient data communication between the

Table 5.2: Number of floating-point operations in an EKF loop taking into consideration symmetry of the cross-covariance matrix. Equations 5.4 - 5.7 are executed c = 20 times, for each observation

Equation no.	FLOP	N=20	N=50	N=100
5.1	361	0.02	0.006	0.002
5.2	17000	1.21	0.26	0.074
5.3	4921N	6.99	3.8	2.13
5.4	c × 868	1.23	0.27	0.08
5.5	c × (420N + 1140)	13.54	6.85	3.74
5.6	c × (49N + 135)	1.58	0.8	0.44
5.7	c × ($102N^2$ + 576N + 811)	75.43	88.01	93.54
Total	$2040N^2$ + 25821N + 76441	100		

Table 5.3: Transfer analysis regarding P matrix in EKF equations

%xfers on	N=6	N=13	N=20
Eq5.2	1.21	0.2	0.06
Eq5.3	2.68	0.95	0.48
Eq5.4	3.95	1.4	0.7
Eq5.5	17.2	11	7.96
Eq5.7	74.96	86.45	90.8

accelerator and the rest of the system. In Table 5.3 we can see the influence of N^2 in equation 5.7 related to number of element-wise memory accesses by changing the value of N. An accelerator which would transfer back all the results into external memory (where program data are stored) would eventually lose its role with growing N. By storing P matrix in on-chip memory instead of storing it in external memory and modifying the software to access the contents of P in on-chip memory when executing equations 5.2,5.3,5.4 and 5.6 would in addition gradually increase the efficiency of the accelerator. It is why we propose to store it on high-speed on-chip memory. In that way we could keep the FLOPs number in equation 5.7 close to a constant value and thus rendering it independent of N^2 in terms of host processor execution time. For a generic EKF-SLAM application, the parameter c may be varied, which also influences considerably FLOPs number in equation 5.5. Thus, hardware acceleration of this equation will be also taken into account.

5.2 Design considerations

This section will start with floating-point matrix multiplication considerations on an FPGA device (latency, I/O bandwidth and storage). After having introduced a Systolic Array (SA) - based processing system on an FPGA, a processing element (PE) - based computational flow for EKF's equation 5.7 will be given.

5.2.1 Matrix multiplication tradeoffs on FPGAs

On a reconfigurable computing system the main tradeoff is between optimal speed and resource utilization. Higher design speeds may be achieved at the cost of a higher resource usage. On the other hand, the limited number of configurable slices should be used for parallel logic implementation. Considering applications of matrix-matrix multiplication algorithms on programmable hardware of limited size, such as FPGAs, we have to take into account specific constraints on latency L, total storage size in words M and memory bandwidth requirement B denoted by number of I/O operations performed in each cycle. Based on a multiplication of two n-squared matrices, authors in [Zhuo and Prasanna, 2007] explain these tradeoffs in reference to lower bound on achievable latency:

$$L \geq max(L_1, L_2) \geq max\left(\frac{n^3}{p}, \frac{n^3}{\sqrt{M}B}\right), (M \leq n^2) \tag{5.8}$$

where p is the number of theoretical Processing Elements (PE) that may be executed in parallel, L_1 is the latency according to computation time and L_2 is the latency dependent on I/O bandwidth. Furthermore, they conclude that an optimal latency is in this case of order $O(n^2)$ when having $p = O(n)$ PEs and $M = O(n^2)$ available storage size in words. However, for large scale matrices the latency is of order $O(\frac{n^3}{p})$. This is due to the fact that having FPGAs with limited resources it is hardly possible to instantiate that many PEs in parallel. A recent work [Jovanovic and Milutinovic, 2012] describes an architecture of linear array PEs, similar to those in [Zhuo and Prasanna, 2007], but achieving an optimal latency of order $O(n^2)$ by exploiting full-duplex communication with the host processor (no on-chip memory storage) and at the cost of having it involved during addition of intermediary values. Our design will focus on minimal latency of matrix multiplications in EKF equation 5.7 and on-chip memory storage.

5.2.2 Notion of systolic arrays on an FPGA

As previously explained, and especially due to available resources on the Zynq-7020 device, we chose to optimize the tri-matrix multiplications. Pipelined designs present a suitable solution when it comes to allocating resources given the specific problem size of our computational model. By identifying computational kernels (i.e. floating point multiplication, addition and subtraction) a problem-specific computational unit called PE may be conceived. Systolic array (SA) is a form of a pipelined design consisting of a multi-dimensional grid of interconnected PEs. Main sequential operation is being optimized (e.g. equation 5.7 in our case) by restructuring its executional flow so that it may be executed on parallel PEs. Besides, a central logic, called *Sequencer*, controls the data flow in the grid. Advantage of a SA based computation over sequential or multi-threaded is that the its grid is able to generate new results after each clock cycle. In consequence, by exploiting structural properties of an SA we are able to lower the dimension of the computational model (which is three-dimensional for n-squared matrices multiplication) and obtain a better throughput at the cost of more complex control logic of data flows.

Authors in [Castillo-Atoche et al., 2010] present useful mapping techniques inherent to an FPGA-based design. In order to avoid using large number of buffers due to routing issues in a standard matrix-matrix multiplication algorithm, they propose a modified locally recursive version upon which they map control logic onto their two-dimensional SA. However, because of speed issues, PEs are often organized in a linear list structure - a special case of a one-dimension systolic array in which they interconnect by short connection wires. In general, broadcasting data on a larger grid in an FPGA using a higher dimensional SA structure is not recommended because of speed degradation due to the size and routing complexity of FPGA-based floating point units. In order to achieve efficient low latency design we focus our efforts on implementing a 1-D SA.

5.2.3 Computational model mapping

The tri-matrix multiplication in equation 5.7 could be performed on hardware by taking into account the identity: $A \times B = (B^T \times A^T)^T$ as in that way the larger matrix K has to be transposed only once during the computation. In same manner we rewrite the equation as:

$$P^{(t+1)} = P^{(t)} - (Z^{(t)T} K^{(t)T})^T K^{(t)T} \tag{5.9}$$

The computation model of a one-dimensional SA with even number of PEs ($p = 2k | k \in \mathbb{N} \setminus \{0\}$) may be described with the following pseudo-code:

Phase I: $A \times B$

1: **for** $i = 0; i < 7N + r; i+ = p/2$ **do**
2: $(in\ parallel\ for\ all\ x = i : 2 : (i + p - 1), y = i : 1 : (p/2 - 1))$
3: $PE_{xmodp} = A_{xmod2,0} \times B_{0,y} + A_{xmod2,1} \times B_{1,y}$
4: $PE_{(x+1)modp} = A_{(x+1)mod2,0} \times B_{0,y} + A_{(x+1)mod2,1} \times B_{1,y}$
5: **end for**

where $A = Z^T$, $B = K^T$ and

Phase II: $C - A \times B$

1: **for** $i = 0; i < 7N + r; i+ = p$ **do**
2: **for** $j = i; j < 7N + r; j++$ **do**
3: (in parallel for all x = i : (i+p-1))
4: $PE_{xmodp} = C_{x,j} - (A_{x,0} \times B_{0,j} + A_{x,1} \times B_{1,j})$
5: **end for**
6: **end for**

where $C = P$, $A = [Z^T K^T]^T$ and $B = K^T$.

In order to minimize changes in control logic (and resources) used for hardware matrix-matrix multiplication, the computational model used for equation 5.7 will be reused for computing equation 5.5. Thus we rewrite it as:

$$K^{(t)} = ((Z^{(t)^{-1}})^T (H^{(t)} P_4^{(t)T}))^T \tag{5.10}$$

and apply the same PE-based computational model like before:

Phase I: $C + A \times B$

1: **for** $k = 0; k < d; k++$ **do**
2: **for** $i = 0; i < 7N + r; i+ = p/2$ **do**
3: $(in\ parallel\ for\ all\ x = i : 2 : (i + p - 1), y = i : 1 : (p/2 - 1))$
4: $PE_{xmodp} = C_{xmod2,y} + A_{xmod2,2k} \times B_{2k,y} + A_{xmod2,1+2k} \times B_{1+2k,y}$
5: $PE_{(x+1)modp} = C_{(x+1)mod2,y} + A_{(x+1)mod2,2k} \times B_{2k,y} + A_{(x+1)mod2,1+2k} \times B_{1+2k,y}$
6: **end for**
7: **end for**

where $C = K^{T'}$, $A = H$ and $B = P_4^T$. In inner loop the variable $K^{T'}$ accumulates the partial products of $H \times P_4^T$. Outer loop runs d times, executing the phase I control logic in equation 5.7. Phase II performs $A \times B$, identicaly as in phase I in equation 5.7, only here $A = [Z^{-1}]^T$ and $B = HP_4^T$.

Our computational model (even number of PEs) builds up on the fact that the innovation matrix Z is always square-two dimensional for a visual EKF-SLAM application. It supports varying parameters N, r and d from Table 5.1 for a different robot configuration and landmark parametrization scheme. A PE well adapted for this task contains two floating-point (FP) multipliers and two FP adders and is executing the operation:

$$C \pm A \times B \tag{5.11}$$

An unidirectional SA containing four PEs may be seen on Fig. 5.1:

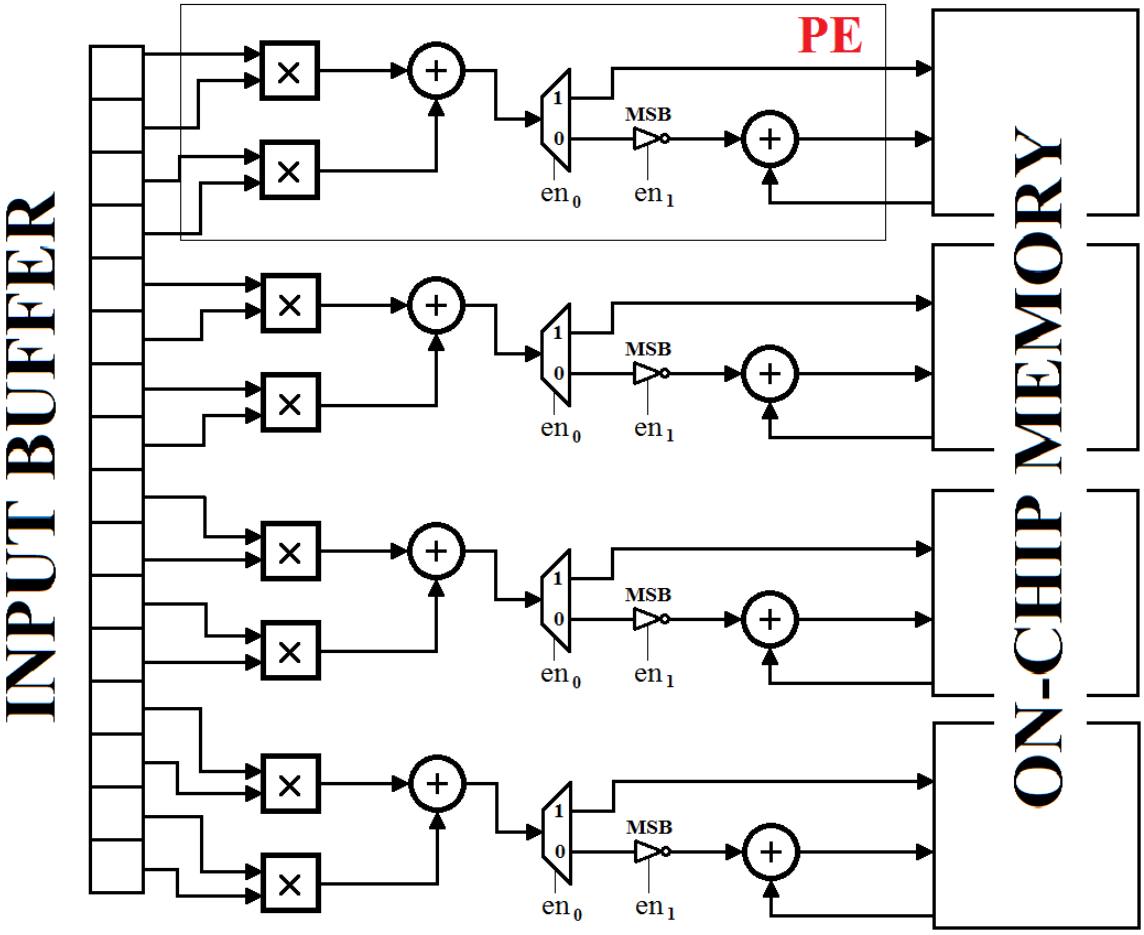

Figure 5.1: Our tiling of the 1-D systolic array for visual EKF-SLAM matrix multiplications

Floating-point subtraction is performed by negating the most significant bit (MSB) of the second operand (in IEEE FP format this negates the given number value) in addition operation - Table 5.4. This structure yields four FP matrix multiplication results in parallel every clock cycle with a simple unidirectional SA-based control logic. In a standard 2-D SA, Multiplication and Accumulation (MAC) units consist of one FP multiplier and one FP adder operators. Thus our 1-D SA with 4 PEs has the same throughput as a 2-D SA with 4 PEs at the expense of higher resource usage (we instantiate 4 FP multipliers in comparison) which leaves us with simpler control logic and consequentally shortens development time. Although we may have instantiated more PEs in the 1-D SA, the throughput already achieved of this design conforms to our real-time constraints, which will be more detailed in section 5.5.

Functional description with allocated memory buffers of our accelerator will be described in detail in following section.

Table 5.4: Control of *en* signals

en	EKF correction function
00	Phase I eq. 5.5
1X	Phase II eq. 5.5
1X	Phase I eq. 5.7
01	Phase II eq. 5.7

5.3 Cross-covariance matrix block accelerator

Accelerator's hardware architecture is given on Fig. 5.2.

The host processor (dual core ARM CortexA9) performs EKF equations 5.1, 5.2, 5.3, 5.4 and 5.6. Accelerator's main logical component is the *Main sequencer*, a state machine generic to parameters N, r and d. It also pre-buffers the input stream onto PEs during computation of EKF equations 5.5 and 5.7 in order to provide the SA with input data at each clock cycle, maximizing the operational throughput. Each First In First Out (FIFO) structure contains p FIFOs of depth of $ceil((7N + r)/p)$ 32-bit words[2]. P matrix is stored in four tri-port 2W/3R Block Random Access Memories (BRAMs)[3]. It is necessary to make use of tri-port BRAMs, because in equation 5.9 in phase II we have simultaneous read and write operations with different addresses from/to on-chip memory (pseudocode line 4: output of PE_{xmodp} is stored in $P_{x,j}$). As one 1W/1R port is used for the store operation, the multiplexed 1R port is used for fetch operation. The third 1W/1R address line functionality is left for the host processor communication for execution of other equations in EKF block that require parts of P matrix. Cross-covariance matrix P uses in total $((7N + r)(7N + r))2$ bytes. Because it is stored entirely in the on-chip memory, pointer values are pre-set in program memory on each corresponding element in the on-chip memory so the host may access elements in P.

HW functional description. We refer to chronological execution of tasks in an EKF correction loop. From computational model described after eq. 5.10:

1. Phase I. Input sequencer fetches H matrix from input BRAM and loads it onto FIFO1. During HP_4^T computation, H is popped into the buffer, along elements of P_4^T by Main sequencer. Partial product matrix $K^{T'} = [HP_4^T]'$ is pushed onto FIFO2. During inner loop computation (pseudocode line 4) elements of $K^{T'}$ are popped and pushed as: $C_{xmod2,y} = K^{T'}_{xmod2,y}$ and $K^{T'}_{xmod2,y} = PE_{xmod2,y}$ respectively in order to perform $K^{T'}_{k+1} = K^{T'}_k + [HP_4^T]_k$. After d iterations, resulting matrix $K^{T'} = HP_4^T$ is pushed onto FIFO2.

2. Phase II. Input sequencer fetches Z^{-1} matrix from input BRAM and loads it transposed onto FIFO1. Main sequencer buffers the matrices in FIFO1 and FIFO2 and pushes the resulting K^T matrix onto FIFO2, to be used for update of matrix P. Input sequencer stores transposed matrix K from FIFO2 to Input BRAM. It does it in a circular fashion using asynchronous read/write operations on FIFO2, pushing and popping all elements of K^T.

[2]FIFOs are instantiated using Xilinx IP Core Generator v14.2 and physically implemented as dual-port BRAMs.

[3]2W/3R BRAMs are obtained by multiplexing a read port of one of 2W/2R (true dual-port) BRAM's ports

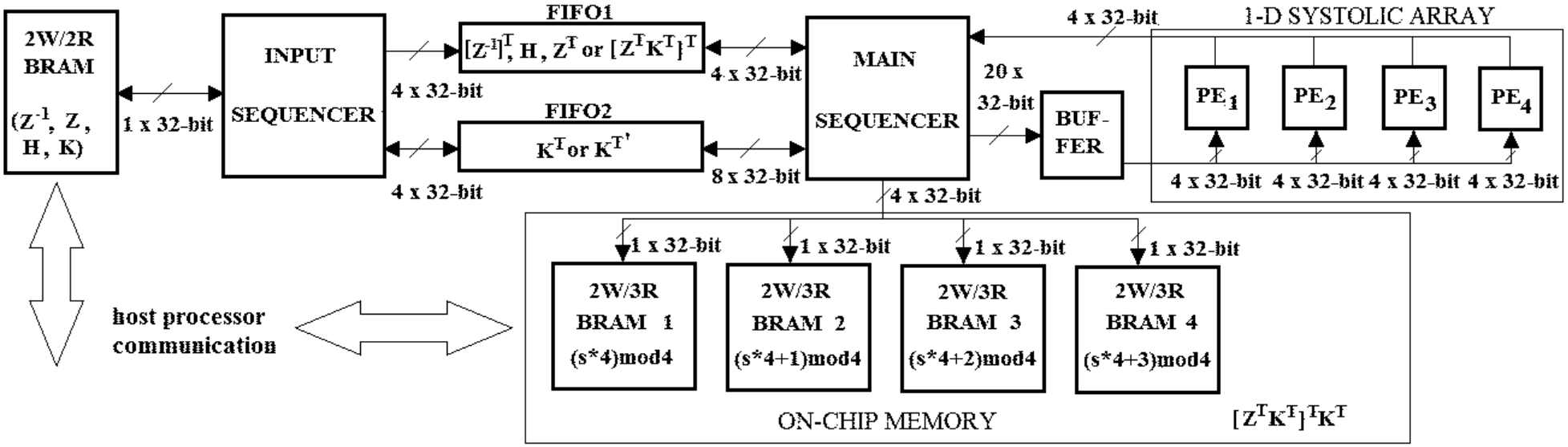

Figure 5.2: Tri-matrix multiplication architecture.

