Implementation of a distributed bio-inspired neural architecture on FPGA

Tarek Elouaret

CY Cergy Paris University - École doctorale n° 405 Économie, management, mathématiques, physique et sciences informatiques (EM2PSI)

Implémentation d'une architecture neuronale bio-inspirée distribuée sur FPGA

by

Tarek Elouaret

This dissertation is submitted for the degree of
Doctor of Philosophy from CY Cergy Paris University

Catherine Dezan	Bretagne Occidentale Université	Rapporteure
Jean-Christophe Prévotet	INSA de Rennes	Rapporteur
Virginie Fresse	Université de Saint-Etienne	Présidente du jury
Andrea Pinna	Sorbonne Université	Examinateur
Olivier Orfila	VEDECOM	Examinateur
Stéphane Zuckerman	CY Cergy-Paris Université	Encadrant
Lounis Kessal	ENSEA	Directeur
Olivier Romain	CY Cergy-Paris Université	Co-Directeur

Submitted: April, 5th, 2023

English Abstract

Implémentation d'une architecture neuronale bio-inspirée distribuée sur FPGA

Abstract:

Autonomous vehicles heavily rely on efficient self-localisation mechanisms. Cameras are the most common kinds of sensors to perform this task: they provide rich input at very low cost. The computational intensity of visual localisation varies depending on the complexity of the environment surrounding the vehicle, which influences which real-time task should run, and when. Real-time image processing in this context requires fast processing, as well as the ability to store enough information to take decisions in an energy-efficient manner, especially in the case of electric vehicles. Hence, leveraging FPGAs is a natural answer to many issues raised by such an application, in order to facilitate application prototyping and estimate the proper energy savings. We introduce a distributed solution to implement a large bio-inspired visual localisation model. The proposed workflow is comprised of the following: (1) an image processing IP which yields the pixels information per each visual landmark detected in each captured image; (2) an implementation of N-LOC, a bio-inspired neural architecture on FPGA board; (3) A distributed version of N-LOC and its evaluation over a single FPGA, as well as a design to be used on a multi-FPGA platform with gigabit transceivers to link the various substrates. Comparisons with a pure software solution show that our hardware-based IP implementation yields up to 9× lower latency times, 7× higher throughput (frames/second) than the reference software implementation, and power footprint as low as 2.741W for the whole system, *i.e.*, up to 5.5-6× less than what Nvidia Jetson TX2 consumes on average.

Keywords:
FPGA, bio-inspired algorithms, neural networks, hardware acceleration, GTX Transceiver

French Abstract

Implémentation d'une architecture neuronale bio-inspirée distribuée sur FPGA

Résumé:
Les véhicules autonomes utilisent des mécanismes d'auto-localisation efficaces pour se déplacer sans l'intervention humaine. Les caméras sont les capteurs les plus courants pour effectuer cette tâche : elles fournissent des données riches à très faible coût.

L'intensité calculatoire de la localisation basé sur caméra varie en fonction de la complexité de l'environnement entourant le véhicule. Ce qui influe sur la tâche en temps réel à exécuter et sur le moment où elle doit être exécutée. Le traitement d'images en temps réel dans ce contexte exige un traitement rapide, ainsi que la capacité de stocker suffisamment d'informations pour prendre des décisions de manière économe en énergie, notamment dans le cas des véhicules électriques.

Par conséquent, l'utilisation des FPGA est une réponse explicite à des nombreux problèmes soulevés par une telle application, afin de faciliter le prototypage de l'application et d'estimer les économies d'énergie appropriées.

Nous présentons ici une solution distribuée originale pour mettre en œuvre un grand modèle de localisation visuelle bio-inspiré. Le projet de travail pour notre collaboration est composé et structuré comme suivants : (1) une IP de traitement d'image qui produit les informations de pixels par chaque repère visuel détecté dans chaque image capturée ; (2) une implémentation de N-LOC, une architecture neuronale bio-inspirée sur une carte FPGA ; (3) une version distribuée de N-LOC et son évaluation sur un seul FPGA, ainsi qu'une conception destinée à être utilisée sur une plate-forme multi-FPGA avec des émetteurs-récepteurs gigabit GTX pour relier les différents substrats.

La comparaison avec une solution purement logicielle montre que notre mise en œuvre matérielle de l'IP permet d'obtenir des temps de latence jusqu'à 9× fois inférieurs, un débit (images/seconde) 7× fois supérieur à celui de la mise en œuvre logicielle de référence, et une empreinte énergétique aussi faible que 2,141 W pour l'ensemble du système, soit jusqu'à 5.5-6× fois moins qu'un système embarqué basé sur une plateforme telle que la Nvidia Jetson TX2 effectuant le même calcul en moyenne.

Mot-clefs:
FPGA, bio-inspired algorithms, neural networks, hardware acceleration, GTX Transceiver

Publications

Journals

- **T. Elouaret**., S.Colomer, S. Zuckerman, F. Demelo, N. Cuperlier, O. Romain, & L. Kessal. *Implementation of a bio-inspired neural architecture for autonomous vehicle on a multi-FPGAs platform.* MDPI Sensors journal. MDPI DOI: https://doi.org/10.3390/s23104631

International conferences

- **T. Elouaret**., S.Colomer, S. Zuckerman, F. Demelo, N. Cuperlier, O. Romain, & L. Kessal. *Implementation of a bio-inspired neural architecture for autonomous vehicle on a reconfigurable platform.* In 2022 Alaska/ Electronic System on Chip and Real Time Embedded Control (ISIE22), pages 1–6, June 2022. IEEE DOI: https://ieeexplore.ieee.org/abstract/document/8881834

Workshops

- **T. Elouaret**., S. Zuckerman, L. Kessal, Y. Espada, N. Cuperlier, G. Bresson, F. B.Ouezdou, & O. Romain. *Position paper: Prototyping autonomous vehicles ap-plications with heterogeneous multi-fpgasystems.* In 2019 UK/ China Emerging Technologies (UCET), pages 1–2, Aug 2019. IEEE DOI: https://ieeexplore.ieee.org/abstract/document/8881834

Contents

List of Figures

List of Tables

Preface

My most sincere gratitude goes to my supervisor Dr. Stephane Zuckerman, who, through his intellectual rigor, his always relevant comments, his knowledge of the subject and methodological aspects, as well as her great availability, his largely contributed to my training in educational research. The constant monitoring she provided at all stages of this doctoral research made this project a reality.

To my thesis director Prof. Olivier Romain, and Dr. Lounis Kessal, a great thanks for their contribution to the realisation of this thesis through their remarks which are always very enlightening, their mastery of various fields and great experience. The critical opinions expressed by my co-supervisors allowed me to deepen my reflection and to take a retrospective look at this research. My work was initiated thanks to the INEX Paris-Seine AAP 2018 project, as well as the financial support of VEDECOM institute in Versailles for their help in the establishment of this project.

I would like to thank the members of the jury individually for doing me the honor of judging this work. I thank them for their careful reading of my thesis as well as for the comments they will address to me during this defense in order to improve my work. Special thanks to Prof. Catherine Dezan and Prof. Jean-Christophe Prévotet who followed me during the monitoring committees of my thesis. It is thanks to their comments that I was able to improve several weak points to which I had not paid attention. During these years I was part of the CELL team of the ETIS laboratory. I thank all the members of CELL who gave me a warm welcome and with whom I had various discussions. I consider of course also those that I have crossed regularly.

Last but not least, I would also like to acknowledge by deeply appreciating the invaluable support (financial, emotional, and intellectual) that my mother FATIMA ZOHRA has given me through my life. I would especially like to thank my sister Fairouz Judy who has believed in me and have always encouraged me to reach the goal. I would like to thank them here for their contribution, without which the realisation of this research would have been impossible.

Part I

General Introduction

Nowadays, new technologies in the communication and robotics fields, have had a substantial influence on our daily life which transportation is no exception. These technologies have given a high increase to the prospect of autonomous vehicle (AV) technology which aims to minimise and mitigate crashes, energy consumption, pollution, and congestion while at the same time increasing transport accessibility [55]. Even though the idea of driverless vehicles has been around for decades, the exorbitant costs have prevented large-scale production [55]. Besides, there has been an acceleration in the research and development efforts in the last decade to bring the idea of the AV to possession [55]. For instance, the advent of the Google car brought AVs to the spotlight [66, 93]. Moreover, the automotive industry spends around €77 billion worldwide on R&D in order to nurture innovation and to stay competitive [74, 105].

The primary motivation behind the research and development of AVs are the need for increased driving safe, an increasing population that also leads to an increase in vehicles on the road, expanding of the infrastructure, the comfort and ease of relying on machines for tasks like driving, and the demand for optimisation of resources and time management. With the population growing, a considerable impact has been created on our roads, infrastructure, open spaces, fuel stations, and resources [109].

One of the most challenging tasks in environment perception of automated driving is differentiating between individual objects, vehicle tracking, self-localisation, pedestrian detection, predicting trajectories for unknown paths by generalisation, and understanding traffic patterns [109]. Figure 1 represents the positioning of different sensors on a vehicle.

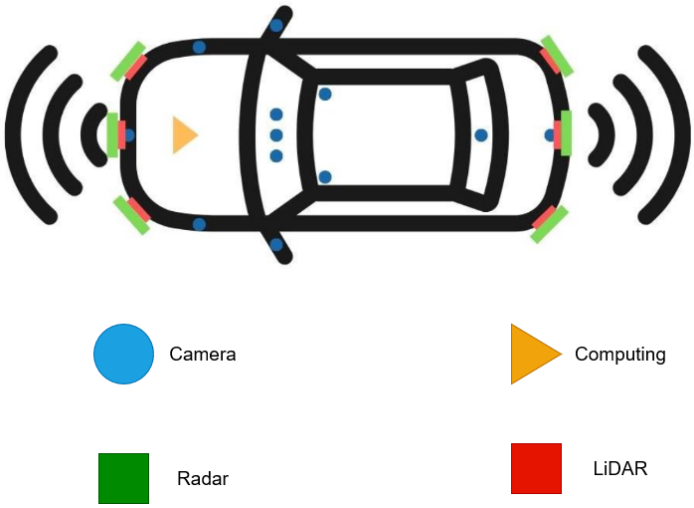


Figure 1: The principle type of sensors in autonomous vehicle (AV).

Electric autonomous vehicles have been at the centre of attention of robotics and embedded systems fields in the last decade [42]. Current solutions tend to promote LIDAR cameras for medium-range localisation, coupled with convolutional neural networks (CNN) [77]. However, leveraging LIDAR cameras for localisation is expensive and energy-consuming. Hence, a neurorobotics architecture modelling mammal's hippocampus was proposed by Espada *et al.* [53] as a counterpart to what exists in the localisation & navigation approaches in the state of art. In particular, the computational intensity varies, depending on the environment surrounding the vehicle (e.g., night vs. day; rural vs. urban; etc.). This leads to changes in terms of which real-time task(s) should run. However, Colomer *et al.* [43] demonstrated in their last work that the LPMP model leads to the same considerable computational intensity wherever the vehicle could be found.

The resulting neural network relies on fewer layers than now-current CNNs, thus simplifying the overall neural architecture, but at the cost of a potentially much higher neuron count. This solution is LIDAR-free and has shown its ability to correctly assess its environment. However, it does not scale: at medium to high speeds, a pure software implementation is not adequate. However, a custom hardware solution is very likely to succeed. In particular, the computational intensity varies, depending on the environment surrounding the vehicle (e.g., night vs. day; rural vs. urban; etc.). This leads to changes in terms of which real-time task(s) should run.

This work revolves around implementing parts of this navigation phase onto a unique multi-FPGA platform, Wizarde, which is a 3×3 tile grid. Each tile contains a system-on-chip, which combines a multicore processor and high-performance FPGA. Adjacent tiles are linked by high-bandwidth transceivers.

Hence, the amount of the workload must be adequately load-balanced across Wizarde's tiles, while at the same time respecting power and energy constraints, as well as real-time constraints.

In addition, it is very likely that hardware tasks will model various sizes of neuron clusters, thus requiring varying needs of hardware resources.

Hence, resizing the whole application, task placement and data locality are important factors, to properly map the application's neural network onto the multi-FPGAs fabric, and efficient scheduling algorithms must be proposed to use, which take these constraints into account.

Therefore, Being able to process images in a real-time context requires not only fast processing means, but also the ability to store enough information to take decisions, as well as means of saving energy as much as possible. As a result, leveraging reconfigurable fabrics is a solution to many issues raised by such an application. However, to ensure energy is properly managed, it is paramount to power on as little of the circuit as possible: everything powered on should be for a good reason.

Eventually, exposing Wizarde as a unique prototyping board combining several Zynq system-on-chips which allows for the configuration of multiple FPGAs, driven by several multi-core processors, Hence, we intend to leverage a heterogeneous platform, composed of high-end general-purpose embedded processors and FPGAs. This will allow us to provide specialised circuits to process specific parts of the NN. The FPGAs will need to carry out several tasks: extraction of salient points in images [57], as well as maintaining and updating a large NN implemented in hardware, we use the LPMP model proposed by [43]. Because the NN will be large, we will suggest such a scenario with an off-line scheduling policy, in order to deploy the NN application on multiple FPGAs, relying on the Cortex A9s to perform local and distributed hardware and software task scheduling.

How to read this manuscript

To address the different challenges previously mentioned, this manuscript was divided into several chapters, illustrating each part of this research work.