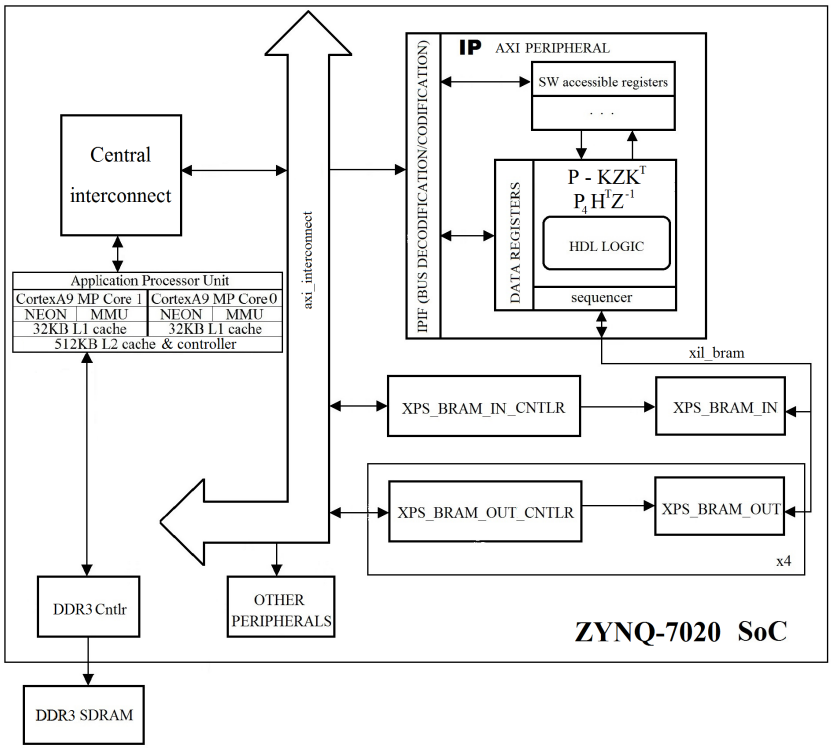

Figure 5.3: Our SoC implementation of the 3D EKF-SLAM application

From computational model described after eq. 5.9:

1. Phase I. Input sequencer fetches Z matrix from input BRAM and loads it transposed onto FIFO1. Z^T and K^T matrices are being popped from FIFO1 and FIFO2, respectively by Main sequencer. While popping, K^T elements are also being pushed in a circular fashion as they are going to be reused later in phase II. Main sequencer pushes the intermediary $[Z^T\ K^T]^T$ matrix in FIFO1.

2. Phase II. $[Z^T\ K^T]^T$ and K^T matrices are fetched from relevant FIFOs and loaded onto the SA. Multiplication result is stored directly into 2W/3R BRAMs.

HW/SW integration. On Figure 5.3 we present the entire Zynq-7020 SoC with our co-processor referred to as Intellectual Property (IP). It is attached as an Advanced eXtensible Interface (AXI) peripheral to a dual-core ARM processor[4].

Communication with host processor unfolds in four simple steps:

1. Host loads matrices H, Z^{-1} and Z onto a fast on-chip memory (input buffer) via *axi_interconnect* bus from the external DDR3 SDRAM and sends a "go" signal to the IP.

2. Co-processor computes equation 5.5 and asserts a "done multiplication" signal.

3. Host fetches K matrix and computes EKF equation 5.6 in parallel with co-processor computing EKF equation 5.7. During phase II, co-processor's main sequencer stores p results into same number of BRAMs, each containing the $(ps+j)modp$ th element,

[4]The system is designed using Xilinx Platform Studio v14.2

where s is the current store cycle count and j is the number of the current row in a $(7N + r)modp$ cycle.

4. Co-processor asserts a "done multiplication" signal. By polling, the host verifies end of the execution of the operations in equation 5.7.

5.3.1 HW/SW Integration

Given that the heterogeneous platform used for co-design of Embedded SLAM prototype is the ZedBoard development board, this accelerator is a memory-mapped AXI peripheral to an ARM Cortex A9 host processor (Figure 4.18).

In order to support this sw/hw communication in Xillinux, first we need to declare global variables for cross-covariance matrix P - $obram$, the memory-mapped co-processor $trimatrix$ and the input memory $ibram$:

```
//config.h
#define N        20
#define MAP_SIZE 7*N+19
#define PE       4
#define EKF      0x80000000
#define oBRAM    8192
#define TRIMAT   1024
#define iBRAM    1024
#define BYTES    (PE*oBRAM+TRIMAT+iBRAM)*4

extern volatile uint* ibram;
extern volatile uint* trimatrix;
extern float** obram;
```

In main program thread we initialize the global variables as:

```
//main.c
...
#include "config.h"
int fd;
volatile uint* ibram;
volatile uint* trimatrix;
float** obram;
...
void init_ekf(){

fd = open("/dev/mem", O_RDWR);
uint map = mmap(0, BYTES, PROT_READ|PROT_WRITE,
                MAP_SHARED, fd, EKF);
obram=(float**)malloc(sizeof(float*)*MAP_SIZE);
int i,j,temp;
// OBRAM INIT
temp = ceil((double)MAP_SIZE/PE);
for(i = 0; i < temp; i++){
```

```
  for(j = 0; j < PE; j++){
    obram[i*PE+j] = map+j*oBRAM+MAP_SIZE*i;
  }
}
// TRIMATRIX INIT
trimatrix = map+PE*oBRAM;
// IBRAM INIT
ibram = map+PE*oBRAM+TRIMAT;
}// end init_ekf{}
...
int main(char* args[]){
...
// sw/hw integration
init_ekf();
float** P = obram;
...
//in correct_lmk()
  // copy matrices Z and K
  memcpy(ibram,&Z,4*sizeof(float));
  memcpy(ibram+4,&K,2*MAP_SIZE*sizeof(float));
  // reset accelerator
  trimatrix[0] = 0x0000ffff;
  // send go
  trimatrix[0] = 0xffff0000;
  trimatrix[0] = 0x0000000;
...
}// end main(char* args[])
```

Software code is written as to support algorithmic and architectural genericity. In order to take into account EKF parameters from equation 5.7, we define preprocessor macros N and MAP_SIZE. Macros PE, $iBRAM$, $oBRAM$ and $TRIMAT$ reflect instantiated accelerator's architecture. Cross-covariance matrix is stored in PE BRAMs, according to number of 32-bit elements output in parallel from accelerator. Therefore, a single output BRAM should contain at least $(int)(((MAP_SIZE \times MAP_SIZE)/PE) + 1)$ 32-bit elements. As in Xilinx XPS we must instantiate memories in chunks of 4096B ($1024 \times$ 4B elemets), we set $oBRAM$ to 8192 when N is 20. Same is the reason for $iBRAM$, as it contains only $Z_{2,2}$ and $K_{7N+19,2}$. Because the accelerator is a memory-mapped device, it has also a minimal address space of $TRIMAT \times 4$ bytes. For integration issues, macro EKF represents the base address of $BYTES$ long memory space of the entire co-processor peripheral on the dedicated AXI bus in Xillinux.

As the accelerator IP is generic itself, this co-design technique may be migrated to bigger reconfigurable devices if there is a demand for more processing power (PEs). Accelerator is also succesfully tested as a PLB peripheral for heterogeneous designs porting an FPGA with IBM's PowerPC440 embedded processor.

Table 5.5: 1-D systolic array resource usage

Resources	**Add**	**Mult**	**Array**	**% IP**
LUTs	254	121	3000	38
DSP48E1s	2	2	32	100
Latency	9	5	14	-
$F_{MAX}[MHz]$	380	368	311	-

5.4 Design evaluation

In this section we give a theoretical estimate on developed multiplier's latency which is later going to be compared with experimental results (in Section 5.5) in order to evaluate the efficiency of the proposed SA-based design in terms of speed of execution. Afterwards we present an overview over resource and power consumption of the entire co-processor on the FPGA.

5.4.1 Multiplier Latency

After the analysis in section 5.2 we have:

$$L > L_0 + L_1 + L_2 \tag{5.12}$$

$$L_1 > max\left(\frac{(d+1)n}{p}, \frac{n}{\sqrt{n}B}\right) \tag{5.13}$$

$$L_2 > max\left(\frac{n^2}{2p}, \frac{n^2}{\sqrt{2n^2}B}\right) \tag{5.14}$$

L_1 latencies correspond to phase I and II multiplications in equation 5.5 and phase I multiplication in equation 5.7. L_2 represents $(Z^T K^T)^T K^T$ matrix multiplication latency. L_0 is the latency due to processor-IP communication. Concerning the matrix multiplication part, we have $M_1 = \sqrt{n}$ and $M_2 = \sqrt{n^2}$ because we have loaded all the corresponding matrices into two FIFO structures so that they could be fetched at each clock cycle. Furthermore, $B = 20$ owing to buffered input data and parallel storing of the results. Thus we are able to deduce that $L_1 > \frac{(d+1)7N+r}{4}$ and $L_2 > \frac{(7N+r)^2}{8}$ as we have $p = 4$ PEs. All delays greater than $L_1 + L_2$ clock cycles are due to L_0 and because of pushing input matrices Z, Z^{-1} and H onto FIFO1 and popping matrix K from FIFO2 by Input sequencer.

5.4.2 Resource usage

In literature, efficient floating point units are implemented with several embedded DSP blocks and Look Up Tables (LUTs). In Table 5.5 an overview over resource usage of the SA-based multiplier versus IP is given. The one-dimensional systolic array consists of 8 deeply pipelined multipliers and 8 adders. In terms of logical resources, the systolic array makes use of 38% of the entire IP while the rest is used by input sequencer, main sequencer, buffer and other internal buffers with 32-bit delay lines. For speed issues, embedded multipliers (DSP48E1s) are instantiated in each PE of the SA.

Table 5.6: Accelerator resource usage with on-chip memory

Resources	**[Tertei et al., 2014]**	**This work**	**Available**
LUTs	7419	7824	53200
DSP48E1s	32	32	220
36kb BRAMs	37	21	140
F_{MAX}[MHz]	198	177	-

In Table 5.6 we show the ratio of resource usage of the accelerator (with instantiated on-chip memory) versus all available resources on Zynq-7020 device. Relevant work [Tertei et al., 2014] is ported from Virtex5 to Zynq device for a proper comparison. Our map containts in total a minimum of $N = 20$ AHP landmarks. A 36k BRAM is used for the input BRAM. Eight 18k BRAMs are used for instantiation of the FIFO structures (4 × 18k per FIFO) and sixteen 36k BRAMs are used to form the four memory banks[5] for storing 51kB of resulting data. Optimal BRAMs consumption in function of matrices cardinalities from Table 5.1 may be calculated as:

$$BRAM36k_no = (M_1 + M_2 + M_3 + M_4) \times 0.5 \tag{5.15}$$

$$M_1 = ceil\left(\frac{|Z| + |Z^{-1}| + |H| + |K|}{512}\right) \tag{5.16}$$

$$M_2 = M_3 = p \times ceil\left(\frac{|K|}{512p}\right) \tag{5.17}$$

$$M_4 = p \times ceil\left(\frac{0.5 \times |P|}{512p}\right) \tag{5.18}$$

where M_1, M_2, M_3 and M_4 is the number of 36kBRAMs that constitute Input BRAM, FIFO1, FIFO2 and P matrix storage, respectively.

We conclude that the accelerator is efficient in terms of resource usage as it consumes only 14.7% (7824 out of 53200 LUTs) of the configurable logic on the FPGA, leaving enough resources for other modules that might be required for the rest of SLAM (acceleration of vision process). BRAM usage (15%) for storing the cross-covariance matrix is minimal due to symmetry-related optimization. The speed issues are going to be detailed in Section 5.5. Comparing to our previous work (Table 5.6), this new version of the accelerator is more efficient in terms of memory usage - from 26% to 15%. Higher consumption of logical resources - from 14% to 14.7% - is due to implementation of EKF equation 5.5 on the Zynq device.

5.4.3 Power consumption

The FPGA power consumption is estimated using Xilinx XPower Analyzer tool - Table 5.7. Most of the power drain is due to dual core ARM processors (which run in low power mode at $667MHz$ dissipating ∼1W). This accelerator uses less power because of lower BRAMs usage for the same application functionality.

[5] A 32-bit 4096 word memory bank is implemented as 4 × 36k BRAM, instantiated in Xilinx Platform Studio v14.2. Each memory bank holds one fourth of the P matrix

Table 5.7: Visual EKF-SLAM accelerator's power distribution: comparison with the previous version

Power consumption [W]	**[Tertei et al., 2014]**	**This work**
Total	1.367	1.302

Table 5.8: EKF equations cycles measured with 125 MHz counter

Equation no°	**[Tertei et al., 2014]**	**This work**	**No accelerator**
5.1	6700	6700	6700
5.2	89000	89000	86600
5.3	507000	507000	500000
5.4	270	270	270
5.5	38800	830	38000
5.6	7500	9400	7500
5.7	11300	3770	220000

5.5 Experimental results

After having synthesized our design, it's maximal operating frequency is 125MHz. Thus its peak throughput is $((p \times 4))FLOP \times 125MHz = 2GFLOPs$.

We differentiate between cases when using older version of the accelerator, the present version and when not using it. Baseline software code executed in all three cases takes into account the P matrix symmetry and $N = c = 20$ setting. Using Gprof we obtain Table 5.8 which show the cycles duration in each EKF block equation. Some equations take longer to execute with acceleration, which is due to the hw/sw co-design; as when using the co-processor, P matrix is stored in on-chip memory in order to leverage the most demanding operations. We then pay the price (though small) by having to transfer the data accros *axi_interconnect* bus. On the other hand, when not using the accelerator, the 51kB matrix is stored in program memory and all P-related operations in correction loop are done by host processor. That consumes a lot more processor time. Thus, it is a profitable trade-off. As in this new version of the accelerator, EKF equations 5.6 and 5.7 are executed in parallel. Thus the latency may be calculated as: $L_{new} = eq.5.1 + eq.5.2 + eq.5.3 + c \times (eq.5.4 + eq.5.5 + max(eq.5.6, eq.5.7))$; while in other two cases it is: $L_{old} = eq.5.1 + eq.5.2 + eq.5.3 + c \times (eq.5.4 + eq.5.5 + eq.5.6 + eq.5.7))$. This new design is thus ∼2.2 times faster than the old one and ∼7.5 times faster than pure software implementation when $N = c = 20$ with $p = 4$ PEs.

5.5.1 Measured multiplication latencies

For twenty landmark observations[6]:

- EKF eq. 5.5. During phase I, Input sequencer fetches H and Z^{-1} matrices in 2×14 and 2×2 clock cycles, respectively. Phase I and II computation take $L_1 \sim 347$ cycles ($d = 7$). After phase II, Input sequencer stores K matrix in 2×159 cycles. Communication latency with host costs $L_0 \sim 830 - 697 = 133$ clock cycles.

[6]cycles count are validated using ChipScope Pro Analyzer

Table 5.9: Device resources usage with $N_{max} = 70$

Resources	**p=2**	**p=4**	**p=6**	**p=8**	**[Tertei et al., 2014] p=4**	**Available**
LUTs	6268	7824	9371	10919	7529	53200
DSP48E1s	16	32	48	64	32	220
36kb BRAMs	131	21	137	138	137	140
F_{MAX}[MHz]	177	177	164	156	180	-

- EKF eq. 5.7. Input sequencer takes 2×2 clock cycles to fetch Z matrix from Input BRAM. The computational latency is $L_1 + L_2 \sim 3752$ clock cycles which leaves us for the host processor-accelerator communication latency $L_0 \sim 3770 - 3756 = 14$ clock cycles.

Computational latency is close to the predicted value in subsection 5.4.1 from which we confirm that the desing performs close to its theoretical optimum.

5.5.2 Design scalability

In general, a VSLAM application could benefit from a larger map size in an unstructured environment. Also, the number of observed landmarks may be higher than $c = 20$. In order to put our work into larger perspective, we scale-up the design for different 1-D SA array sizes: $p = \{2, 4, 6, 8\}$. In Table 5.9 we give the synthesis results for design of different SA sizes. We keep high dimensional landmark parametrization $d = 7$ and $r = 19$ in the robot state vector $dN + r$. So, the presented configurations in the table count for largest possible map sizes N_{max}, limited by number of 36k BRAMs on the Zynq-7020 device. All F_{MAX} are greater than $125MHz$, which means these co-processor designs may be well integrated into the SoC. For comparison, our previous work is also listed. That work permits map sizes of up to 50 landmarks, while the design presented in this thesis allows for 70.

On Figure 5.4 the EKF block throughput is given in function of corrected landmarks when using and when not the co-processor: for smaller maps (containing less than 21 AHP points) all landmarks are observed and corrected; for bigger map sizes there were made only 20 correction loops. On Figure 5.5 we may see the same scheme with the difference that all landmarks in the map are observed and corrected: $c = N$. Depending on the VSLAM application, a designer may choose which one to instantiate.

5.5.3 Power per feature

In Table 5.10 power per feature benchmarking is presented for our 3D EKF SLAM application. It contains 20 landmarks in its map and executes same amount of corrections. For FPGA-based platforms, the developed accelerator with 4 PEs-based 1-D SA multiplication architecture is used. N_{max} is the number of AHP landmarks retained in the SLAM's map that permit $30Hz$ EKF block period.

5.5.4 Future work

Having validated our design, further work encompasses optimizing the resource usage by implementing a 2-D SA along with a more complex control logic as described in

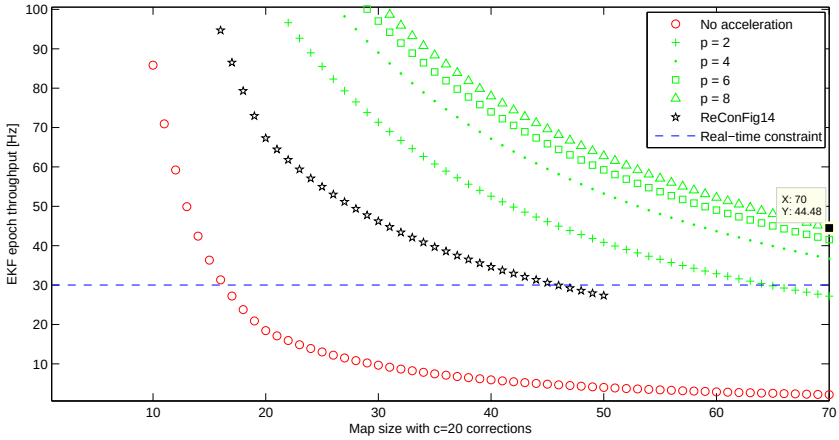

Figure 5.4: EKF block throughput comparison #1

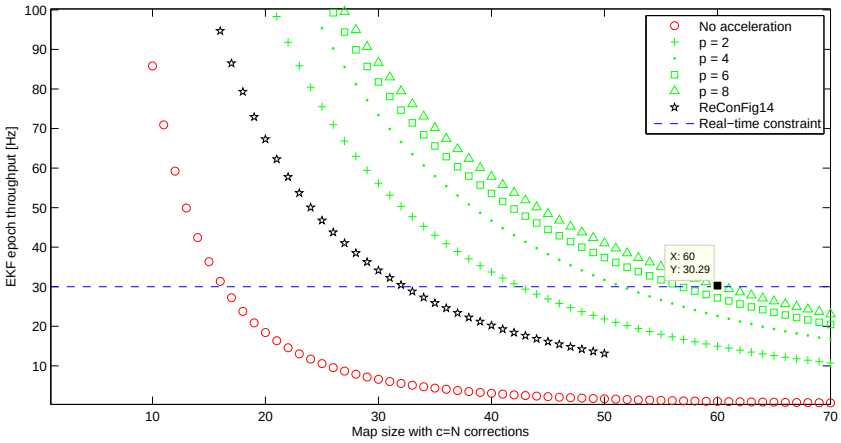

Figure 5.5: EKF block throughput comparison #2

Table 5.10: Power per feature metric for visual EKF-SLAM with AHP points

Hardware platform	N_{max}	**Power [W]**	**Power/feature**
Intel i3 (2 × 1.8 GHz, 4GB RAM)	∼ 100	34	0.34
[Tertei et al., 2014] @ Zynq-7020 SoC (2 × 667 MHz)	45	1.367	0.030
This work: Zynq-7020 SoC (2 × 667 MHz)	70	1.302	0.019

[Castillo-Atoche et al., 2010].

5.6 Conclusion

In this chapter an accelerator for a visual 3D EKF-SLAM application (without loop closure) that uses seven-dimensional landmark parametrization for mobile platform localization is proposed. It is designed as an SoC on a Zynq-7020 device, yielding high performance metrics for our EKF setting: 20 AHP landmarks in the map with all of them being observed and corrected at a rate of at least 30Hz. It is characterised by minimal latency (close to theoretical optimum), low resource usage (14.7% of LUTs and 15% of on-chip memory) and low power consumption of only 1.302W. It provides improved power-per-feature measure of 0.019 mW/AHP with respect to previous work and an order of magnitude improvement when executing the application on a standard personal computer platform (Intel i3 architecture 3217U, taking into account only processor's power consumption). As it consumes small amounts of resources, other accelerators (i.e. for feature detection and/or feature matching) or soft-core processors (i.e. MicroBlaze) may be implemented on the FPGA.

The design is scalable, permitting map sizes of up to 70 AHP landmarks, while correcting 20 out of them or maps with 60 AHP landmarks with all being corrected; all at rates over 30Hz. Scalability of the design allows for adapting this FPGA-based design in unstructured environments where it is potentially required to have larger map sizes with higher amount of landmark observations and corrections for a particular visual EKF SLAM application.

Chapter 6

Embedding Algorithms for Visual Obstacle Detection

Contents

6.1 Related work and motivation 85

6.2 Proposed framework 86

6.2.1 Appearance kernel 87

6.2.2 Shape kernel 87

6.2.3 Classifiers co-design 91

6.2.4 Classifiers training 94

6.2.5 Single scale decision making 96

6.3 Conclusion 100

Efficient navigation of an intelligent mobile vehicle in a congested unstructured environment relies heavily on accurate obstacle detection. With the advent of high-resolution imaging sensors providing rich contextual information, there is high demand for development of mountable image processing hardware and software which are in their turn restrained in terms of power consumption, processing power and on-chip memory. While limited in physical resources, these embedded systems are required to achieve acceptable precision/recall metrics in real-time in comparison with their state-of-the-art performing software counterparts executed on standard personal computers.

This chapter presents a suitable framework for monocular embedded obstacle detection. The developed approach is tested on a road sequence taken from the KITTI public data set. Results, presented and discussed in next chapter, show satisfying performance in detection metrics over vehicle detection and real-time execution on ZedBoard-based embedded ecosystem.

6.1 Related work and motivation

Image indexation presented in [Harzallah et al., 2009] treats obstacles from a holistic point of view: from a sliding detection window an image patch is cropped out and put into a two-stage hierarchical classification scheme. The first stage uses a weak and computationally inexpensive binary classifier to reject the majority of the negative samples. The positive candidates are passed to a second stage classification, where a computationally expensive classifier attributes the image patch a score. This approach is repeated on detection windows of multiple sizes (multi-scale decision making) and scores are compared. Non-maximal suppression based on greedy search is used to discard false positives. Despite being very performing[1], this approach as-is is not embeddable because of huge runtime memory requirements and heavy computational load. Its advantages, on the other hand, lie in no assumptions being made about object's location (using exhaustive sliding window approach) and algorithmic genericity for detection of multiple obstacle classes.

In this chapter I propose a monocular, knowledge- and appearance-based framework for building efficient embedded obstacle detection systems. Targeted hardware platforms are emerging multi-core heterogeneous[2] boards equipped with an exteroceptive sensor - a high resolution camera. I intend to port the essence of the described approach into embedded domain using parallelization techniques, rule-based binary classification, computational optimizations for classification and single-scale detection window based decision making for accurate hypothesis verification.

Classifier performance will be evaluated based on metrics most widely used in object detection-related literature. They may be classified into two schemes:

1. Per Window (PW) and
2. Full Image (FI).

The first method determines performance based on cropped positive and negative image windows. This approach puts forward unitary classifier performance from overall detection system, thus making it ideal for evaluating classifier performance in a hierarchical

[1] won the 2007 Pascal VOC challenge
[2] multi-core CPU + FPGA

classification or boosting scheme [Dollar et al., 2012]. The FI method relies on whole input image evaluation where objects are searched for in bounded boxes. It makes an assumption that the classifier under evaluation performs multi-scale detection and appropriate non-maximal suppression (NMS). Hence, this scheme is most suitable for evaluating a complete detection system.

6.2 Proposed framework

The essence of the proposed approach in [Harzallah et al., 2009] is to combine the localization and shape characteristics in a single kernel, thus building a higher-dimensional kernel $K(u, v)$ which is a convolution of HoG- and BoW- kernels by multiplication:

$$K(u, v) = K_{HoG}(u_{HoG}, v_{HoG}) \times K_{BoW}(u_{BoW}, v_{BoW}) \tag{6.1}$$

However algorithmically well-performing, it requires a huge computational effort which makes it difficult for embedding. Instead of augmenting the dimensionality of the problem, shape- and appearance-related information may be encoded by a proper kernel (HoG and BoW) and combined in a hierarchical boosting framework with stages:

- Object localization[3] based on interest point characterization (appearance)
- Object classification based on gradient representation (shape)
- Decision making - NMS

which is more suitable for embedded domain. In literature, when it comes to generic object recognition, most works make use of sliding windows technique (or its workarounds) in order to perform foreground estimation at multiple scales (equation 6.2):

$$S_{obj} = argmax_{S \subseteq I} f(S) \tag{6.2}$$

where S is the subset of all rectangles of defined scales (either given as an aspect ratio $n : m$ or as $[height, width]$) in an image I. After hypothesis verification, the decision making stage implements a *non-maximal suppression* technique on S in order to discard the false positive detections, based on a confidence score after some quality function evaluation $f(S)$. Its main drawback lies in algorithmic complexity of $\mathcal{O}(n^4)$ for a given image of dimension $[height, width] = [n, n]$ in a standard multi-scale decision making scheme.

Another hypothesis being made in this thesis is the single-scale decision making, where the search over I is performed with a fixed-sized detection window $s \in S_{obj}$ in order to alleviate the algorithmic complexity: $\mathcal{O}(n^4) \rightarrow \mathcal{O}(n^2)$. Based on detection scores after stage II, windows are either concatenated or discarded based on assumptions which are going to be explained in more detail in the subsection 6.2.5. In following subsections each stage is going to be explained in more detail.

[3] also known as foreground estimation

6.2.1 Appearance kernel

Interest point characterization. In the literature, a known robust feature detector and descriptor is SIFT [Lowe, 2004]. Authors in [Mikolajczyk and Schmid, 2005b] show its high repeatability rate and invariance to affine image transformations. A more recent study of interest points characterization conducted on KITTI dataset may be found in this thesis - [Li, 2013]. From these algorithmic performance comparisons, it may be concluded that SIFT characterization, among others, is performing well. It is the same feature description algorithm used in [Harzallah et al., 2009] and it will be used for proof-of-concept of algorithmic performance of the proposed object detector in this thesis. Main reason behind choosing SIFT is because this transform identifies well reproducible keypoints between consequent frames in a streaming application, which is an essential criteria in crowded scenery when having to track obstacles by detection.

As the first stage kernel relies on presence of feature keypoints in a given fixed-size bounding box s, a BoW histogram will be used to encode this information. After [Harzallah et al., 2009], a visual vocabulary C of representative 100 SIFT keypoints will be built offline based on k-means clustering of all positive object samples in KITTI training set (from cropped images). Afterwards, during hypothesis generation, the dimensionality of the problem may be scaled to a mapping function m_{BoW}:

$$s \xrightarrow{SIFT} \mathbb{R}^{K\times 128} \xrightarrow{||C-KP||^2} \mathbb{R}^{1\times 100} \tag{6.3}$$

where KP is the set of found SIFT keypoints, $K = |KP|$ and the number 128 stands for 128-dimensional[4] SIFT keypoints in a given fixed-size detection window. In other words, the object appearance represented as a collection of local features KP in a detection window s is encoded into a BoW histogram of fixed size. During norm computation, BoW histogram is computed based on Euclidean distance measure with each word in visual vocabulary C. During histogram calculation, a candidate point is considered ambiguous if its distance to the nearest visual word over the distance to the second nearest is less than 0.8 (Lowe's criteria [Lowe, 2004]).

6.2.2 Shape kernel

Gradient-based characterization. Histogram of Gradients (HoG) model may be useful in object representation because of its invariant properties related to:

- illumination,
- contrast and
- affine transforms.

Computing the intensity difference in a color image by 2D (horizontal and vertical) first order derivatives eliminates small changes in illumination and contrast (i.e. due to imager noise or abrupt environmental changes). Furthermore, gradient computation captures the intensity transition around objects' boundaries, which results in information retrieval about it's structure. The pooling in the form of a histogram renders these features robust with respect to small shifts, ameliorating deformation tolerance.

[4]each dimension is 4B floating-point value

Popular gradient-based descriptor used mostly for pedestrian detection is HoG descriptor, introduced by [Dalal and Triggs, 2005]. Here, an entire object is represented by a single feature vector as explained in 3.1.1.4. As cell dimension, number of cells within a block, number of blocks and binning are all empirical values, a study on these variables on a given object class needs to be performed:

Algorithm 1 Classification based empirical study on HoG parameters

Require: $N_SET, S_{obj}, B, C, D, O, object$
1: Initialize $X, |X| = |B| \times |C| \times |D| \times |O|$
2: Initialize $EER(s, x) = 0$, for all $s \in S_{obj}$ and $x \in X$
3: $[train, test] = prepare_dataset(N_SET, object)$
4: **for each** $s \in S_{obj}$ **do**
5: $\quad [train', test'] = resize(train, test, s)$
6: $\quad$ **for each** $b \in B$ **do**
7: $\quad\quad$ **for each** $c \in C$ **do**
8: $\quad\quad\quad$ **for each** $d \in D$ **do**
9: $\quad\quad\quad\quad$ **for each** $o \in O$ **do**
10: $\quad\quad\quad\quad\quad [train_HoG, test_HoG] = calculate_HoG(train', test', x)$
11: $\quad\quad\quad\quad\quad svm_tmp = svmtrain(train_HoG)$
12: $\quad\quad\quad\quad\quad svm_pred = svmclassify(test_HoG, svm_tmp)$
13: $\quad\quad\quad\quad\quad [TPR, TNR, EER(s, x)] = vl_roc(svm_pred, test_HoG)$
14: $\quad\quad\quad\quad$ **end for**
15: $\quad\quad\quad$ **end for**
16: $\quad\quad$ **end for**
17: $\quad$ **end for**
18: **end for**
Ensure: $EER(s^*, x^*) = min(EER(s, x))|\{b^*, c^*, d^*, o^*\} \in x^*$

This study aims at finding minimal equal error rate (EER) metric of a given classifier in a PW scheme. It measures the classifier performance against an another classifier's (the lower the better) by identifying the point on the DET (detection error trade-off) curve that has the same False Alarm Rate (FAR) and False Reject Rate (FRR, or missing rate). Input sets are:

- N_SET, the number of frames containing at least one whole-sized instance of defined object class. These images are indexed from KITTI training set which counts a maximum of 7481 annotated frames (using Amazon's Mechanical Turk [Turk, 2012]).
- S_obj, a finite set of $[width, height]$ scalar pairs which define the detection window dimension.
- B, C, D, O - finite sets which define HoG parameters $\{block_size\}$, $\{cell_size\}$, $\{binning\}$ and $\{orientation\}$ respectively.
- $object$, the object class.

Upon B, C, D, O, a new ordered set X is defined where element x_i represents a given HoG parameter configuration. EER measure is given in matrix form, where each row contains classifier performance for a given fixed-size detection window. Values related to

Table 6.1: Histogram of Gradients: an empirical study

Set	Values
N_SET	{100, 200, 300}
S_{obj}	{[120,60],[128,64],[136,68],[144,72],[152,76]}
B	{[12,12],[10,10],[8,8],[6,6],[4,4]}
C	{[4,4],[3,3],[2,2],[1,1]}
D	{9,10,11,12}
O	{0,1}
$object$	{'Car'}
$\|samples\|$	{1048, 2670, 3299}

HoG parameters are based on *baseline HoG* paper [Dalal and Triggs, 2005] and are varied in this study. Function descriptions are as follow:

- $prepare_dataset()$: for a given object class and number of frames, cropped images are extracted from KITTI training set as positive samples. In order to augment the positive sample base, mirroring technique is used. Negative samples are extracted randomly from same frames from non-object image locations in 5 : 1 ratio with positive samples. Afterwards, the *train* set is constituted by random sampling of 80% of both positive and negative samples, leaving the rest to *test* set.

- $resize()$: cropped images are resized (using bicubic interpolation) to fit the detection window size s.

- $calculate_HoG()$: HoG descriptor calculation based on x parameters configuration.

- $svmtrain()$: Matlab's function for training a linear SVM classifier with default values, using the Sequential Minimal Optimization (SMO) [Platt et al., 1998] algorithm.

- $svmclassify()$: Matlab's function for generating an SVM response.

- $vl_roc()$: an open source VLFeat's [Vedaldi and Fulkerson, 2010] function for generating True Positive Rate (TPR) and True Negative Rate (TNR) metrics, used for display of Receiver Operating Curve (ROC) along with EER metric.

The varying parameter varied in this study are presented in Table 6.1. The $|samples|$ row stands for set cardinality when $N_SET = 100$, $N_SET = 200$ and $N_SET = 300$, respectively. The lowest EER was obtained with $N_SET = 300$. Results of that study may be seen on Figure 6.1: Best parameter configuration are tuples $x^* = [1x1, 4x4, 11, 1]$ and $s^* = [152, 76]$, with $EER = 0.0376$. From this study it may be deduced that for vehicle detection, HoG descriptor works best on larger image patches (in comparison to those used for pedestrian detection) that are divided into a grid of preferably small blocks which in their turn contain small number of cells. These cells map a small number of neighboring pixels. Higher binning value (oriented binning) suggests that HoG works best with larger histograms for vehicle class.

Size of the HoG descriptor may be calculated as:

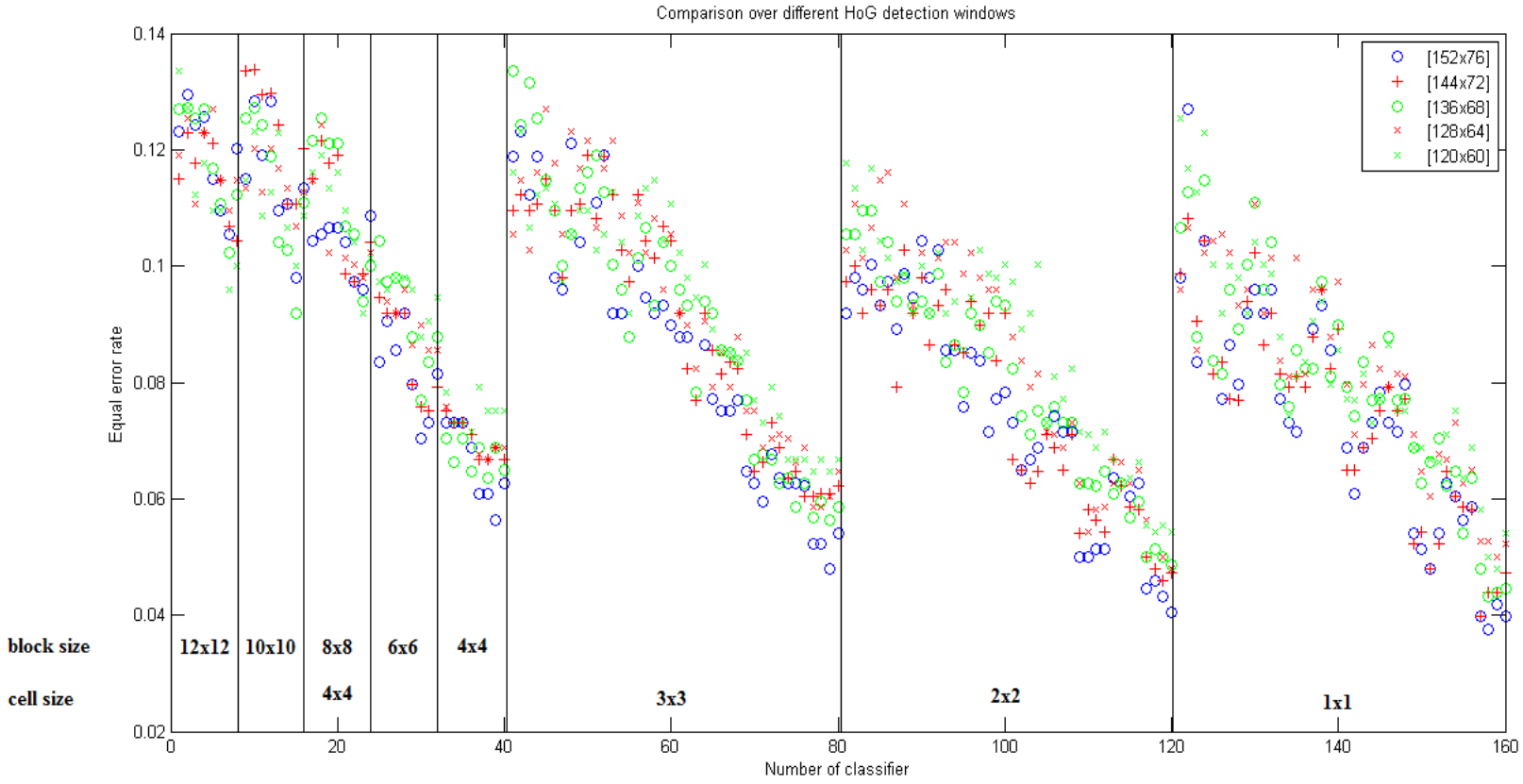

Figure 6.1: Equal error rate of a set of classifiers for $N_SET = 300$

$$hist_1 = 2 + ceil(-0.5 + \frac{s_1^*}{c^*}) \tag{6.4}$$

$$hist_2 = 2 + ceil(-0.5 + \frac{s_0^*}{c^*}) \tag{6.5}$$

$$HoG_size = (hist_1 - 2 - (b^* - 1)) * (hist_2 - 2 - (b^* - 1)) * d^* * b^{*^2} \tag{6.6}$$

which gives the dimension of the shape kernel as:

$$s \xrightarrow{HoG(x^*)} \mathbb{R}^{7942}. \tag{6.7}$$

6.2.3 Classifiers co-design

Having previously defined object representation kernels and detection window size s^* for a particular object class *Car*, a methodology for training proper classifiers with respect to overall framework will be discussed in this section. Co-design metrics cover algorithmic performance and application latency, the latter being directly conditioned by kernel dimensionality for a given obstacle class.
As K_{HoG} exhibits higher dimensionality, it will be used for hypothesis validation *Stage I* on object's localization in the image. Hypothesis verification (*Stage I*) will thus be performed with K_{BoW}. The detection pipeline relies on high precision and recall of second-stage classifier. In a hierarchical setup, stage n has to provide higher recall than precision during object identification which implies discarding as many false positive samples as possible. At the same time, in order that such a pipeline is executable on a physical embedded system, it has to provide lower executional latency than stage $n + 1$. Finally, the last classification stage should have both high precision and recall.