Part I (General introduction): Presents the aim concerning the motivation behind the research topic for my dissertation. Along with some concepts and definitions of the targeted application with the related and needed tools.

Part II (background & state of the art): Presents the related works on the autonomous vehicle in the recent era such as how can we provide both localisation and mapping aspects into it. We explore different research in the state of the art of how we could incorporate those aspects to propose a real-time localisation process with a high-security level, as well as confidence. We demonstrate some of the artificial intelligence, neural network, and especially bio-inspired neural network algorithms. Finally, we highlight the issues addressed and the chosen approach to address them.

Part III (contributions): Introduces the bio-inspired neural network model called N-LOC that we aim to prototype, and implement onto the FPGA board. We also show some results related to this implementation, like the latency, accuracy, and resource consumption using different parameters required by each different scenario.

Within that, a proposed scenario as a strategy for the deployment of the bio-inspired neural architecture onto multiple FPGAs called Wizarde, by using GTX transceivers as hardcore resources provided by Xilinx vendors.

Part IV (General conclusion): A general conclusion is made, summarizing the main contributions of this work. Perspectives are also drawn, providing some elements for future improvements and addressing the future work to be done in order to fully implement and evaluate the application, and further by incorporating it into a real mobile robot.

Funding

This work is supported by the VEDECOM Institute and CY University's INEX program $\#2017 - 2182017 - 2018 - C01 - A0$..

Part II

Background and state of the art

1

Background

1.1 Generalities on autonomous vehicles, their applications, and limitations

In this chapter, we present the motivation of this dissertation, by providing some background and literature on the target application, which is self-driving cars with the specific feature of driving delegation. We then show the necessity of both localisation and navigation in this environment exploration. Hence, the localisation task will be our main focus in this thesis, alongside using monocular cameras instead of LiDARs, due to their high energy consumption and financial cost as will be depicted later in this chapter.

We then present some different types of algorithms to perform the localisation like the bio-inspired neural architecture which is a novel approach, in contrast to what state-of-art proposes like CNN and/or DNN.

Autonomous vehicles have been at the centre of attention in robotics and embedded systems fields in the last decade. Research and development work on autonomous vehicles (AVs) has accelerated in recent years, driven by technological advances in both hardware and software. However, a number of significant challenges must be addressed before the widespread adoption of AVs. These include legal and regulatory issues, public perceptions of AV technologies, cost, accuracy and reliability of onboard sensing equipment, computational constraints and the limitations of current algorithms and systems for tasks such as environment perception, path planning and control.

Environment perception is arguably one of the most important fields for future AV development; accurate perception of the environment allows automated driver assistance systems (ADASs) to perform an informed decision during functions such as adaptive cruise control, lane changing, parking and obstacle and collision avoidance [92]. Components of the environment which must be detected by an ego vehicle include: traffic signals and signs, road markings, lane and junction topology and other road users including vehicles, cyclists and pedestrians [92].

To provide well-trusted security, AVs must rely upon reliable solutions. According to a recent technical report by the National Highway Traffic Safety Administration, 94% of road accidents are caused by human errors [131]. Thus, automated Driving solutions (ADSs) are considered to be reliable solutions and are being implemented with the promise of reducing accident rates, with an ease-of-use of driving vehicles [46]. They require precise knowledge of their position and orientation in all weather and traffic conditions for path planning, perception, control, and general safe operation [46].

The challenge facing localisation for autonomous systems in terms of required accuracy and reliability at scale is unprecedented [117]. Reid *et al.* in [117] shows that AVs require decimeter-level positioning for motorway operation and near-centimetre level for operation on local and

residential streets (Residential street means a subdivision street adjacent to a property that is anticipated to develop as single-family residences, apartment buildings, or other similar dwelling structures [13]). These requirements trunk from one target: ensure that the vehicle knows it is within its lane. Horizontally, this is broken down by lateral (side-to-side) and longitudinal (forward-backward) components [117]. Vertically, the vehicle must know what road level it is on when located among multi-level roads. At a certain point, the vehicle will have an assessment of its maximum position error in each direction. These are known as protection levels [117]. These alert limits are design variables which are needed to be sufficiently small to guarantee that the vehicle remains within its lane at all times. If the protection level is larger than the alert limit at any given time, there is less certainty the vehicle will remain within its lane. The challenge of decimeter location accuracy is presented in perspective by table 1.1 which shows the progress in localisation throughout the last century.

System	Years Active	Horizontal Accuracy [m]	Latency	Fix Type	Coverage
Celestial-Chronometry	1770–1920	3,200	Hours	2D	
LORAN-C	1957–2010	460	None	2D	North America, Europe
Transit	1964–1996	25	30–100 min	2D	Global
GPS	1995–Present	3	None	3D	Global

Table 1.1: Evolution of localisation accuracy in the last century. Data collected by Reid *et al.* [117].

We are interested in autonomous electrical vehicles EVs because they offer the potential to substantially decrease carbon emissions from both the transportation and power generation sectors of the economy [118]. The mass adoption of EVs is expected to have a significant and beneficial impact in several ways, including the ability to assist in the integration of renewable energy into existing electric grids [118]. Alternative vehicle technologies, such as EVs, are being developed to reduce the world's dependence on oil for transportation and limit transportation-related CO_2 emissions [118]. Likewise, renewable energy sources are being developed and deployed to displace fossil fuel-based electricity generation, reducing greenhouse gas emissions as well as the emission of other pollutants such as nitrous oxides (NO_x) and sulfur dioxide (SO_2) [118]. Hence, the integration of the transportation and electricity sectors, in combination with EVs and renewable energy, offers the potential to drastically decrease the world's dependence on fossil fuels and the consequent emission of greenhouse gases [118].

According to what we have presented and highlighted previously, EVs are a promising choice for a more ecological environment. In the remainder of this thesis, localisation will be our main focus and field of research during our thesis and project as part of the EVs.

1.2 Common tools for autonomous vehicles localisation

The localisation as presented and described in the section 1.1 and in the table 1.1 is thus a key capability for any AVs to be able to navigate properly in any well-known environment. Current solutions tend to combine multi-modal sensors (light detection and ranging LiDAR, cameras, radars, *etc.*) to perform an end-to-end environmental localisation task, coupled with convolutional neural networks (CNNs as core-process to take the decision) [77].

Among techniques used in mapping, especially for localisation are the landmarks or points of interest of an image, which are used as primary features in an urban environment, with the use of a stereo camera system [132]. Spangenberg *et al.* presented the resulting map representation, allowing for easy storage and online updates See 1.2. The localisation is carried out in real-time by a stereo camera system as the main sensor, using vehicle odometry and an off-the-shelf GPS as secondary information sources, along with a particle filter approach, coupled with a Kalman filter for robustness and sensor fusion [132]. This leads to a lateral accuracy below 20 cm in various urban test areas (according to Levinson *et al.* [84].). The proposed system has been validated in such autonomous test vehicles and successfully demonstrated the full loop from mapping to autonomous driving (specifically in an urban environment).