First stage classifier. As we have two stages of classification, appearance-related reasoning accommodates to a fact of building a weak, preferably biased over overfitted, first-level classifier. Also, the latency in *StageI* may be significantly reduced if using a linear SVM kernel for classification. Because of these issues the associated $SVM(K_{BoW})$ will not make use of a computationally expensive nonlinear kernel trick function. However, the problem complexity defined by appearance kernel K_{BoW} sometimes may not be learned without a nonlinear transformation. For these reasons I propose a rule-based algorithm for linear kernel SVMs training:

where:

- *positives* and *negatives* are sets of histograms representing positive and negative K_{BoW} samples respectively.
- *low* and *high* are lower and upper boundaries on number of non-negative elements in a histogram sample, upon which the problem space is downsized
- *thresh* is a threshold value used for admissible amount of misclassified samples after a linear SVM training in a crossvalidation scheme
- *counter* is a temporary variable used for limitting the training iterations of new SVMs if the problem is linearly separable but does not produce a well-generalizing classifier in a crossvalidation scheme

Algorithm 2 Rule-based linear kernel SVM training

Require: $positives, negatives$

Initialize $low = 10, thresh = 1$
Initialize $P_{min} = min(nnz(positives)), P_{max} = max(nnz(positives))$
while $low\ != P_{max}$ **do**
    $high = P_{max}$
    $counter = 0$
    while $thresh > 0.1$ **do**
        $positives^{'} \subseteq positives \mid low < nnz(positives) \leq high$
        $negatives^{'} \subseteq negatives \mid low < nnz(negatives) \leq high$
        $[train, test] = crosstrain(positives^{'}, negatives^{'})$
        try
            $SVM_{low_high} = svmtrain(train)$
            $counter++$
            $pred = svmclassify(SVM_{low_high}, test)$
            $thresh = misclass(pred, test)$
            $counter == 100\ ?\ high--, counter = 0 :$ **continue**
        catch
            $high--$
    end while
    $save_svm(SVM_{low_high})$
    $low = high$
end while

Function descriptions are the following:

- $nnz()$ counts the number of non-zero elements in a set of histograms
- $crosstrain()$ shuffles randomly the input sets and divides them into *train* and *test* subsets in 5:1 ratio respectively
- $misclass()$ builds a confusion matrix upon predicted and true values and returns a missclassification measure defined as:

$$\frac{FP + FN}{TP + FP + TN + FN} \tag{6.8}$$

- $save_svm()$ saves the cross-validated SVM with the information on lower (low) and upper ($high$) boundaries of the solved problem to be used later during prediction in the detection cascade

Main reasoning behind this algorithm is to downsize the training set following a criterium of non-negative elements in K_{BoW} histograms. By doing so, the training problem gets simplified and becomes prone to linear hyperplane separation. It is only natural to consider an object's appearance in a detection window s when having histograms with rich information i.e. those with large number of non-zero elements. Intuitively, regions in image that have a uniform texture and thus low number of extracted keypoints are more likely to belong to background. Simplified block-diagram of rule-based reasoning for first stage classifier may be seen on Figure 6.2:

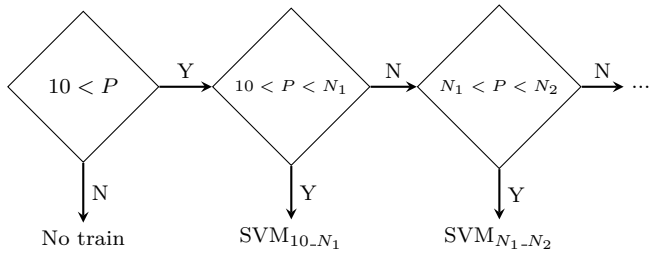

Figure 6.2: Rule-based BoW classification.

Second stage classifier. For a second-stage classifier a PW scheme based study is conducted whether to use or not a Gaussian for kernel trick function. SVM regularization parameter C influences the margin between an SVM classifier's separating hyperplane to input data (positively and negatively categorized m-dimensional samples) in terms of m-dimensional Euclidean distance. A large boundary is caused by high regularization parameter values which induces a high-variance classifier. Very large margin SVMs return minimal training error but do not generalize well on testing data (problem of overshooting). In contrary to high-variance, an SVM classifier with small margin is biased. It has high training error and also generalizes bad on testing data. This study thus evaluates adequacy of using a nonlinear or linear SVM kernel with or without HoG descriptor on vehicle image patches from KITTI dataset as described in Algorithm 3.

Algorithm 3 Classification based empirical study on SVM training parameters

Require: $N_SET, S_{obj}, x^*, P, object$
1: Initialize $Y, |Y| = 2$
2: Initialize $EER_{0:3}(s, y) = 0$, for all $s \in S_{obj}$ and $y \in Y$
3: $[train, test] = prepare_dataset(N_SET, object)$
4: **for each** $s \in S_{obj}$ **do**
5: $\quad [train', test'] = resize(train, test, s)$
6: $\quad$ **for each** $\sigma \in P$ **do**
7: $\quad\quad$ **for each** $C \in P$ **do**
8: $\quad\quad\quad [train_HoG, test_HoG] = calculate_HoG(train', test', x^*)$
9: $\quad\quad\quad svm_tmp_0 = svmtrain(train', C)$
10: $\quad\quad\quad svm_tmp_1 = svmtrain(train', C, \sigma)$
11: $\quad\quad\quad svm_tmp_2 = svmtrain(train_HoG, C)$
12: $\quad\quad\quad svm_tmp_3 = svmtrain(train_HoG, C, \sigma)$
13: $\quad\quad\quad svm_pred_0 = svmclassify(test', svm_tmp_0)$
14: $\quad\quad\quad svm_pred_1 = svmclassify(test', svm_tmp_1)$
15: $\quad\quad\quad svm_pred_2 = svmclassify(test_HoG, svm_tmp_2)$
16: $\quad\quad\quad svm_pred_3 = svmclassify(test_HoG, svm_tmp_3)$
17: $\quad\quad\quad [TPR_{0:1}, TNR_{0:1}, EER_{0:1}(s, y)] = vl_roc(svm_pred_{0:1}, test')$
18: $\quad\quad\quad [TPR_{2:3}, TNR_{2:3}, EER_{2:3}(s, y)] = vl_roc(svm_pred_{2:3}, test_HoG)$
19: $\quad\quad$ **end for**
20: $\quad$ **end for**
21: $\quad EER(s, y^*) = min(EER(s, y))|\{\sigma^*, C^*\} \in P$
22: **end for**

Table 6.2: SVM kernels: an empirical study

Set	Values
N_SET	{100, 120, 200}
S_{obj}	$\{h, w\} \mid h = 375 * (0.2 : 0.1 : 0.8), w = h * (0.5 : 0.1 : 2)$
P	{ 0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30 }
$object$	{'Car'}
$\lvert samples \rvert$	{951, 1187, 2912}

It's goal is to determine whether the performance of classification improves significantly when using a Gaussian kernel. This kernel trick is more computationaly expensive than a linear kernel (dot product) and computational latency at $StageII$ may be significantly reduced in case the Gaussian function does not contribute much in algorithmic performance. Classifiers with minimal EER value over a given detection window size s^* are chosen and their regularization parameter value is plotted on Figure 6.3. Experimental setup used in this study is given in Table 6.2. The difference from previous experimental setup described in subsection 6.2.2 is only with following input sets:

- x^* : the resulting HoG descriptor parameters from previous study and
- P : SVM training parameter values. For nonlinear Gaussian kernel function, higher values of σ^2 induce higher bias and lower values induce higher variance.

From Figure 6.3 it may be seen that C values are lower when using an SVM with HoG descriptor for vehicle patches. Secondly, in general, regularization parameter is lower for classifiers with linear kernels in comparison to classifiers with Gaussian kernels. Overal EER measure for Gaussian-based classification is $\sim$0.11 while for linear kernel based is $\sim$0.14.

The conclusion drawn is that once having applied a HoG shape-based kernel, the problem space may be better linearly separable, having in mind C values fluctuation after this study for vehicle obstacle class on KITTI dataset. In consequence, it is a worthy trade-off to chose a linear SVM over a nonlinear (Gaussian) given the small amount of performance increase it provides for the classifier in contrast to computational speed-up.

6.2.4 Classifiers training

After having defined whether to use or not kernel trick functions, in this subsection the technique for offline dataset extraction upon which final SVMs are trained will be discussed in short. As the second stage classifier performs performance scoring, samples extracted here with respect to this classifier are also going to be passed as *positives* and *negatives* (see Algorithm 2) sets to first stage classifier.

Dataset extraction makes use of *bootstrapping* technique in the PI scheme: from a number of N_SET image frames an **initial training set** is built. Positive samples, based on ground truth KITTI annotations and negative samples, from random locations in background scenery, are extracted in ratio 8:1 respectively. High positives-to-negatives ratio contributes to building an initially "optimistic" K_{HoG} classifier, meaning it is more prone to predicting positive values, thus having small precision and high recall. Once

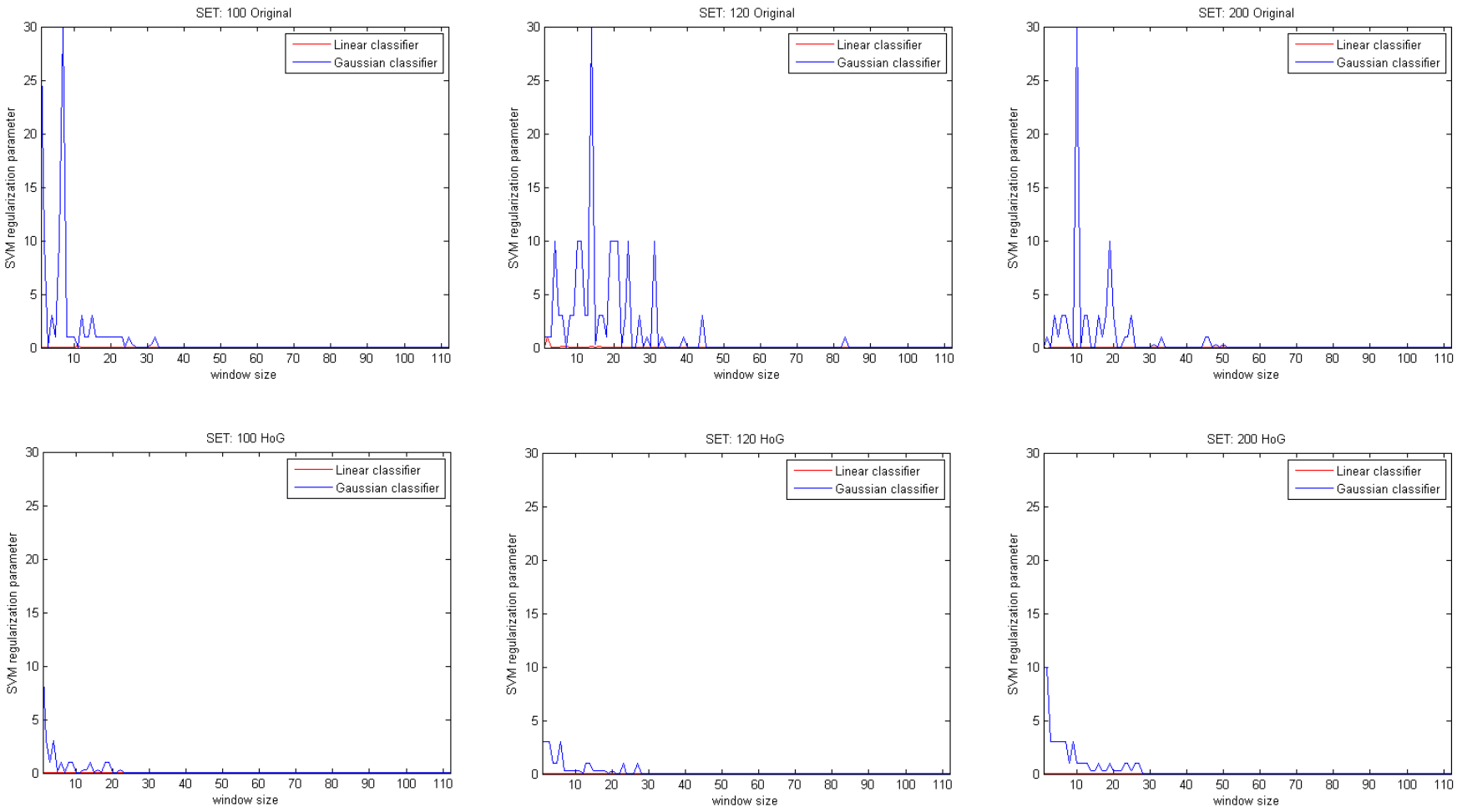

Figure 6.3: Learning curves for $SVM(K_{raw})$ and $SVM(K_{HoG})$

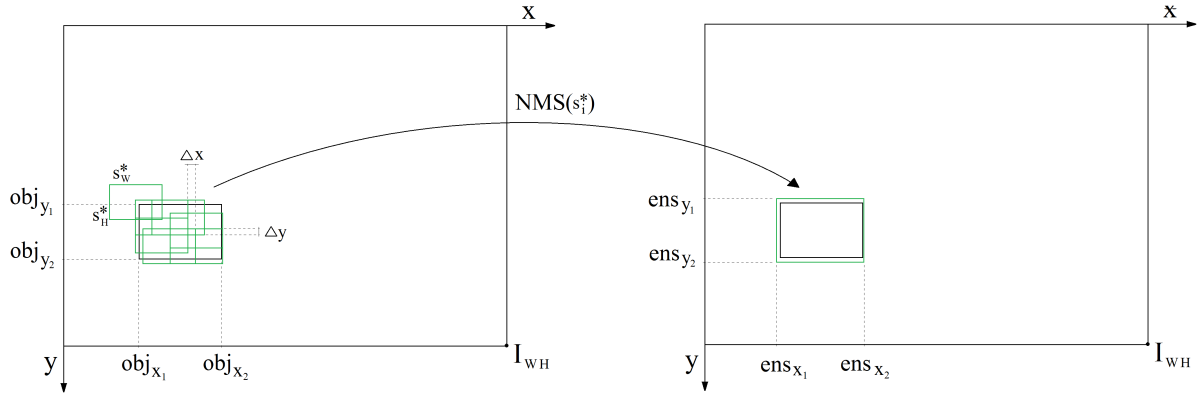

Figure 6.4: Single scale detection window non-maximal suppression scheme

the classifier is trained, it is applied to new unseen N_SET images in a sliding windows-based evaluation on the detection window size s^*. All misclassified instances are added to **hard training set**. First bootstrapping iteration ends by building a new cross-validated classifier on the updated training set which is now a union set of the initial and hard examples sets. In same manner, $n-1$ more iterations are performed, each contributing to building a classifier with higher precision than before.

6.2.5 Single scale decision making

Third stage of the hierarchical object detection framework has for its goal to perform non-maximal suppression (NMS) based on responses from linear HoG-SVM classifier - Figure 6.4. In this figure, real location of the object *obj* within image I is given in image coordinates by $[obj_{x_1}, obj_{y_1}, obj_{x_2}, obj_{y_2}]$. The matrix of sliding windows based responses, defined as *scoremap*, is built after stage II. Its dimension is conditioned by horizontal stride Δx and vertical stride Δy as:

$$scoremapX = floor\left(\frac{I_W}{\Delta x}\right) - floor\left(\frac{s_W^*}{\Delta x}\right) + 1 \tag{6.9}$$

$$scoremapY = floor\left(\frac{I_H}{\Delta y}\right) - floor\left(\frac{s_H^*}{\Delta y}\right) + 1 \tag{6.10}$$

Task of NMS is to infer an ensemble *ens* of overlapping detection windows s^* by means of windows concatenation and supression logic that maximizes the Jaccard index defined as:

$$J(obj, ens) = \frac{obj \cap ens}{obj \cup ens} \tag{6.11}$$

In the scope of implementing this NMS functionality in a single-scale search space, two sequential tasks have to be solved:

1. Obstacle localization
2. Final detection window size adjustment

As we are dealing with model-based object classification, NMS may be used in two different scenarios: for image indexation (content retrieval) where the location of an object

given by its template has to be identified or in a more general object recognition scheme where an object may appear at different scales and viewpoints. For first application, we propose using subtractive clustering [Yager and Filev, 1994] to solve the NMS problem. On Figure 6.5 proof-of-concept detection pipeline on two template-based obstacle instances may be seen. Second stage SVM classifier responses are binarized after applying a threshold. Subtractive clustering regroups binary SVM responses based on a predefined radius r, yielding close to optimal responses. Threshold and radius are empirical values tuned after applying a PI based cross-validation scheme. However, this thesis focuses more on general object recognition problem. Here I introduce $cumulative functions$ for obstacle localization and a $probabilistic model$ reasoning for ensemble dimension adjustment - Algorithm 4.

Algorithm 4 Cumulative functions based single scale non-maximal suppression

Require: $scoremap, s^*, \Delta x, \Delta y, t_{0:3}$
Initialize $H, |H| = scoremapX$
Initialize $V, |V| = scoremapY$
Initialize $obstacles, obstacles \in \{\emptyset\}$
Initialize $ensembles, ensembles \in \{\emptyset\}$
$H = sum_columns(scoremap)$
if $max(H) > t_0$ **then**
  $x_1 = 0$
  $x_2 = 0$
  for $x = 0 : scoremapX - 1$ **do**
    if $x > x_2$ **then**
      if $H(x) > t_0$ **then**
        $x_1 = x$
        $x_2 = x$
        $x++$
        while $H(x) > t_0$ **do**
          $x_2 = x$
          $x++$
        end while
        $V = sum_rows(scoremap(:, x_1 : x_2))$
        if $max(V) > t_1$ **then**
          $y_1 = 0$
          $y_2 = 0$
          for $y = 0 : scoremapY - 1$ **do**
            if $y > y_2$ **then**
              if $V(y) > t_1$ **then**
                $y_1 = \mathrm{y}$
                $y_2 = \mathrm{y}$
                while $V(y) > t_1$ **do**
                $y_2 = y$
                $y++$
                end while
                $confidence_matrix = scoremap(y_1 : y_2, x_1 : x_2)$
                $confidence_score = \frac{sum(confidence_matrix)}{nnz(confidence_matrix)}$
                if $confidence_score > t_2$ **then**
                $X_1 = x_1 * \Delta x$
                $X_2 = x_2 * \Delta x + s^*_W$
                $Y_1 = y_1 * \Delta y$
                $Y_2 = y_2 * \Delta y + s^*_H$
                $obstacles.append([X_1, Y_1, X_2, Y_2, confidence_score])$
                end if
              end if
            end if
          end for
        end if
      end if
    end if
  end for
end if
Ensure: $ensembles = depth_first_search(obstacles, t_3)$

Figure 6.5: Subtractive clustering in a template-based NMS inferring

Cumulative H and V functions are discrete functions calculated as sums along columns and rows of the scoremap matrix, respectively. Identifying continous sections in H and V above threshold values t_0 and t_1 results in assumptions on obstacle boundaries, given that we have good HOG-SVM performance. However, these assumptions are corroborated iff scores of overlapping detection windows satisfy the condition on line 34 in Algorithm 4, a threshold value t_2 defined in a Bayesian scheme. In machine learning, besides discriminative models (such as an SVM), generative probabilistic models attain very good performance as they infer upon model statistics. A basic representative - a Naïve Bayes classifier - tends to maximize the equation (as summarized in [Bradski and Kaehler, 2008]):

$$P(object \mid features_{0:n-1}) = P(object) \times \prod_{i=0}^{n-1} P(features_i \mid object) \tag{6.12}$$

in a model-based classification scheme. Probability of detecting an object in an image depends on *prior probability* $P(object)$ and detected constituant features (*likelihood* the product part on the right hand part of the equation 6.12). In our case, as we work with regressional models, we define a confidence score $P(object \mid s_{0:n-1})$ upon obstacle boundaries assumptions similarly as in Bayesian scheme:

$$P(object \mid s_{0:n-1}) = \frac{1}{n} \sum_{i=0}^{n-1} f(s_i \mid object) \tag{6.13}$$

as here $P(object)$ is implied given the use of sliding windows approach. Furthermore, threshold t_2 is thus given by:

$$T = argmax(pdf(\{f(test_positives)\})) \tag{6.14}$$

where *pdf* is a probability density function built upon SVM-HoG predictions in a PI evaluation in crossvalidation scheme for its random variable. The threshold $t_2 = T$ is thus its mode measure, giving us an intuition over acceptable confidence score values for windows concatenation/rejection. Also it is good starting point for tuning threshold values $t_{0:2}$. As objects that are considerably bigger than the detection window size tend to generate multiple overlapping (already) adjusted detection boxes, a *depth first search* algorithm additionaly concatenates all which overlap more than a given threshold t_3. Usually it is set to 10%. That is the final adjustment for definitive response from our hierarchical object detection system.

6.3 Conclusion

The summary of the proposed approach may be seen on Figure 6.6:

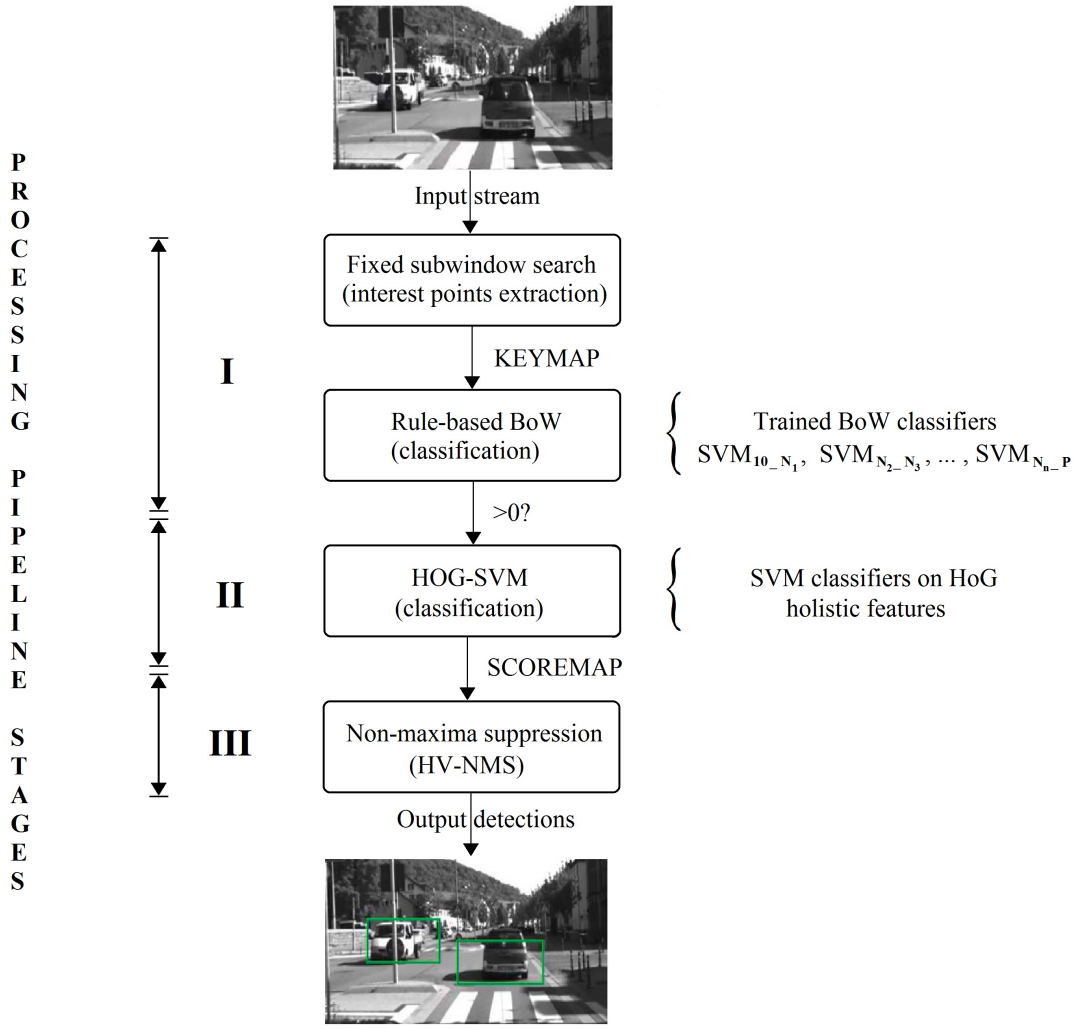

Figure 6.6: Block diagram of the proposed framework.

The hypothesis being made in this chapter is that the information relative to the presence of the object, searched for in a detection window of a fixed size, can be described by using a robust interest point detector with a region descriptor. The encoded information is then put into a Bag ofWord (BoW) representation. This vector-histogram is classified using a linear Support Vector Machine (SVM) classifier (end of *Stage I*). If the first stage classification yields a positive value, second stage verifies the shape of the obstacle candidate using a tuned Histogram of Gradients descriptor with its proper linear SVM. If the supposition is validated, a positive confidence score is assigned to the bounding box location (end of *Stage II*). The third stage reasons on obtained scores and performs non-maxima suppression. Remaining positive candidate windows are concatenated if there is a significant overlap between them (end of *Stage III*). After the final decision has been made, an obstacle is considered to be within its adjusted bounding box dimensions. The whole obstacle detection by identification pipeline is run on a frame by frame basis without making any data correspondences between frames.

Chapter 7

Towards Real-Time Visual Vehicle Detection

Contents

7.1 Software architecture 102

7.1.1 Concurrence 104

7.2 Algorithmic Evaluation 105

7.2.1 Estimation of vehicles detection on an embedded platform . . . 108

7.3 Conclusion 110

Migrating object recognition algorithms on systems of limited size, such as MAVs and intelligent vehicles, is most often done by developing dedicated hardware accelerators on FPGAs [Advani et al., 2015], [Chang et al., 2013], [Wilson et al., 2014], [Hahnle et al., 2013]. Their convenient size, powerful parallel processing capabilities and low power drain (∼5W vs ∼300W for an average GPU) allow for efficient embedded systems development. Nowadays, the embedded industry is shifting towards issuing heterogeneous hardware architectures which port besides an FPGA (which already comes with a multi-core hard processor such as ARMv7) also an embdedded GPU [Xilinx, 2015]. In this chapter, given the maturity of the FPGA-based keypoint detector accelerators, an efficient software architecture will be proposed for real-time vehicles detection that is to be executed on a multi-core hardware platform. Afterwards, an evaluation of our single scale object detection framework on KITTI dataset for vehicles tracking will be shown and discussed.

7.1 Software architecture

Main bottleneck of any vision-based object recognition application are feature detectors. In reference to the proposed framework in previous chapter, feature detectors referred to are SIFT and HoG. They may algorithmically execute in parallel, as they appear in different stages in the processig pipeline. However, every scale-invariant feature detector algorithm firstly downsamples the image in log_2 based scale-space (so-called "image pyramids" or "octaves"). Then, in each octave, it builds a gradient image in which it searches for maximal values. All values that are above a certain threshold (referred most often to as "contrast") are candidates for keypoints. In Figure 7.1 a creation of SIFT *keymap* may be seen, where each detected SIFT keypoint is saved in an image-size matrix of values 0-99 (the size of BoW vocabulary). Keypoints that are extracted from within multiple octaves are represented with bigger radii and are mapped in *keymap* with miltiple codewords, i.e. the SIFT keypoint with biggest radius is a maximum present in 5 octaves. The drawback of these algorithms in relation to subwindow search post processing logic is that they a priori do not allow for parallel execution of subwindow search tasks, eventually imposing accumulation of latencies of both stages I and II. In order to avoid this accumulation of latencies we propose to divide the input image on subimages upon which kepoints extraction algorithm is to be run - Figure 7.2. Based on last two figures, it may be said that image is divided into overlapping subimages of size $[X_{SIFT}, Y_{SIFT}]$, spaced horizontally by ΔD_x and vertically by ΔD_y. These values are defined based on empirical studies as:

$$[X_{SIFT}, Y_{SIFT}] = [2^a, 2^b] \tag{7.1}$$

where

$$a = floor(log_2(positives_x^*)) + 1 \tag{7.2}$$

$$b = floor(log_2(positives_y^*)) + 1 \tag{7.3}$$

and

$$[\Delta D_x, \Delta D_y] = [0.9 * X_{SIFT}, 0.9 * Y_{SIFT}] \tag{7.4}$$

In equations 7.2 and 7.3 the $positives^*$ variable stands for a positive sample from dataset that has an interest point at highest scale-space over all interest points extracted in all positive samples. Feature detectors often compensate for edges within an image (in order

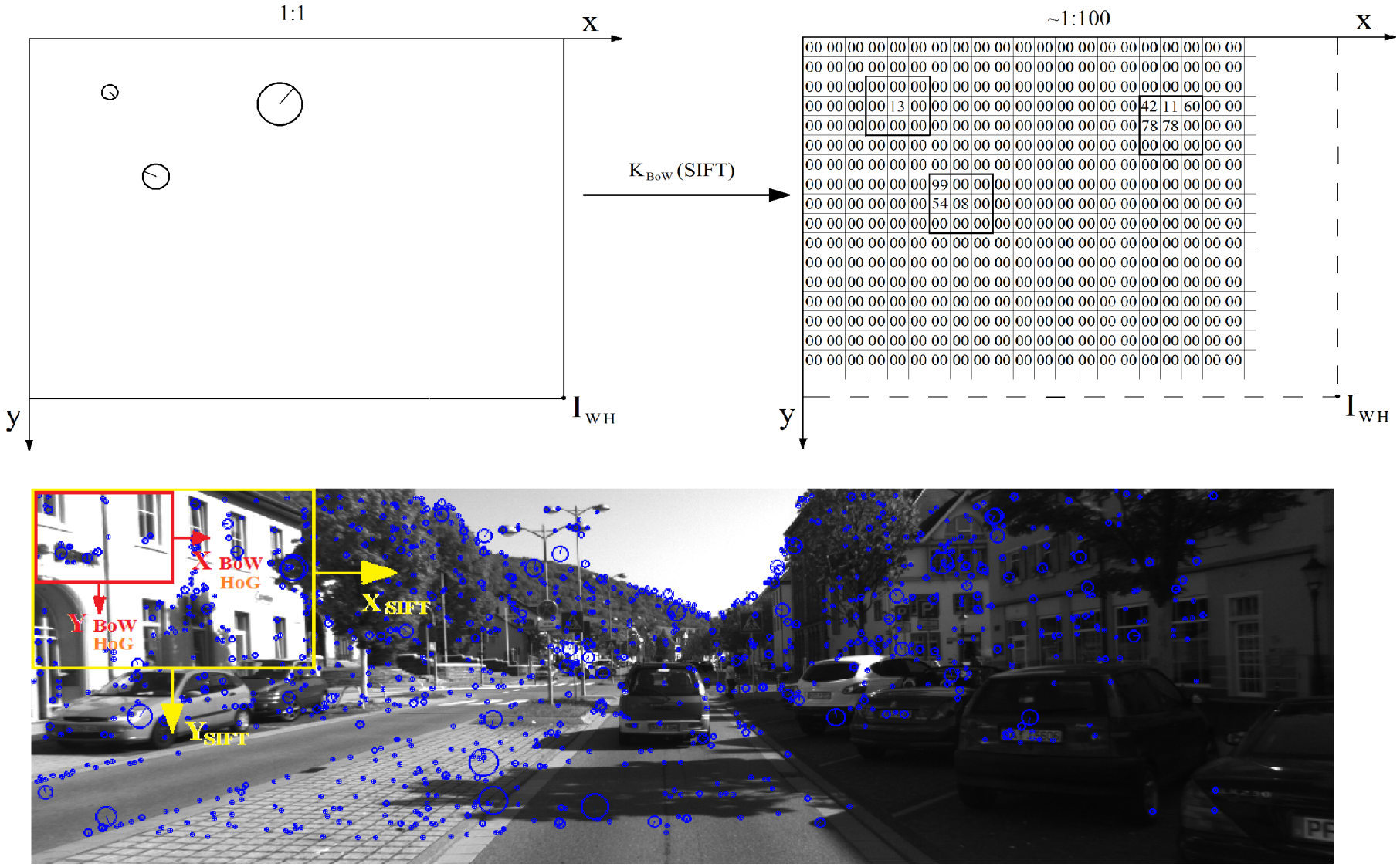

Figure 7.1: Partial SIFT KEYMAP, after having applied K_{BoW} on extracted SIFT keypoints

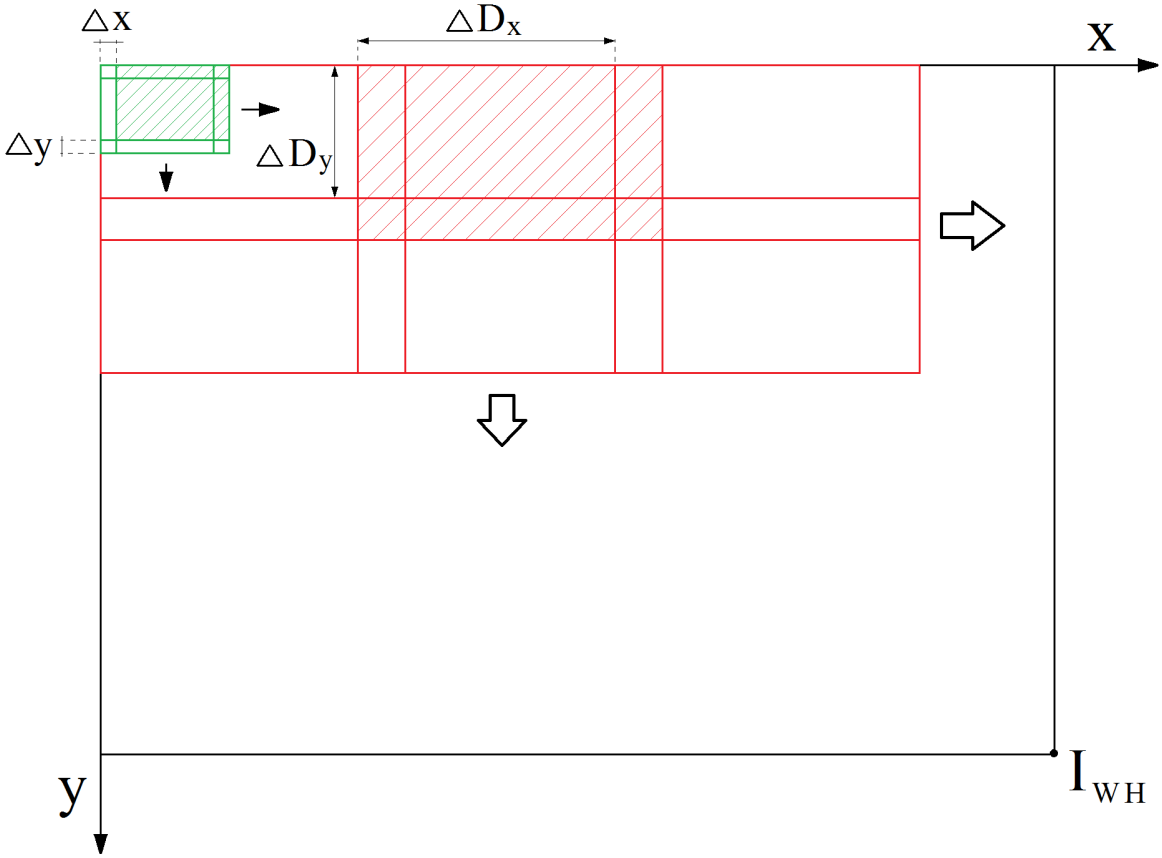

Figure 7.2: Parallelization of stages I and II

not to count in false interest points which appear close to image boundaries) because of incomplete pixel neighborhood. That is why a 10% overlap is used between subwindows - equation 7.4. Following these equations, algorithmic performance in stage I is being preserved. The smaller rectangle is the detection window with size s^* and $[\Delta x, \Delta y]$ are defined by horizontal and vertical strides.