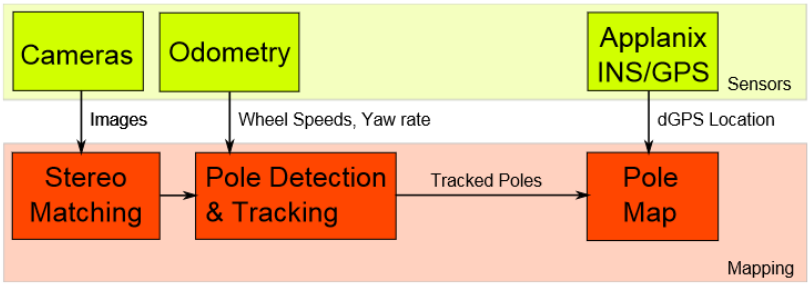


Figure 1.0: mapping dataflow.

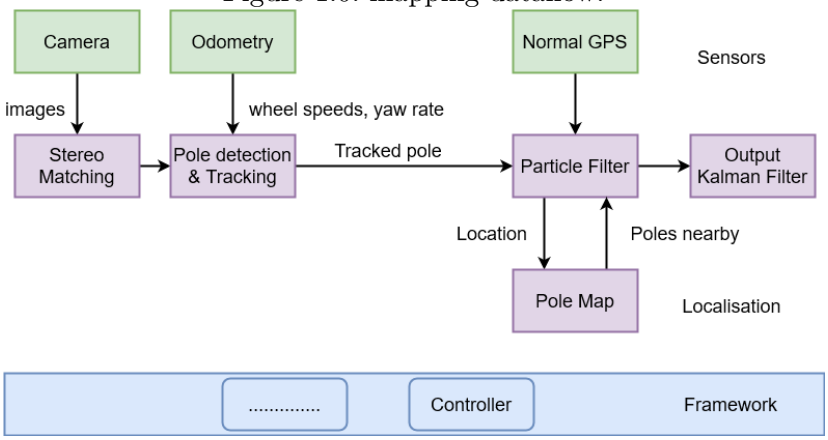


Figure 1.1: localisation dataflow.

Figure 1.2: Mapping and Localisation dataflow pipeline used in the implementation project by Levinson*et al.* [84].

Levinson *et al.* in [84] have demonstrated based on their approach to landmarks detection called pole-like structures (described on 1.2), it can serve as a reliable approach in urban scenarios for autonomous driving. The realised accuracies were sufficient in various scenarios and allowed for smooth autonomous control behaviour. Increasing the availability of the localisation to a comprehensive approach by the integration of further landmark classes is a future goal. The integration of the map building into a large-scale SLAM method is another extension, which is explicitly related to the question of efficient map updates to maintain them accurately. The average accuracy in the night, and day modes, is about $\approx 91 \cdot \%$.

Wang *et al.* in [144] propose another approach for precise and robust localisation. That method is proposed to precisely localise the autonomous vehicle using a 3D-LIDAR sensor. First, a curb detection algorithm is performed (without any assumption for the type of road

in which the AV is navigating). Next, a beam model is utilised to extract the contour of the multi-frame curbs and to eliminate the most of outliers. Then, the iterative closest point algorithm and two Kalman filters are employed to estimate the position of autonomous vehicles based on the high-precision map [143]. However, the proposed algorithm may break to locate the vehicle correctly when the obstacle blocks the curb (according to Wang *et al.* in [143]). The proposed method also has the limitation of locating the vehicle where there is no curb, also the 3D-LIDAR sensor consumes highly the battery in the case of incorporating it in the EVs. The results of the performance evaluation presented in this work showed that CAET (content-aware video encoding technique) increased the bitrate by 2.9% over the main existing segmentation schemes in the field of distributed encoding, but it also increased the encoding speed by 15.3% and improved the overall performance efficiency.

Figure 1.3 shows the proposed curb-map-based localisation algorithm. First, an algorithm of curb detection is performed on the point cloud of a single frame. The detected curbs are densified by projecting former curbs into the current vehicle coordinate system. Then, the beam model is applied to extract the contour of the densified curbs. Finally, the generated contour is matched to the high-precision map by ICP algorithm [143]. The framework of the proposed algorithm is shown in figure 1.3.

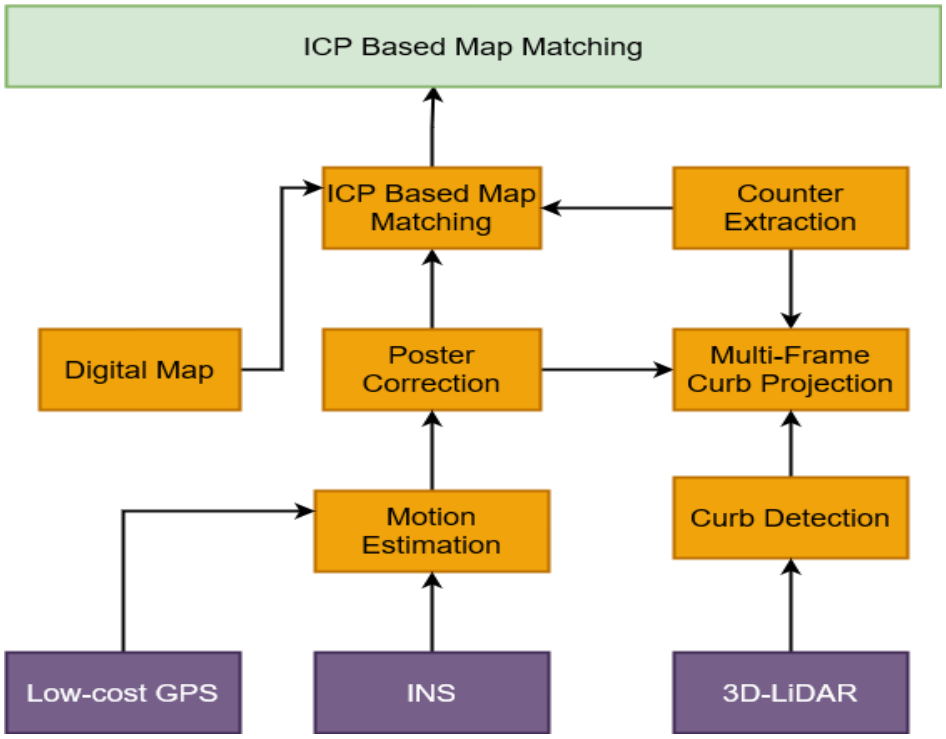


Figure 1.3: Framework of the proposed algorithm, recreated from Wang *et al.* work in [143].