7.1.1 Concurrence

Within the hierarchical object detection framework we define the following tasks:

- in Stage I : *IP_extract*, *build_keymap* and *SVM1* for interest keypoint extraction, BoW kernel computation and linear SVM dot-product calculation
- in Stage II : *get_HoG* and *SVM2* for HoG kernel computation with its respective linear classifier and
- in Stage III : *NMS* for H and V functions based final detection window size adjustment

As said in previous subsection, this software model may exhibit a minimal two-level parallelism, that is tasks in Stage I and II may run on two physical threads in parallel. A profiling[1] of this tasks based on a test video sequence[2] and on ZedBoard platform may be seen on

[1]average time consumption in milliseconds
[2]road sequence and experimental setup will be more detailed in proceeding subsection

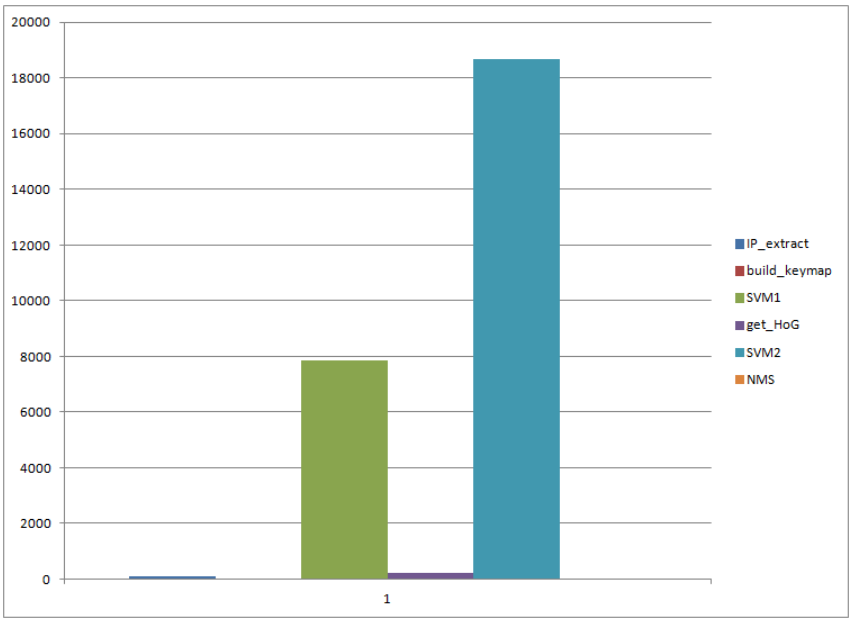

Figure 7.3: Tasks timing in hierarchical detection pipeline

Figure 7.3. We made use of NEON instruction-optimized OpenCV3.0.0 [OpenCV, 2015] libraries for feature detection and matching. Algorithms used for *IP_extract* task are detailed in subsection 7.2.1. It may be seen that most time is spent on classical dot-product vector-matrix operations, which may be significantly accelerated by a GPU-based (e.g. Nvidia's Jetson board) or equivalent FPGA-based (e.g. Xilinx Ultrascale) acceleration, depending on designer's choice of the embedded platform. On Figure 7.4 we emphasize duration of all other tasks than vector-matrix multiplication. Kernel computation in Stage II takes in average more than twice the time than feature detection, which leaves us for the timing bottleneck of the whole hierarchical detection pipeline to be the $IP_extract$ and get_HoG tasks (given that dot-product calculation is performed using off-the-shelf solutions supported by board vendors). Given our parallelization principle, this bottleneck is surmountable either as a software-only solution on i.e. quad-core platforms (platforms that provide higher number of physical threads, enabling at same time code optimizations in forms of vectorial instructions) or as a hardware solution where kernel(s) are implemented as co-processors, or a combination of two.

7.2 Algorithmic Evaluation

Our framework has been tested on road sequences in urban environments taken from the KITTI public data set [Geiger et al., 2012]. Here, a typical city road sequence contains ∼600 frames with ∼60 vehicles. The experimental setup is presented in Tables 7.1 - 7.5. In first table, bootstrapping iterations may be seen with growing number of hard negative samples. Classifier dimension denotes number of identified support vectors (over all samples in training set) lying on the separating hyperplane: greater it gets, higher is the dot-product induced computational latency. In second table, rule-based classification training results may be seen, based on 60190 samples generated after bootstrapping step (see previous table). Table 7.3 shows implementation details over interest keypoints ex-

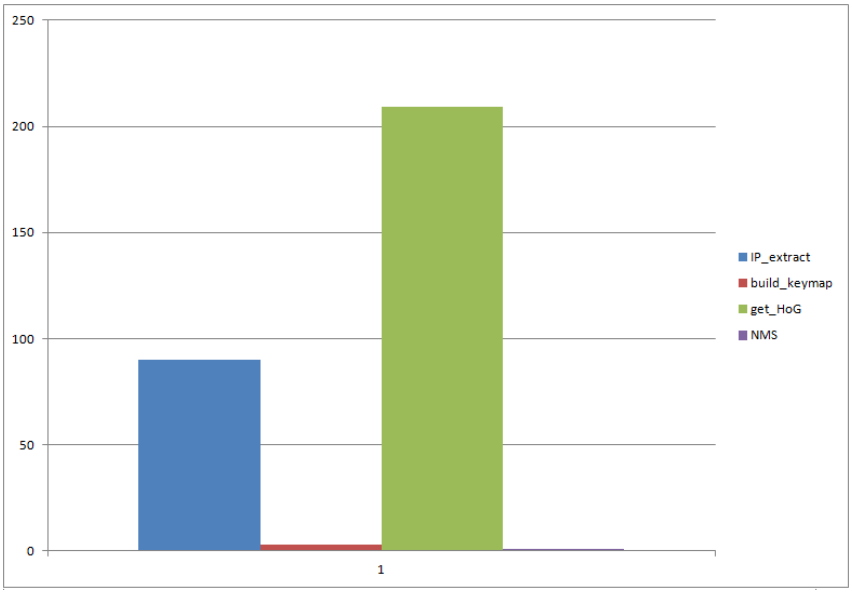

Figure 7.4: Tasks timing without linear SVMs computation

Table 7.1: Stage II training

Bootstrap	No. positives	No. negatives	Classifier dimension
Init	22768	2846	[569/25614]× 7942
*RTR*1	+0	+34110	[2857/59724]× 7942
*RTR*2	+0	+466	[3049/60190]× 7942

Table 7.2: Stage I training

No. classifiers	Costliest classifier dimension	P
11	[1033/13482]×100	[42 → 56]/69

Table 7.3: Stage I implementation

$[s_x^*, s_y^*]$	$[X_{SIFT}, Y_{SIFT}]$	$[\Delta x, \Delta y]$	N_{eval}^{IP}	avg.no.$\lvert KP \rvert$	$N_{eval}^{s^*}$
[152,76]	[256,128]	[8,8]	12	108	5206

Table 7.4: Stage II implementation

avg.overlap	avg. N_{eval}^{HoG}
73%	509

traction and sliding windows processes. Number of keypoints extraction subimages N_{eval}^{IP} may be calculated as:

$$N_{eval}^{IP} = \left(floor\left(\frac{I_W}{\Delta D_x}\right) - floor\left(\frac{X_{SIFT}}{\Delta D_x}\right)\right) * \left(floor\left(\frac{I_H}{\Delta D_y}\right) - floor\left(\frac{Y_{SIFT}}{\Delta D_y}\right)\right) \quad (7.5)$$

and the same principle applies to number of sliding windows evaluations in the image:

$$N_{eval}^{s^*} = \left(floor\left(\frac{I_W}{\Delta x}\right) - floor\left(\frac{s_x^*}{\Delta x}\right)\right) * \left(floor\left(\frac{I_H}{\Delta y}\right) - floor\left(\frac{s_y^*}{\Delta y}\right)\right) \quad (7.6)$$

Average number of interest keypoints (avg. no. $|KP|$) found in an extraction subimage influences the need of writing efficient software code for keymap-based histogram handling (Figure 7.1 and equation 6.3) : bigger the average number of keypoints, it is more necessary. In this implementation, for vehicles class of objects, average 103 keypoints do not present a demanding computational effort for this implementation. In fourth table, average number of HoG kernel evaluations N_{eval}^{HoG} in Stage II is given, along with average overlap percentage based on adjacent detection windows in which HoG descriptor is computed (either in Δx or Δy sliding direction). This overlap percentage denotes the number of overlapping windows over number of all windows in Stage II. Higher overlap percentage decreases the computational load, as calculating this gradient vector is a compositional operation[3]. Finally in the last table, threshold values for non-maximal suppression in Stage III are presented. Number of final, adjusted size-detection windows, passed for concatenation is denoted with N_{eval}^{DFS}. It is worth noting that an initial detection window based search space in an image of dimension $[I_W, I_H] = [1242, 375]$ of $N_{eval}^{s^*} = 5206$ evaluations in Stage I is filtered to $N_{eval}^{HoG} = 509$ in Stage II and finally reduced to a mere number of $N_{eval}^{DFS} = 14$ adjusted windows in Stage III, in average.

For the sake of a fair comparison with performing state-of-the-art object detection techniques, we will give two separate evaluations of the proposed framework: over one

[3]HoG vector descriptor is basicaly a concatenation of gradients values computed on evenly spaced groups of pixels

Table 7.5: Stage III implementation

t_0	t_1	t_2	t_3	N_{eval}^{DFS}
0.3	0.1	1.1	0.1	14

road sequence and over the whole testing dataset for tracking. On both sequences we perform *tracking by detection*, using the Matlab open-source code provided by KITTI staff, based on [Geiger et al., 2014]. The evaluation pipeline is as follows[4]:

1. Our software implementation performs object detection independently in each frame

2. Inter-frame detected object's position estimation is made by a Kalman filter, following data association via the Hungarian method. For each detection a tracklet is initialized.

3. In following frames, tracklets are associated between themselves via the Hungarian method. Only tracklets that are at least three frames long are considered.

Evaluation over a city road sequence. Detection results only may be seen here [Tertei et al., 2016a], and this video [Tertei et al., 2016b] shows them coupled in a tracking scheme. From the first video we count true positives as: i) detections which have an overlap of at least 50% with ground truth and ii) they have been redetected in at least three consecutive frames. All vehicle types: cars, vans and trucks are counted as ground truth. These criteria encompass requirements for our future work on tracking by detection. We calculate the misclassification rate to be 15/55[5]=27%. By lowering the stride parameter from 8 to 4, the misclassification rate drops to 9% at cost of smaller precision and higher recall of the classifier.

Evaluation over KITTI's tracking testbed. Tracking results, based on our detection framework have been submitted to the KITTI's server for a more objective and severe evaluation. Our single-scale based reasoning has 48% precision with a smaller recall rate. Also, its multiple object tracking precision (MOTP [Bernardin and Stiefelhagen, 2008]) metric is 65%. We provide three videos in order to briefly discuss the issue:

1. Video showing high precision and recall [Tertei et al., 2016c], low amount of false positives and no false negatives.

2. Mediocre precision and recall: [Tertei et al., 2016d]. Higher amount of false negatives and inconsistent frame-to-frame tracklet association (many are reinitialized on the same object).

3. High precision and recall [Tertei et al., 2016e].

7.2.1 Estimation of vehicles detection on an embedded platform

Given that the aforementioned SIFT keypoints calculation took ∼4 seconds on dual-core i3 platform @ 1.8GHz, it cannot be considered for real-time interest point characterization. Instead, faster feature detectors with binary pixels neighborhood description should be used. However, in order of maintaining application performances with SIFT, firstly multi-scale feature detectors should be used in combination with suitable rotation- and scale-invariant descriptors. Authors in [Leutenegger et al., 2011] introduce Binary Robust Invariant Scalable Keypoints (BRISK). They make use of multi-scale Adaptive and

[4]our contribution lies in the first stage

[5]52 cars and 3 vans

Generic Accelerated Segment Test (AGAST [Mair et al., 2010]) feature detector and show detailed comparisons with SIFT and SURF. They conclude very similar performances in cases when applying small scale-space image transformations on criteria of keypoints repeatability, which is essential for interest points based logic in Stage I of our application. Similar experimental evaluations are performed within a relatively novel binary descriptor - Fast Retina Keypoint (FREAK [Alahi et al., 2012]). Keypoints neighborhood descriptions are matched via Hamming distance instead of Euclidean distance. FREAK description is presented on the basis of multi-scale AGAST detector, which is the same keypoint characterisation method we intend to use when reproducing our application's algorithmic performances while embedding the detection application. In order to do so, one needs to close the performance gap between baseline SIFT evaluation and AGAST-FREAK evaluation by respecting some constraints:

- Determination of *contrast* threshold that influences average number of generated interest keypoints. It is not at all straightforward how many feature points extracted in average in a subimage contribute to a quality appearance kernel. In lack of an analytical solution, the performance evaluation should be based on predictions of trained SVMs in rule-based classification scheme. As it is shown in [Leutenegger et al., 2011], the performance function based on different number of keypoints is monotone, which helps a lot in an empirical search for optimal *contrast* value, which should be varied accordingly.

- Determination of maximum and minimum values defining *scale-space*. For example, when choosing the right configuration for multi-scale AGAST, number of scale-spaces is determined after equation 7.1.

These *performance-equivalence* relations are based on empirical evaluation metrics presented in papers [Leutenegger et al., 2011] and [Alahi et al., 2012]. Timing estimation concerns the ordering of physical threads and tasks:

- $Thread_1$ ($T1$) should execute tasks $IP_extract$ and *build_keymap* in parallel with

- $Thread_2$ ($T2$) which executes $SVM1$ and get_HoG tasks.

- Remaining tasks should sequentially be executed after $Thread_1$ and $Thread_2$ pass first $[X_{SIFT}, Y_{SIFT}]$ subimage on a single thread ($T3$).

In order to insure real-time performance, the dot-product and get_HoG tasks should be executed on the device. In a given heterogeneous system, the device could be either an embedded GPU (e.g. Nvidia Jetson board) or an FPGA. Hereby detailed computational latency may be expressed as:

$$L = max(L_{T1}, L_{T2}) + L_{T3} < 100ms \tag{7.7}$$

where $100ms$ is the real-time constraint of 10Hz. Overall dot-product $SVM1$ computational complexity expressed in FLOPs may be calculated after Tables 7.2 and 7.3 as:

$$N_{FLOP_{SVM1}} = 5206 \times (2 * 100 - 1) * 1033 * 1 = 1070181802 \tag{7.8}$$

and the same for $SVM2$ is deduced after Tables 7.1 and 7.4 as:

$$N_{FLOP_{SVM2}} = 509 \times (2 * 7942 - 1) * 2857 * 1 = 23097265079 \tag{7.9}$$

Both configurable and fixed embedded hardware architectures must comply to these computational efficiency constraints. For fixed hardware, i.e. with an embedded GPU, minimal latencies can be calculated as:

$$L_{T2} = \frac{N_{FLOP_{SVM1}}}{MAX_{GFLOPS}} \tag{7.10}$$

$$L_{T3} = \frac{N_{FLOP_{SVM2}}}{MAX_{GFLOPS}} \tag{7.11}$$

In case of Nvidia's Jetson board, $MAX_{GFLOPS} > 300/s$ [NVIDIA, 2015a] which leaves $L_{T2} < 3.57ms$ and $L_{T3} < 77ms$. This board features also an off-the-shelf GPU implementation for HoG. For a reconfigurable hardware, same timing performances may be met by developing a fixed-point multiplication architecture for dot-product operations and by using the architectural concept for efficient HoG described in [Hahnle et al., 2013]. We note that, under these achievable timing constraints, $Thread_2$ takes less time to execute than $Thread_1$, so it is worth also launching $Thread_3$ (if the hardware platform supports that many physical threads) in order to better meet real-time application performance.

7.3 Conclusion

In this chapter a software implementation of single-scale model-based detection framework is presented. It is evaluated on KITTI urban road sequences with vehicles types of obstacles. This approach permits higher execution times than standard multi-scale detection schemes on mobile processing devices while retaining good detection performance, making it a suitable solution for object detection tasks on both contemporary and emerging embedded hardware. Future work encompasses improvements in algorithmic performance. More hard positive samples in the form of object's truncated representation in a given fixed detection window should be included. This way we should be able to better performances of both first and second stage classifiers.

Chapter 8

Conclusion

The problems addressed in this thesis concern two basic perception functionalities in a navigational pipeline in mobile robotics: localization and obstacle detection. Moreover, the key issue in this domain, besides algorithmic performance, is level of autonomy of the mobile platform which directly calls for efficient embedded design of these functional blocks. This thesis describes and evaluates an efficient FPGA-based hardware accelerator for solving localization task and proposes a model-based obstacle detection framework to be executed on multi-core heterogeneous[1] hardware platforms. Navigation based on visual odometry with obstacle detection may be used in all indoors/outdoors and both natural and structured environments for different kinds of applications: AGVs, UGVs, MAVs, intelligent vehicles and even augmented reality with a device (tablet, smart phone) moved by a human.

Firstly, knowledge from state-of-the-art in the domains of robot localization techniques and model-based obstacle detection in computer vision is put forward. For localization, an introduction into SLAM was shown with accent on monocular visual filtering SLAM implementations, which are more suitable for VO functionality on compact systems than Optimization techniques. Obstacle recognition process is presented, based on sliding windows exhaustive approach which has for its advantage that it does not make a priori assumptions on environment or camera intrinsics and extrinsics. For designing adequate embedded systems, hardware/software co-design methodology is explained.

Going further through chapters 4 and 5, step-by-step implementation of co-design of CSLAM algorithm into SoPC architecture is covered. During this process, FPGA accelerators in form of EKF, FAST and Correlator[2] co-processors are designed, integrated and validated on ZedBoard heterogeneous[3] platform running Xillinux operating system. Profiling the accelerated application, it is shown that CSLAM may be run on ZedBoard at 100Hz, implementing both front-end and back-end tasks and consuming less than 5W. Main research highlights regarding EKF co-processor are as follow:

- Exploiting cross-covariance matrix symmetry greatly reduces computational and resource costs,

[1] multi-core CPU + FPGA

[2] FAST corner detector and BRIEF Correlator hardware accelerators are developed by a fellow PhD student in RAP group in LAAS-CNRS

[3] CPU + FPGA

- It should be stored in BRAM for efficient execution of VO on heterogeneous platform,
- EKF innovation matrix dimension allows for simple systolic array based computational designs,
- Our innovation supports observations of 60 AHP landmarks in real time on Zynq-7020 device
- Accelerator is generic regarding system state size - other sensors may be added/removed, and feature state size - up to seven-dimensional and
- Proposed hardware accelerator for EKF VO functionality is platform independent.

For model-based obstacle detection, categorical object classifiers are trained and tested based on a suitable annotated database - KITTI. In chapter 6 a single-scale detection window based object localization framework is proposed. Its main components are i) Stage I appearance kernel, ii) Stage II shape kernel and iii) Stage III non-maximal suppression or final detection window adjustment reasoning. In order to define adequate feature space representations of obstacles with corresponding discriminative (SVM) classifiers in first two stages, algorithmic performance regarding classification accuracy vs kernels dimensionality co-design is presented:

- First study is conducted to determine an optimal detection window size for a given object type. It is based on EER metric of test responses from object representation with higher dimension (which is shape kernel in case of vehicles, calculated by HoG descriptor) coupled to an SVM classifier with default training parameters,
- Second study investigates whether to use non-linear kernel trick for SVM classification in second, scoring stage in the framework. It is important to relieve here as much computational effort as possible given the large dimensionality of shape kernel and in our case we chose using linear SVM kernel (dot-product operation).
- Third trait is definition of boosting-like cascade which we call *Rule-based* classification in order not to use non-linear kernel tricks in first stage of the framework, as it is applied most often in sliding windows mode. The *rule* defines a variable upon which a complex, linearly non-separable, training set may be divided into subsets upon which dot-product SVM may be successfuly trained. In our case the variable presented number of non-zero elements in positive samples dataset.