The common sensor on these platforms, a three-dimensional (3D) light detection and ranging (LIDAR) scanner, which generates dense point clouds with measures of surface reflectively-which other state-of-the-art localisation methods have shown are capable of centimetre-level accuracy. Alternatively, in order to obtain comparable cheaper localisation accuracy, Wolcott *et al.* in [145] proposed to localise a single monocular camera within a 3D prior ground-map, generated by a survey vehicle equipped with 3D LIDAR scanners. The system exploited a graphics processing unit (GPU) to generate several synthetic views of the belief environment [145].

They proposed an exploitation of 3D prior maps (an offline mapping stage, which generates

the map to be used for online localisation), augmented with surface reflectivities constructed by a survey vehicle equipped with 3D LIDAR scanners. Afterwards, they move to localise a vehicle by comparing imagery from a monocular camera against several candidate views [145], seeking to maximise normalised mutual information (NMI) (as outlined in figure 1.4).

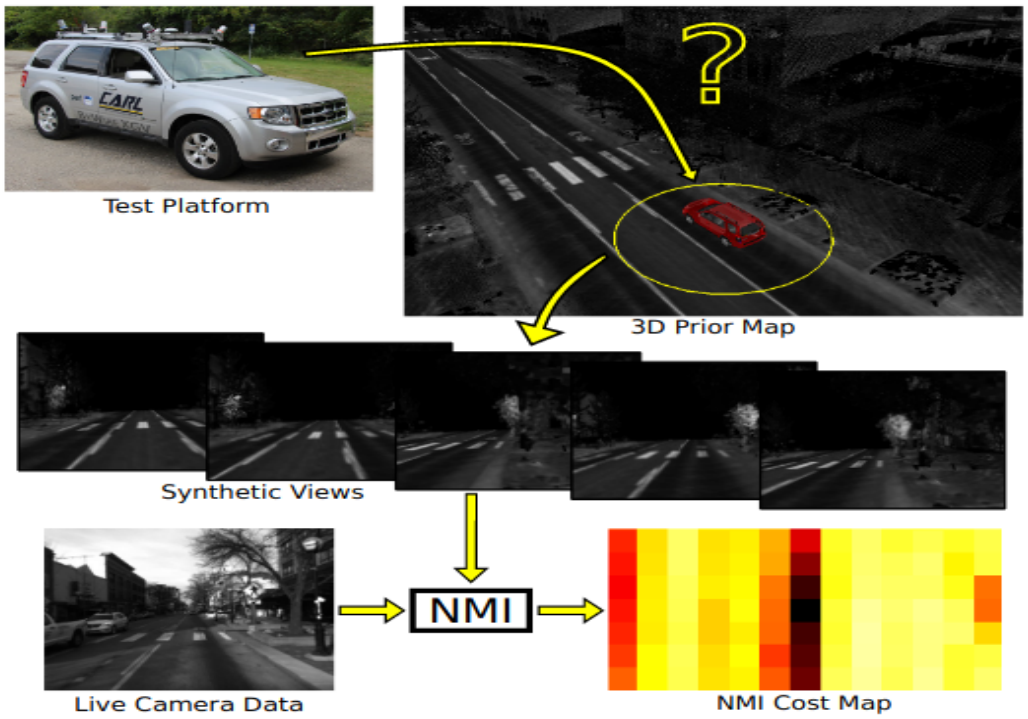


Figure 1.4: Overview of the proposed visual localisation system. The aim is to localise a monocular camera within a 3D prior map (augmented with surface reflectivities) constructed from 3D LiDAR scanners. Given an initial pose belief, after, they generate numerous synthetic views of the environment, which allow them to evaluate using normalised mutual information against the live view from camera imagery. Recreated from Wolcott *et al.* work in [145].

The Authors of this project demonstrated that a single monocular camera can be used as an information source for visual localisation in a 3D LiDAR map containing surface reflectivities. By maximising normalised mutual information, it will be accessible to register a camera stream to the prior map. The system is aided by a GPU implementation, leveraging OpenGL to produce synthetic views of the environment; this implementation is able to give corrective positional updates at ≈10 Hz. Moreover, Wolcott *et al.* in [145] compared their algorithm against the state-of-the-art LiDAR-only automated vehicle localisation, revealing that the proposed approach can achieve a similar order of magnitude error rate, with a sensor that is several orders of magnitude cheaper. Likewise, this can reveal that monocular cameras can deliver accurate information with certain precision into localisation benchmarking. They show then that they are able to achieve longitudinal and lateral root mean square (RMS) errors of 19.1 cm and 14.3 cm, respectively, on the Downtown dataset. They also got longitudinal and lateral RMS errors of 45.4 cm and 20.5 cm, respectively, on the Stadium dataset.

Modern automation requires mobile robots to be robustly localised in complex scenarios. Current localisation systems typically use maps that require to be built and interpreted by experienced operators, growing deployment costs as well as decreasing the adaptability

of robots to rearrangements in the environment. In contrast, architectural floor plans can be understood by non-expert users and typically explain only the non-rearrangeable parts of buildings [35]. Boniardieta *et al.* in [35] proposed a system for robot localisation in architectural CAD drawings. The aim focuses on employing a method for a simultaneous localisation and mapping approach to online augment the floor plan with a map represented as a pose-graph with LiDAR measurements (a pose-graph object stores information for a 2-D pose graph representation. A pose graph includes nodes connected by edges. Each estimated node is connected to the graph by edge constraints that define the relative pose between nodes and the uncertainty on that measurement [12]). Whenever the environment is accurately mapped in the vicinity of the robot, they use the graph to perform relative localisation. Authors thoroughly evaluate the system in challenging real-world scenarios. Experiments demonstrate that the method is able to robustly track the robot pose even when the floor plan shows major discrepancies from the real-world [35].

Figure 1.5 shows the system is localising a robot in a factory-like environment, including situations where the floor plan is fully blocked up by large structures.

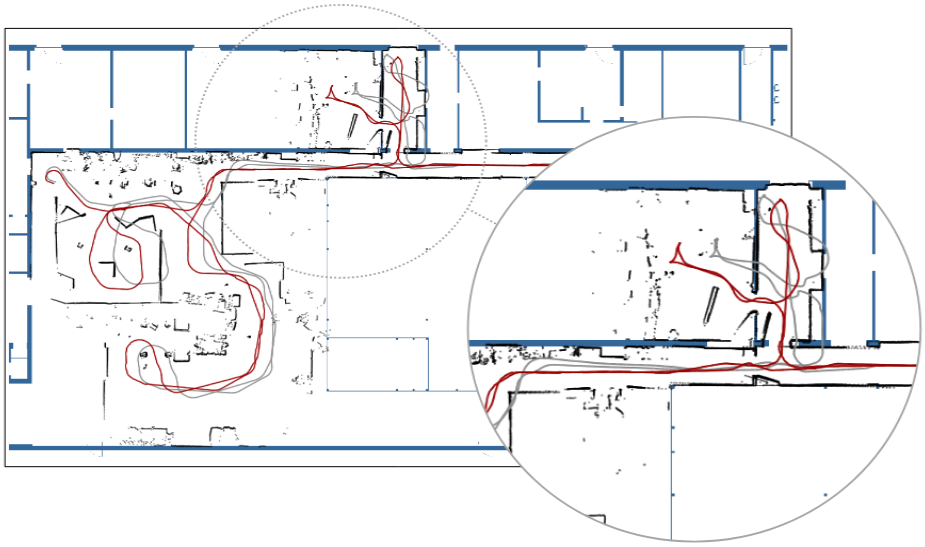


Figure 1.5: The trajectory obtained with the proposed localisation system (red) in an architectural floor plan (blue) of a factory-like scenario. The map of LiDAR observations (black) shows also structures not represented in the floor plan. The map is aligned online to the CAD drawing to localise the robot. The system works robustly even when the floor plan is fully occluded and in situations where Monte Carlo Localisation (grey) fails [35]. Recreated from Boniard *et al.* work in [35].

Furthermore, Boniardieta *et al.* in [35] presented a LiDAR-based system for localisation in architectural floor plans, while the robot is moving through its environment, it uses a SLAM method to online generate a map, represented in figure 1.5 as a pose-graph with LiDAR readings. The priors for the trajectory are computed with the proposed GICP-based scan-to-map-matcher to fit the generated pose graph onto the floor plan. When the map sufficiently covers the area in the vicinity of the robot, Boniardieta *etal.* estimates its relative pose with respect to the matching nodes of the pose graph without a global optimisation process. This combination makes the system robust to missing information in CAD drawings and also computationally efficient. They evaluated the proposed approach in several real-world scenarios and showed that the method works robustly in complex environments and is as accurate and efficient as common state-of-the-art localisation systems. Through the

experimental evaluation based on different datasets recorded in different buildings, they got a localisation accuracy error of $\approx 9.5\%$mm along the x and y axes and a yaw average error of $\approx 0.47°$. Along with this, they carried a runtime performance on an 8-core 4.0GHz I7 CPU, the elapsed time at each update step and over each entire experiment were ≈ 54ms.

The function of higher visual camera detection and better-performing markings has been studied with Vedecom. LiDAR (Light Detection And Ranging scanning) technology could assist to fill the remaining gaps, as it actively sends out infrared light (IR), that yields reliable images of the road scenario and pavement markings both day and nighttime. To evaluate the opportunities of LiDAR technology for the detection of road markings, 3M Deutschland GmbH, Transportation Safety Laboratory and the University of Applied Sciences in Dresden worked together on a joint research project. All-Weather Elements (AWE), are the latest development of high-performance optics, using high index beads to provide reflectivity with both in dry and wet conditions. It was determined that high-performance markings help to increase the level of detection by both camera and LiDAR sensors [124].

1.3 LiDAR systems detecting pavement marking

Mono or stereo cameras with suitable image processing algorithms have been incorporated in several implementations as sensors for lane detection up to now. Used in highly and fully automated driving functions, these sensors will be one of the sources of information to manoeuvre the car in the centre of the traffic lane. However, there are various situations in which this sensor technology fails, according to Sauter *et al.* [124].

In existing SAE 2 driver assistance systems (the Society of Automotive Engineers (SAE) defines 6 levels of driving automation ranging from 0 (fully manual) to 5 (fully autonomous). These levels have been adopted by the U.S. Department of Transportation [1].), the driver must immediately turn off the systems himself, if the driver must immediately deactivate the systems (if they do not automatically turn off) and regain control of the vehicle. This is the state of the art for autonomous vehicles (AVs), e.g. in Tesla's "Autopilot" system, the driver remains responsible and must always keep control of the system. In the case of the higher level of automation (SAE level 3) of an AV, a period of at least 4 seconds is allowed for the change; in the case of fully automated vehicles (SAE level 4), it is several minutes. This represents a completely new challenge for the detection systems for unambiguous and permanent lane recognition [124]. This cannot be solved with cameras or video sensors alone, especially since it is not a measurement process from a physical point of view. A sensor that is particularly suitable for this application is a laser scanner. The travel time of the infrared radiation (time of flight, ToF) is measured so that a physical connection exists [124]. This is a particularly important point circumstance for the development of safety-critical technologies [124]. In figure 1.6, the potential is illustrated by means of measurement at the University of Applied Sciences Dresden.

Sauter *et al.* in [124], aimed to estimate the performance of different pavement marking products in various conditions by evaluating the detectability with an advanced camera Driving assistance systems (ADAS). They specifically measure the contrast between the road surface and the pavement marking with a camera as an indicator of how easily they would be detected by an ADAS such as a Lane Lane Assist System (LKA) in actual operation. See figure 1.7.

A dedicated methodology has been developed in order to maximise the detection of the pavement marking section by the onboard camera so as to measure accurate contrast values.

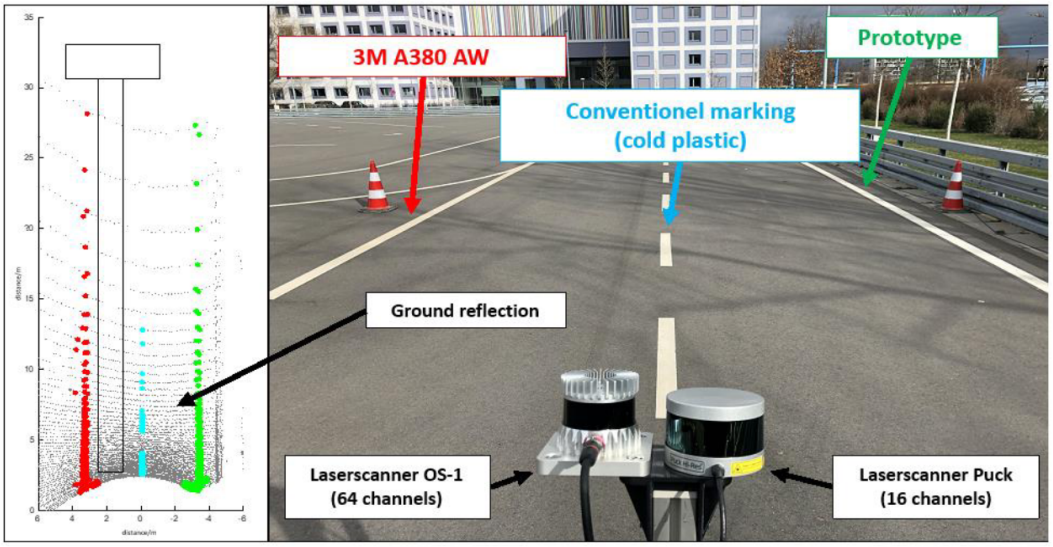


Figure 1.6: LiDAR Retroreflection intensities of different pavement markings. Recreated from Boniard *et al.* work in [35].