Afterwards, a Bayesian scheme is developed further acted on scoring responses from second stage: all overlapping detection windows with high scores are concatenated, others (if present) are rejected. Based on a final confidence score, resulting detection window size is determined.

Lastly, in Chapter 7, a case-study of the proposed framework is given on vehicles type of obstacles. It shows good algorithmic performance based on MOTP metric, and future work on its amelioration is mentioned. A software architecture that enables concurrent execution of interest points extraction and visual vocabulary matching (keypoints ectraction part of Stage I) along with HoG and dot-product computations for classification with consequent non-maximal suppression (classification part of Stage I, all Stages II and III)

is described. Based on profiling results on ZedBoard, tasks partitioning on at least two physical threads-enabling hardware platform (either reconfigurable i.e. Xilinx Ultrascale or not i.e. Nvidia Jetson boards) for real-time object detection on an embedded platform is explained.

This work enabled several scientific communications:

Science Citation Indexed international journal

1. Tertei D. T., Piat J., and Devy M. (201?). FPGA design of EKF block accelerator for 3D visual SLAM. *International Journal of Computers and Electrical Engineering (CAEE).* DOI: 10.1016/j.compeleceng.2016.05.003.

International conference

1. Tertei D. T., Piat J., and Devy M. (2014). FPGA design and implementation of a matrix multiplier based accelerator for 3D EKF SLAM. *In Proc. Conference on ReConFigurable Computing and FPGAs (ReConFig).*

A fast visual EKF-SLAM gives rise to more robust egomotion estimation as the number of observed points in its sparse map may be considerably increased. Optimization methods correct travelled trajectories by means of loop closure i.e. by using non-linear optimization techniques (such as bundle adjustment) upon all last camera poses when revisiting a known location. However in order to do so efficiently, they need a preferably large number of previously saved keypoints at those camera poses for matching purposes. The proposed embedded architecture for VSLAM in this thesis allows for implementation of loop closure within EKF based Filtering methods on an embedded system too, refining current robot's pose based on past camera poses.

Single scale model-based obstacle detection framework presents a viable basis for implementing complete embedded obstacle detection systems, depending on the environment type: structured on unstructured with given obstacle classes i.e. pedestrians, cars, trucks, vans, bicycles and road signs for an urban environment. Final responses on particular obstacle class and location within an image would be determined based on classifiers' confidence score. This complete embedded obstacle recognition system would, in such a manner, provide the mobile platform with necessary information for safe navigation in crowded environments: i) either on the "per frame" basis where the whole detection pipeline is run independently in each frame; ii) either by using the acquired information from within previous frame in a "tracking" fashion in current frame so as to reduce computational effort in case of large numbers of obstacle classes. As a mobile robot may be equipped also with other exteroceptive sensors such as lasers, RGBD, and/or LIDARs besides camera(s), this information may be fused with the proposed embedded detection system where hypothesis verification is being reinforced by additional sensorial input. This is of particular interest in cases where obstacles are in nearest viccinity of the mobile robot so they cannot be physically perceived in full (in a holistic manner) on camera(s), such as when navigating in a winyard besides agricultural staff or on an airport during pre take-off inspection because of frequent near field-of-view crossovers (in camera(s) frames) by operators and technicians.

In context of performing SLAMMOT on an embedded platform, an integration of these two functional blocks proposed may be envisaged also as subject of future research.

Their combination would permit a more stable navigation, knowing that dynamical objects in the scene may be recognized by the detection block: after foreground subtraction, the VSLAM component may initialize and observe only points that belong to static background, which is a presumption upon which SLAM paradigm works. Mobile robots that don't perform obstacle detection during visual odometry compensate by initializing and observing large amounts of landmarks for statisticaly proper egomotion estimation in return. That solution is not practical when it comes to embedded systems because of high computational load due to image processing tasks.

In same context, besides instantly eliminating moving obstacles, the model-based obstacle detection component also identifies stationary obstacles (i.e. parked vehicles, bicycles, etc.) which is of particular interests for navigation in urban environments. In that manner when revisiting known places, a more robust, non-corrupted (obstacle-free), loop closure may be performed.

Contributions of the thesis should reflect in all the industries which make use of autonomous navigation:

- mobile robotics,
- automotive industry,
- surveillance and inspection
- space industry

and sincerely I hope it will serve as reference for SoPC design of future navigational applications for other researchers or roboticians.

Bibliography

[Acunzo et al., 2007] Acunzo, D., Zhu, Y., Xie, B., and Baratoff, G. (2007). Context-adaptive approach for vehicle detection under varying lighting conditions. In *Proc. IEEE Intelligent Transportation Systems Conference.*

[Advani et al., 2015] Advani, S., Tanabe, Y., Irick, K., Sampson, J., and Narayanan, V. (2015). A scalable architecture for multi-class visual object detection. In *25th International Conference on Field Programmable Logic and Applications (FPL).*

[Afonso et al., 2011] Afonso, G., Ben Atitallah, R., Loyer, A., Dekeyser, J., Belanger, N., and Rubio, M. (2011). A prototyping environment for high performance reconfigurable computing. In *Proc. of IEEE International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, Montpellier, France.

[Agarwal et al., 2004] Agarwal, S., Awan, A., and Roth, D. (2004). Learning to detect objects in images via a sparse part-based representation. *IEEE Trans. Pattern Anal. Mach. Intell.*

[Alahi et al., 2012] Alahi, A., Ortiz, R., and Vandergheynst, P. (2012). FREAK: Fast Retina Keypoint. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[Atev et al., 2005] Atev, S., Arumugam, H., Masoud, O., Janardan, R., and Papanikolopoulos, N. P. (2005). A vision-based approach to collision prediction at traffic intersections. In *IEEE Trans. Intell. Transp. Syst.*

[Atev and Papanikolopoulos, 2008] Atev, S. and Papanikolopoulos, N. P. (2008). Multi-view 3-D vehicle tracking with a constrained filter. In *Proc. IEEE ICRA.*

[B. Leibe and Schiele, 2004] B. Leibe, A. L. and Schiele, B. (2004). Combined object categorization and segmentation with an implicit shape model. In *Proc. ECCV Workshop Stat. Learn. Comput. Vis.*

[Balboni et al., 1996] Balboni, A., Fornaciari, W., and Sciuto, D. (1996). Tosca: A pragmatic approach to co-design automation of control-dominated systems. In De Micheli, G. and Sami, M., editors, *Hardware/Software Co-Design*, volume 310 of *NATO ASI Series*, pages 265–294. Springer Netherlands.

[Ballard, 1981] Ballard, D. H. (1981). Generalizing the hough transform to detect arbitrary shapes. *Pattern Recognition.*

[Bay et al., 2006] Bay, H., Tuytelaars, T., and Gool, L. V. (2006). SURF: Speeded-up robust features. In *Proc. ECCV*.

[Belongie et al., 2002] Belongie, S., Malik, J., and Puzicha, J. (2002). Shape Matching and Object Recognition Using Shape Contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[Bernardin and Stiefelhagen, 2008] Bernardin, K. and Stiefelhagen, R. (2008). Evaluating multiple object tracking performance: the CLEAR MOT metrics. *EURASIP Journal on Image and Video Processing*.

[Bertozzi and Broggi, 1998] Bertozzi, M. and Broggi, A. (1998). Gold: A parallel real-time stereo vision system for generic obstacle and lane detection. *IEEE Transactions on Image Processing*.

[Betge-Brezetz et al., 1996] Betge-Brezetz, S., Hebert, P., Chatila, R., and Devy, M. (1996). Uncertain map making in natural environments. In *IEEE International Conference on Robotics and Automation*.

[Betke et al., 1996] Betke, M., Haritaglu, E., and Davis, L. (1996). Multiple vehicle detection and tracking in hard real time. In *IEEE Intelligent Vehicles Symposium (IV)*.

[Bezati et al., 2014] Bezati, E., Thavot, R., Roquier, G., and Mattavelli, M. (2014). High-level dataflow design of signal processing systems for reconfigurable and multicore heterogeneous platforms. *Journal of Real-Time Image Processing*, 9(1):251–262.

[Bonato et al., 2008] Bonato, V., Marques, E., and Constantinides, G. (2008). A Floating-Point Extended Kalman Filter Implementation for Autonomous Mobile Robots. *Journal of VLSI Signal Processing*.

[Borgefors, 1988] Borgefors, G. (1988). Hierarchical chamfer matching: A parametric edge matching algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[Botero et al., 2012] Botero, D., Piat, J., Devy, M., and Boizard, J. (2012). An fpga accelerator for multispectral vision-based ekf-slam. *Proc. IROS Workshop on Smart CAmeras for roBOTic applications (SCaBot), Vilamoura (Portugal)*.

[Botero-Galeano, 2012] Botero-Galeano, D. (2012). *Development of algorithms and architectures for driving assistance in adverse weather conditions using FPGAs*. PhD thesis, INSA de Toulouse.

[Bradski and Kaehler, 2008] Bradski, G. and Kaehler, A. (2008). *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.".

[Bresson et al., 2014] Bresson, G., Feraud, T., Aufrere, R., Checchin, P., and Chapuis, R. (2014). Real time monocular slam with low memory requirements. *IEEE Intelligent Transportation Systems Transactions and Magazine*.

[Bruhn et al., 2005] Bruhn, A., Weickert, J., and Schnorr, C. (2005). Lukas/Kanade meets Horn/Schunck: Combining local and global optic flow methods. *International Journal of Computer Vision*.

[Buch et al., 2009a] Buch, N., Cracknell, M., Orwell, J., and Velastin, S. A. (2009a). Vehicle localization and classification in urban CCTV streams. In *Proc. 16th ITS WC.*

[Buch et al., 2008] Buch, N., Orwell, J., and Velastin, S. A. (2008). Detection and classification of vehicles for urban traffic scenes. In *Proc. Int. Conf. VIE.*

[Buch et al., 2009b] Buch, N., Orwell, J., and Velastin, S. A. (2009b). Three-dimensional extended histograms of oriented gradients (3-DHOG) for classification of road users in urban scenes. In *Proc. BMVC.*

[Buch et al., 2010] Buch, N., Orwell, J., and Velastin, S. A. (2010). Urban road user detection and classification using 3-D wireframe models. *IET Comput. Vis.*

[Buch et al., 2009c] Buch, N., Yin, F., Orwell, J., Makris, D., and Velastin, S. A. (2009c). Urban vehicle tracking using a combined 3-D model detector and classifier. *Knowledge-Based and Intelligent Information and Engineering Systems.*

[Canny, 1986] Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*

[Castellanos et al., 1997] Castellanos, J., Tardos, J., and Schmidt, G. (1997). Building a global map of the environment of a mobile robot: The importance of correlations. In *IEEE International Conference on Robotics and Automation.*

[Castillo-Atoche et al., 2010] Castillo-Atoche, A., Torres-Roman, D., and Shkvarko, Y. (2010). Towards real time implementation of reconstructive signal processing algorithms using systolic array coprocessors. *Journal of Systems Architecture*, pages 327–339.

[Chang et al., 2013] Chang, L., Hernandez-Palancar, J., Sucar, L. E., and Arias-Estrada, M. (2013). FPGA-based detection of SIFT interest keypoints. *Machine Vision and Applications.*

[Cheeseman et al., 1987] Cheeseman, R., Smith, R., and Self, M. (1987). A stochastic map for uncertain spatial relationships. In *International Symposium on Robotic Research.*

[Cheng, 1995] Cheng, Y. (1995). Mean shift mode seeking and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*

[Choset and Nagatani, 2001] Choset, H. and Nagatani, K. (2001). Topological simultaneous localization and mapping (SLAM): toward exact localization without explicit localization. In *IEEE Transactions on Robotics and Automation.*

[Civera et al., 2006] Civera, J., Davison, A. J., and Montiel, J. M. (2006). Unified Inverse Depth Parametrization for Monocular SLAM. In *In Proceedings of Robotics: Science and Systems.* Citeseer.

[Cornelis et al., 2008] Cornelis, N., Leibe, B., Cornelis, K., and Gool, L. V. (2008). Three-dimensional urban scene modeling integrating recognition and reconstruction. *Int. J. Comput. Vis.*

[Crandall et al., 2005] Crandall, D., Felzenszwalb, P., and Huttenlocher, D. (2005). Spatial priors for part-based recognition using statistical models. In *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*

[Creusen et al., 2009] Creusen, I., Wijnhoven, R., and de With, P. H. N. (2009). Applying feature selection techniques for visual dictionary creation in object classification. In *Proc. Int. Conf. IPCV Pattern Recog.*

[Csorba, 1998] Csorba, M. (1998). *Simultaneous localisation and map building.* PhD thesis, Univeristy of Oxford.

[Dalal and Triggs, 2005] Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *Proc. IEEE Comput. Soc. Conf. CVPR.*

[Daugman, 1985] Daugman, J. (1985). Uncertainty Relation for Resolution in Space, Spatial Frequency, and Orientation Optimized by Two-Dimensional Visual Cortical Filters. *J. Optical Soc. Am.*

[Davison, 2003a] Davison, A. (2003a). Real-time simultaneous localisation and mapping with a single camera. In *IEEE International Conference on Computer Vision.*

[Davison, 2003b] Davison, A. J. (2003b). Real-time simultaneous localisation and mapping with a single camera. In *Proc. Ninth IEEE International Conference on Computer Vision - Volume 2*, ICCV '03.

[Dissanayake et al., 2001] Dissanayake, M., Newman, P., Clark, S., Durrant-Whyte, H., and Csorba, M. (2001). A solution to the simultaneous localization and map building (slam) problem. *IEEE Transactions on Robotics and Automation.*

[Doersch and Efros,] Doersch, C. and Efros, A. Improving the hog descriptor.

[Dollar et al., 2012] Dollar, P., Wojek, C., Schiele, B., and Perona, P. (2012). Pedestrian detection: An evaluation of the state of the art. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(4):743–761.

[Engel et al., 2014] Engel, J., Schöps, T., and Cremers, D. (2014). Lsd-slam: Large-scale direct monocular slam. In *Computer Vision–ECCV 2014*, pages 834–849. Springer.

[Fei-Fei and Perona, 2005] Fei-Fei, L. and Perona, P. (2005). A bayesian hierarchical model for learning natural scene categories. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR).*

[Fischler and Bolles, 1981] Fischler, M. and Bolles, R. (1981). Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM.*

[Florack et al., 1994] Florack, L. M. J., t. Haar Romeny, B. M., Koenderink, J. J., and Viergever, M. A. (1994). General intensity transformations and differential invariants. *Journal of Mathematical Imaging and Vision.*

[Forster et al., 2014] Forster, C., Pizzoli, M., and Scaramuzza, D. (2014). Svo: Fast semi-direct monocular visual odometry. In *Proc. IEEE International Conference on Robotics and Automation (ICRA).*

[Freeman and Adelson, 1991] Freeman, W. T. and Adelson, E. H. (1991). The Design and Use of Steerable Filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[Freund, 1995] Freund, Y. (1995). Boosting a weak learning algorithm by majority. *Information and Computation*.

[Freund and Schapire, 1999] Freund, Y. and Schapire, R. E. (1999). A short introduction to boosting. In *Proceedings of the 16th international joint conference on Artificial intelligence*.

[Gabor, 1946] Gabor, D. (1946). Theory of Communication. *J. IEE*.

[Geiger et al., 2014] Geiger, A., Lauer, M., Wojek, C., Christoph, C., and Urtasun, R. (2014). 3d traffic scene understanding from movable platforms. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE (PAMI)*.

[Geiger et al., 2013] Geiger, A., Lenz, P., Stiller, C., and Urtasun, R. (2013). Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*.

[Geiger et al., 2012] Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Gonzalez et al., 2011] Gonzalez, A., Codol, J., and Devy, M. (2011). A C-embedded Algorithm for Real-Time Monocular SLAM. In *18th International Conference on Electronics, Circuits and Systems*, Beyrouth, Liban.

[Hahnle et al., 2013] Hahnle, M., Saxen, F., Hisung, M., Brunsmann, U., and Doll, K. (2013). FPGA-Based Real-Time Pedestrian Detection on High-Resolution Images. In *Computer Vision and Pattern Recognition Workshops (CVPRW)*.

[Hanley and McNeil, 1982] Hanley, J. A. and McNeil, B. J. (1982). The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36.

[Harris and Pike, 1987] Harris, C. and Pike, J. (1987). 3d positional integration from image sequences. In *In Proc. Alvey Vision Conference, Cambridge.England*.

[Harris and Stephens, 1988] Harris, C. and Stephens, M. (1988). A combined corner and edge detector. In *Proceedings of the Alvey Vision Conference*.

[Hartley and Zisserman, 2000] Hartley, R. and Zisserman, A. (2000). *Multiple view geometry in computer vision*. Cambridge University Press.

[Hartley and Zisserman, 2004] Hartley, R. I. and Zisserman, A. (2004). *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition.

[Harzallah et al., 2009] Harzallah, H., Jurie, F., and Schmid, C. (2009). Combining efficient object localization and image classification. In *ICCV 2009 - 12th International Conference on Computer Vision*, pages 237–244, Kyoto, Japan. IEEE.

[Irki et al., 2013] Irki, Z., Bendaoudi, H., Devy, M., and Khouas, A. (2013). Fpga implementation of the v-disparity based obstacles detection approach. In *Control & Automation (MED), 2013 21st Mediterranean Conference on*, pages 1104–1111. IEEE.

[Itu and Danescu, 2014] Itu, R. and Danescu, R. (2014). An Efficient Obstacle Awareness Application for Android Mobile Devices. In *IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*.

[Jaulin, 2011] Jaulin, L. (2011). Range-only slam with occupancy maps: A set-membership approach. *Robotics, IEEE Transactions on*, 27(5):1004–1010.

[Jones and Snow, 2008] Jones, M. J. and Snow, D. (2008). Pedestrian detection using boosted features over many frames. In *Proc. ICPR*.

[Jovanovic and Milutinovic, 2012] Jovanovic, Z. and Milutinovic, V. (2012). FPGA accelerator for floating-point matrix multiplication. *IET Computers and Digital Techniques*.

[Jurie and Triggs, 2005] Jurie, F. and Triggs, B. (2005). Creating efficient codebooks for visual recognition. In *Proc. IEEE Int. Conf. on Computer Vision (ICCV)*.

[Kaess et al., 2011] Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J. J., and Dellaert, F. (2011). isam2: Incremental smoothing and mapping using the bayes tree. *The International Journal of Robotics Research*, page 0278364911430419.

[Kahn, 1974] Kahn, G. (1974). The semantics of simple language for parallel programming. In *IFIP Congress*, pages 471–475.

[Kalman, 1960] Kalman, R. (1960). *A new approach to linear filtering and prediciton problems.* Basic Engineering.