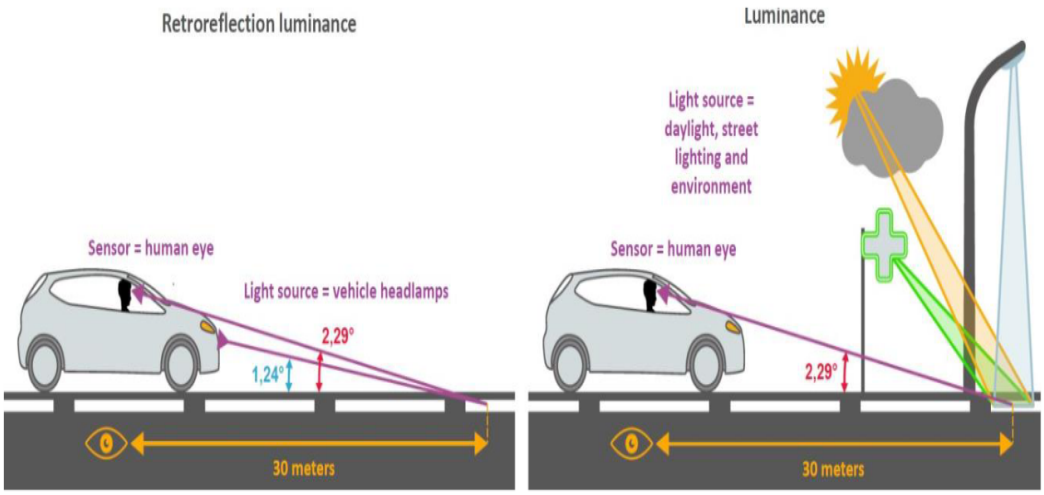


Figure 1.7: Retroreflection luminance and luminance of a pavement marking according to the CEN standard EN 1436. Recreated from Sauteret *et al.* work in [124].

The vehicle camera system and the algorithms were modified in order to acquire data. See figure 1.8.

Different state-of-art research has been dissected within this chapter. As seen before, a lot of projects tend to use LiDAR camera tool-based solution to navigate over different types of environments (wet, dry, rural, and urban). Benterki *et al.* [29], Espada *et al.* [53], and Colomer *et al.* [43] put forward an alternative solution to minimise the power consumption of the electric vehicle with driver delegation.

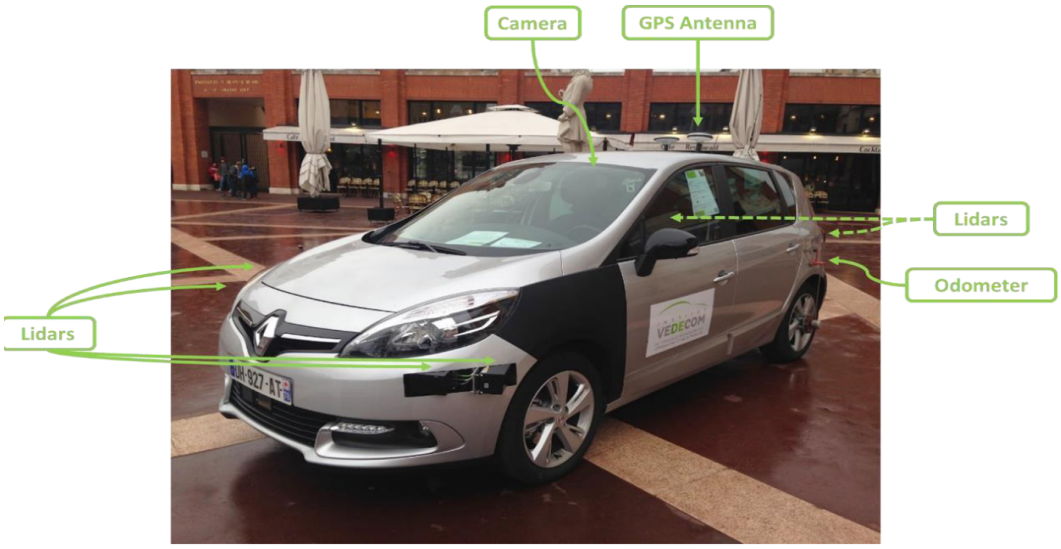


Figure 1.8: Picture of the perception vehicle Recreated from Sauteret *et al.* work in [124].

1.4 Processing and deciding: Traditional AI systems for autonomous vehicles

1.4.1 Artificial neural network and their beneficial

Artificial intelligence AI has become an essential component of Autonomous vehicles (AV) for perceiving the surrounding environment and making the appropriate decisions in motion. To achieve the goal of full automation (i.e., self-driving), it is important to know how AI works in AV systems. Existing research has made great efforts in investigating different aspects of applying AI in AV development [89]. Mhafuzulet *et al.* [71] their main idea is to shorten the gap by providing a comprehensive study in this research avenue. Specifically, it intends to analyse the use of AIs in supporting primary applications such as perception, localisation and mapping, and decision-making.

The performances of robotic localisation systems rely on their ability to continuously build a stable and accurate representation of their environment, according to Yurtsever *et al.* [150]. Besides, building such a representation remains a challenge for autonomous cars that have to deal with large and dynamic environments, since they are intended to be deployed over long periods of time in environments of several tens of kilometres. Even on a daily scale, changing conditions such as light variations, the transient presence of vehicles or pedestrians, and unpredictable changes in the urban landscape (road works) particularly affect the perception of space [151]. you

1.4.2 Convolutional and Deep Neural Networks

Deep learning (DL), a branch of machine learning (ML) algorithms, is inspired by the biological process of neural networks, and it is the most effective, supervised, time and cost-efficient ML approach.

In DL techniques, there is direct learning from the data for all aspects of the model [78]. The lowest level features characterise a suitable representation of the data, It then provides

higher-level abstractions for each of the specific problems in which it is applied [78]. Thus, DL becomes more advantageous when the amount of training data is increased. The development of DL models has grown with the increase in the software and hardware infrastructure [78].

DNNs have made a prestigious breakthrough with appreciable performance in a wide variety of applications. Among architectures that interest us and especially in face recognition, object detection, classification domains, is the convolution neural network (CNN) [48, 147]. A deep neural network architecture with 2 hidden layers in it, See 1.9.