[Ke and Sukthankar, 2004] Ke, Y. and Sukthankar, R. (2004). PCA-SIFT: A More Distinctive Representation for Local Image Descriptors. In *Proc. Conf. Computer Vision and Pattern Recognition*.

[Khammari et al., 2005] Khammari, A., Nashashibi, F., Abramson, Y., and Laurgeau, C. (2005). Vehicle detection combining gradient analysis and AdaBoost classification. In *Proc. IEEE Intell. Transp. Syst.*

[Kim and Malik, 2003] Kim, Z. and Malik, J. (2003). Fast vehicle detection with probabilistic feature grouping and its application to vehicle tracking. In *Proc. 9th IEEE Int. Conf. Comput. Vis.*

[Klein and Murray, 2007] Klein, G. and Murray, D. (2007). Parallel tracking and mapping for small AR workspaces. In *Proc. 6th IEEE and ACM Int. Symp. on Mixed and Augmented Reality*. IEEE Computer Society.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[Kümmerle et al., 2011] Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K., and Burgard, W. (2011). g 2 o: A general framework for graph optimization. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3607–3613. IEEE.

[Labayrade, 2004] Labayrade, R. (2004). *Détection générique, robuste et rapide d'obstacles routiers par stéréovision embarquée.* PhD thesis, Paris 6.

[Lampert et al., 2008] Lampert, C. H., Blaschko, M. B., and Hofmann, T. (2008). Beyond sliding windows: Object localization by efficient subwindow search. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[Lazebnik et al., 2003] Lazebnik, S., Schmid, C., and Ponce, J. (2003). Sparse Texture Representation Using Affine-Invariant Neighborhoods. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[Lee and Messerschmitt, 1987] Lee, E. A. and Messerschmitt, D. G. (1987). Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. Comput.*, 36(1):24–35.

[Lee and Parks, 1995] Lee, E. A. and Parks, T. (1995). Dataflow process networks. In *Proceedings of the IEEE*, pages 773–799.

[Lee and Lee, 2013] Lee, S. and Lee, S. (2013). Embedded visual SLAM: applications for low-cost consumer robots. *IEEE Robot. Automat. Mag.*, 20(4):83–95.

[Leibe et al., 2007] Leibe, B., Cornelis, N., Cornelis, K., and Gool, L. V. (2007). Dynamic 3-D scene analysis from a moving vehicle. In *Proc. IEEE Conf. CVPR.*

[Leibe et al., 2008a] Leibe, B., Leonardis, A., and Schiele, B. (2008a). Robust object detection with interleaved categorization and segmentation. *Int. J. Comput. Vis.—Special Issue on Learning for Recognition and Recognition for Learning.*

[Leibe et al., 2008b] Leibe, B., Schindler, K., Cornelis, N., and Gool, L. V. (2008b). Coupled object detection and tracking from static cameras and moving vehicles. In *IEEE Trans. Pattern Anal. Mach. Intell.*

[Leibe et al., 2005] Leibe, B., Seemann, E., and Schiele, B. (2005). Pedestrian detection in crowded scenes. In *IEEE Comput. Soc. Conf. CVPR.*

[Leutenegger et al., 2011] Leutenegger, S., Chli, M., and Siegwart, R. Y. (2011). Brisk: Binary robust invariant scalable keypoints. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2548–2555. IEEE.

[Li et al., 2012] Li, S., Wang, D., Zheng, Z., and Wang, H. (2012). Multi-view vehicle detection in traffic surveillance combining hog-hct and deformarle part models. In *Wavelet Analysis and Pattern Recognition (ICWAPR), 2012 International Conference on*, pages 202–207. IEEE.

[Li, 2013] Li, Y. (2013). *Stereo vision and Lidar based dynamic occupancy grid mapping: Application to scenes analysis for intelligent vehicles.* PhD thesis, Université de Technologie de Belfort-Montbeliard.

[Lindeberg, 1998] Lindeberg, T. (1998). Feature detection with automatic scale selection. *Int. J. of Computer Vision.*

[Lowe, 1999] Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proc. ICCV*.

[Lowe, 2004] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (IJCV)*.

[Ma and Grimson, 2005] Ma, X. and Grimson, W. E. L. (2005). Edge-based rich representation for vehicle classification. In *Proc. 10th IEEE Int. Conf. Comput. Vis.*

[Magnenat et al., 2010] Magnenat, S., Longchamp, V., Bonani, M., Retornaz, P., Germano, P., Bleuer, H., and Mondada, F. (2010). Affordable SLAM through the co-design of hardware and methodology . In *IEEE International Conference on Robotics and Automation*.

[Mair et al., 2010] Mair, E., Hager, G. D., Burschka, D., Suppa, M., and Hirzinger, G. (2010). Adaptive and generic corner detection based on the accelerated segment test. In *Proceedings of the European Conference on Computer Vision (ECCV'10)*.

[Mallot et al., 1991] Mallot, H., Bulthoff, H., Little, J., and Bohrer, S. (1991). Inverse perspective mapping simplifies optical flow computation and obstacle detection. *Biological Cybernetics*.

[Marek Kraft and Kasinski, 2008] Marek Kraft, A. S. and Kasinski, A. J. (2008). High-speed image feature detection using FPGA implementation of fast algorithm.

[Márquez-Gámez, 2012] Márquez-Gámez, D. A. (2012). *Towards visual navigation in dynamic and unknown environment: trajectory learning and following, with detection and tracking of moving objects*. PhD thesis, Toulouse, INSA.

[Masoud and Papanikolopoulos, 2004] Masoud, O. and Papanikolopoulos, N. P. (2004). Using geometric primitives to calibrate traffic scenes. In *Proc. IEEE IROS*.

[Maybeck, 1979] Maybeck, P. (1979). *Stochastic models, estimation and control.* Mathematics in Science and Engineering.

[Messelodi et al., 2005] Messelodi, S., Modena, C. M., and Zanin, M. (2005). A computer vision system for the detection and classification of vehicles at urban road intersections. *Pattern Anal. Appl.*

[Mikolajczyk and Schmid, 2005a] Mikolajczyk, K. and Schmid, C. (2005a). A performance evaluation of local descriptors. In *IEEE Trans. Pattern Anal. Mach. Intell.*

[Mikolajczyk and Schmid, 2005b] Mikolajczyk, K. and Schmid, C. (2005b). A performance evaluation of local descriptors. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE (PAMI)*.

[Mindru et al., 2004] Mindru, F., Tuytelaars, T., Gool, L. V., and Moons, T. (2004). Moment invariants for recognition under changing viewpoint and illumination. *Computer Vision and Image Understanding*.

[MobilEye, 2009] MobilEye (2009). SeeQ2™.

[Mohammad Awrangjeb and Fraser, 2012] Mohammad Awrangjeb, G. L. and Fraser, C. S. (2012). Performance comparisons of contour-based corner detectors.

[Montemerlo et al., 2002] Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. (2002). FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *National Conference on Artificial Intelligence*.

[Montiel, 2006] Montiel, J. (2006). Unified inverse depth parametrization for monocular slam. In *Proc. Robotics: Science and Systems (RSS)*.

[Mur-Artal et al., 2015a] Mur-Artal, R., Montiel, J., and Tardos, J. (2015a). Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics (ITRO)*.

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

[Negri et al., 2007] Negri, P., Clady, X., and Prevost, L. (2007). Benchmarking haar and histograms of oriented gradients features applied to vehicle detection. In *ICINCO-RA (1)*, pages 359–364.

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

[Nikolic et al., 2014] Nikolic, J., Rehder, J., Burri, M., Gohl, P., Leutenegger, S., Furgale, P., and Siegwart, R. (2014). A synchronized visual-inertial sensor system with FPGA pre-processing for accurate real-time SLAM. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[Nowak et al., 2006] Nowak, E., Jurie, F., and Triggs, B. (2006). Sampling strategies for bag-of-features image classification. In *Proc. European Conference on Computer Vision (ECCV)*.

[NVIDIA, 2015a] NVIDIA (2015a). Cuda 7.0 Jetson TX1 performance and benchmarks.

[NVIDIA, 2015b] NVIDIA (2015b). NVIDIA® Jetson™ TX1 Supercomputer-on-Module Drives Next Wave of Autonomous Machines.

[Opelt, 2006] Opelt, A. (2006). *Generic Object Recognition*. PhD thesis, Graz Univ. Technol., Styria, Austria.

[Opelt et al., 2006a] Opelt, A., Pinz, A., Fussenegger, M., and Auer, P. (2006a). Generic object recognition with boosting. In *IEEE Trans. Pattern Anal. Mach. Intell.*

[Opelt et al., 2006b] Opelt, A., Pinz, A., and Zisserman, A. (2006b). A boundary fragment model for object detection. In *Proc. Eur. Conf. Comput. Vis.*

[Opelt et al., 2006c] Opelt, A., Pinz, A., and Zisserman, A. (2006c). Incremental learning of object detectors using a visual shape alphabet. In *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*

[OpenCV, 2015] OpenCV (2015). OpenCV release 3.0.0. http://docs.opencv.org/3.0.0.

[Park et al., 2007] Park, K., Lee, D., and Park, Y. (2007). Video-based detection of street-parking violation. In *Proc. Int. Conf. Image Process. CVPR.*

[Platt et al., 1998] Platt, J. et al. (1998). Sequential minimal optimization: A fast algorithm for training support vector machines.

[Qasim et al., 2010] Qasim, S., Telba, A., and AlMazroo, A. (2010). Fpga design and implementation of matrix multiplier architectures for image and signal processing applications. *International Journal of Computer Science and Network Security (IJCSNS)*, 10(2).

[Raphael et al., 2011] Raphael, E., Kiefer, R., Reisman, P., and Hayon, G. (2011). Development of a Camera-Based Forward Collision Alert System. *SAE Int. J. Passeng. Cars – Mech. Syst.*

[Rosten and Drummond, 2005] Rosten, E. and Drummond, T. (2005). Fusing points and lines for high performance tracking.

[Roussillon et al., 2011] Roussillon, C., Gonzalez, A., Solà, J., Codol, J., Mansard, N., Lacroix, S., and Devy, M. (2011). RT-SLAM : A Generic and Real-Time Visual SLAM Implementation. In *8th International Conference on Computer Vision Systems*, Sophia Antipolis (France).

[Rybski et al., 2010] Rybski, P. E., Huber, D., Morris, D. D., and Hoffman, R. (2010). Visual classification of coarse vehicle orientation using histogram of oriented gradients features. In *Intelligent Vehicles Symposium (IV), 2010 IEEE*, pages 921–928. IEEE.

[Sarraf et al., 2007] Sarraf, E., Ahmed-Ouameur, M., and Massicotte, D. (2007). Fpga design and implementation of direct matrix inversion based on steepest descent method. In *Proc. 50th Midwest Symposium on Circuits and Systems (MWSCAS).*

[Schaffalitzky and Zisserman, 2002] Schaffalitzky, F. and Zisserman, A. (2002). Multi-View Matching for Unordered Image Sets. In *Proc. Seventh European Conf. Computer Vision.*

[Sciuto et al., 2002] Sciuto, D., Salice, F., Pomante, L., and Fornaciari, W. (2002). Metrics for design space exploration of heterogeneous multiprocessor embedded systems. In *Proceedings of the Tenth International Symposium on Hardware/Software Codesign (CODES).*

[Serre et al., 2007] Serre, T., Wolf, L., Bileschi, S., Riesenhuber, M., and Poggio, T. (2007). Robust object recognition with cortexlike mechanisms. In *IEEE Trans. Pattern Anal. Mach. Intell.*

[Shaout et al., 2009] Shaout, A., El-mousa, A. H., and Mattar, K. (2009). Specification and modeling of hw/sw co-design for heterogeneous embedded systems.

[Sivic et al., 2005] Sivic, J., Russel, B., Efros, A., Zisserman, A., and Freeman, W. (2005). Discovering objects and their location in images. In *Proc. IEEE Int. Conf. on Computer Vision (ICCV).*

[Solà et al., 2005] Solà, J., Monin, A., Devy, M., and Lemaire, T. (2005). Undelayed initialization in bearing only slam. In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2499–2504.

[Solà et al., 2011] Solà, J., Vidal-Calleja, T., Civera, J., and Montiel, J. (2011). Impact of landmark parametrization on monocular EKF-SLAM with points and lines. *International Journal on Computer Vision*.

[Spivey, 2003] Spivey, J. (2003). Fast, accurate call graph profiling. *Oxford University Computing Laboratory*.

[Sérot et al., 2014] Sérot, J., Berry, F., and Bourrasset, C. (2014). High-level dataflow programming for real-time image processing on smart cameras. *Journal of Real-Time Image Processing*, pages 1–13.

[Stein, 2004] Stein, F. (2004). Efficient computation of optical flow using the census transform. In *26th DAGM Symposium*.

[Strasdat et al., 2010] Strasdat, H., Montiel, J., and Davison, A. (2010). Real-time monocular slam: Why filter? In *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*.

[Sun et al., 2002] Sun, Z., Miller, R., Bebis, G., and DiMeo, D. (2002). A real-time precrash vehicle detection system . In *Proceedings of Sixth IEEE Workshop on Applications of Computer Vision (WACV)*.

[Tertei, 2013] Tertei, D. T. (2013). Histogram of Gradients descriptor applied to vehicles detection. Report in Robotics-Action-Perception research team, LAAS-CNRS.

[Tertei et al., 2016a] Tertei, D. T., Devy, M., and Mekonnen, A. A. (2016a). Hierarchical visual obstacle detection. https://youtu.be/742AG7GEeKw.

[Tertei et al., 2016b] Tertei, D. T., Devy, M., and Mekonnen, A. A. (2016b). Hierarchical visual obstacle detection. https://youtu.be/yzNFrp0jME8.

[Tertei et al., 2016c] Tertei, D. T., Devy, M., and Mekonnen, A. A. (2016c). Hierarchical visual obstacle detection. https://youtu.be/B4g35E5cp4U.

[Tertei et al., 2016d] Tertei, D. T., Devy, M., and Mekonnen, A. A. (2016d). Hierarchical visual obstacle detection. https://youtu.be/MQLdaMO32Lg.

[Tertei et al., 2016e] Tertei, D. T., Devy, M., and Mekonnen, A. A. (2016e). Hierarchical visual obstacle detection. https://youtu.be/cvZqbE3MuLQ.

[Tertei et al., 2014] Tertei, D. T., Piat, J., and Devy, M. (2014). FPGA design and implementation of a matrix multiplier based accelerator for 3D EKF SLAM. In *Proc. European Conference on ReConFigurable Computing and FPGAs (ReConFig)*.

[Thrun et al., 2005] Thrun, S., Burgard, W., and Fox, D. (2005). Probabilistic Robotics. *MIT Press*.

[Tiwari et al., 2013] Tiwari, S., Singh, S., and Meena, N. (2013). Fpga design and implementation of matrix multiplication architecture by ppi-mo techniques. *International Journal of Computer Applications*, 80(1):19–22.

[Turk, 2012] Turk, A. M. (2012). Amazon mechanical turk. *Retrieved August*, 17:2012.

[Ullman, 2007] Ullman, S. (2007). Object recognition and segmentation by a fragment-based hierarchy. *Trends Cognitive Sci.*

[Underwood and Hemmert, 2004] Underwood, K. and Hemmert, K. (2004). Closing the Gap: CPU and FPGA Trends in Sustainable Floating-Point BLAS Performance. In *Proc. IEEE Symp. Field-Programmable Custom Computing Machines.*

[Vedaldi and Fulkerson, 2010] Vedaldi, A. and Fulkerson, B. (2010). Vlfeat: An open and portable library of computer vision algorithms. In *Proceedings of the 18th ACM international conference on Multimedia*, pages 1469–1472. ACM.

[Vincke, 2012] Vincke, B. (2012). *Architectures pour des systèmes de localisation et de cartographie simultanées.* PhD thesis, Université Paris Sud and Institut d'Electronique Fondamentale (IEF), Orsay.

[Vincke et al., 2012a] Vincke, B., Elouardi, A., and Lambert, A. (2012a). Efficient Implementation of EKF-SLAM on a Multi-Core Embedded System. In *IECON.*

[Vincke et al., 2012b] Vincke, B., Elouardi, A., and Lambert, A. (2012b). Real time simultaneous localization and mapping: towards low-cost multiprocessor embedded systems. *EURASIP Journal on Embedded Systems.*

[Viola and Jones, 2001] Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE.

[Wijnhoven and de With, 2009] Wijnhoven, R. and de With, P. H. N. (2009). Comparing feature matching for object categorization in video surveillance. In *Proc. Adv. Concepts Intell. Vis. Syst.*

[Wijnhoven et al., 2008] Wijnhoven, R., de With, P. H. N., and Creusen, I. (2008). Efficient template generation for object classification in video surveillance. In *Proc. 29th Symp. Inf. Theory Benelux.*

[Wijnhoven and de With, 2007] Wijnhoven, R. G. J. and de With, P. H. N. (2007). Experiments with patch-based object classification. In *Proc. IEEE Conf. Adv. Video Signal Based Surv.*

[Wilson et al., 2014] Wilson, C., Zicari, P., Craciun, S., Gauvin, P., Carlisle, E., George, A., and Lam, H. (2014). A power-efficient real-time architecture for SURF feature extraction. In *International Conference on ReConFigurable Computing and FPGAs (ReConFig).*

[Wu and Nevatia, 2007] Wu, B. and Nevatia, R. (2007). Detection and tracking of multiple, partially occluded humans by Bayesian combination of edgelet-based part detectors. *Int. J. Comput. Vis.*

[Xilinx, 2011] Xilinx (2011). All Programmable FPGAs.

[Xilinx, 2015] Xilinx (2015). UltraScale Architecture. http://www.xilinx.com/products/technology/ultrascale.html.

[Xillinux, 2010] Xillinux (2010). Xillinux: A Linux distribution for Zedboard, ZyBo, MicroZed and SocKit.

[Yager and Filev, 1994] Yager, R. R. and Filev, D. P. (1994). Generation of fuzzy rules by mountain clustering. *Journal of Intelligent & Fuzzy Systems*, 2(3):209–219.

[Zhang and Vela, 2015a] Zhang, G. and Vela, P. (2015a). Good features to track for visual slam. In *Proc. IEEE International Conference on Computer Vision and Pattern Recognition (CVPR).*

[Zhang and Vela, 2015b] Zhang, G. and Vela, P. (2015b). Optimally observable and minimal cardinality monocular slam. In *Proc. IEEE International Conference on Robotics and Automation (ICRA).*

[Zhang et al., 2005] Zhang, W., Yu, B., Zelinsky, G. J., and Samaras, D. (2005). Object class recognition using multiple-layer boosting with heterogeneous features. In *IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recog.*

[Zhuo and Prasanna, 2007] Zhuo, L. and Prasanna, V. (2007). Scalable and Modular Algorithms for Floating-Point Matrix Multiplication on Reconfigurable Computing Systems. In *IEEE Transactions on Parallele and Distributed Systems, Vol. 18, No. 4.*