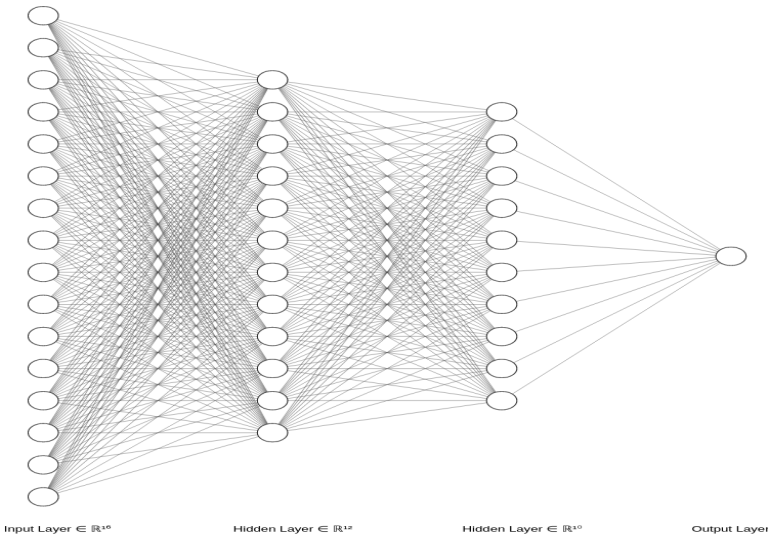


Figure 1.9: overview description of a simple artificial neural network composed of input, two hidden layers, and final output for needed results.

The two principal key factors on which DL methodology is based are :

- Nonlinear processing in multiple layers or stages
- supervised or unsupervised learning

Nonlinear processing in multiple layers refers to a hierarchical method in which the present layer accepts the results from the previous layer and passes its output as input to the next layer. Hierarchy is established among layers so as to organise the importance of the data. This is unlike supervised and unsupervised learning which are linked to the class target label. Its availability means a supervised system and absence indicates an unsupervised system. See 1.10.

1.4.3 Convolutional Neural Network (CNN) architecture

A CNN is a multi-layer neural network and is based on the animal visual cortex [48]. The first CNN was developed by Le Cun *et al.* [80]. The application areas of CNN include mainly image classification & recognition, and handwritten character recognition, e.g., postal code interpretation. Given the architecture, see 1.11, the top layers are used to identify features such as the edges of an image and the subsequent layers are used for feature recombination to form high-level attributes of the input followed by classification. Next, pooling operations will be performed, which mitigates the dimensionality of the extracted features [48].

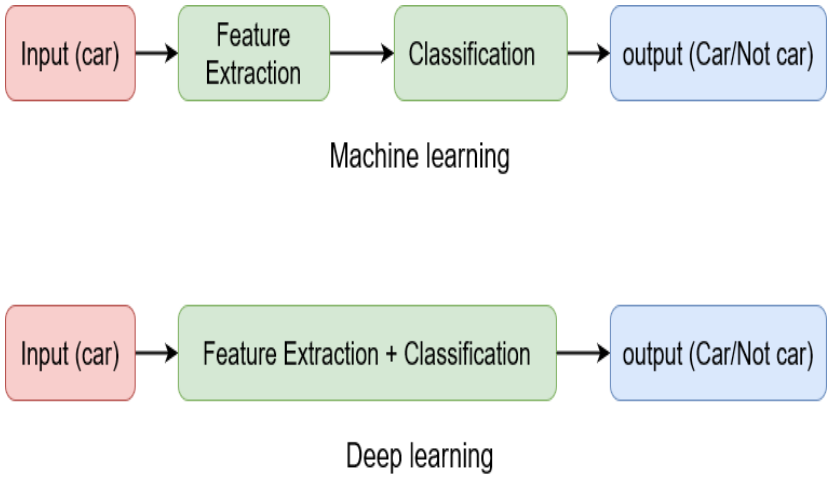


Figure 1.10: Difference between machine learning and deep learning.

The next step is to perform convolution and then again pooling, which is fed into a perfectly linked multi-layer perceptron (MLP) [48], which is a supervised learning algorithm that learns a function $f(.) : R^m \rightarrow R^o$ by training on a dataset where m is the number of dimensions for input and o is the number of dimensions for output [11]. The responsibility of the final layer called the output layer is to recognise the features in the image using back-propagation algorithms, which is a set of steps used to update network weights in order to reduce the network error [5]. In CNN, the advantage of deep processing layers, convolutional layer, pooling and a fully connected classification layer reveals various applications such as speech recognition, medical applications, video recognition and various natural language processing tasks. In figure 1.11, a conventional Neural Network composed of conventional and Pooling blocs is illustrated [48].

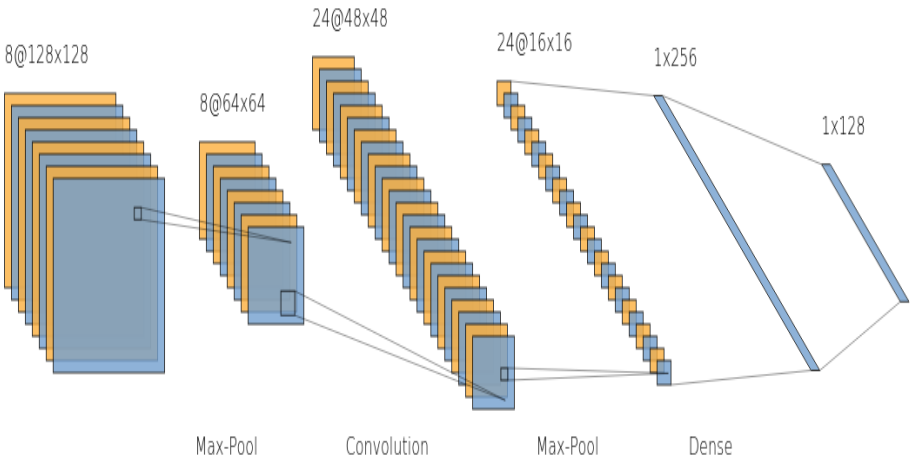


Figure 1.11: Architecture of Conventional Neural Network composed of conventional and Pooling blocs.

Vision-based navigation systems of autonomous vehicles primarily focus on DNN-based systems, in which the controller obtains input from sensors/detectors, such as cameras, and produces a vehicle control output, such as a steering wheel angle to navigate the vehicle safely

in a roadway traffic environment. Typically, these DNN-based systems in the autonomous vehicle are trained through supervised learning [149, 116].

Etxeberria-Garcia *et al.* in [54] have shown that using a monocular vision-based method, incorporated with a deep learning inference can give promising results to localise a railway using techniques like visual Odometry, SLAM or pose estimation [54]. However, it requires generating a new dataset to perform new learning for a given application, which basically slows down the navigation process and doesn't make it a real-time solution for AVs applications.

Mhafuzulet *et al.* in [71] show that a trained DNN-based system can be compromised by perturbation or some negative inputs. Similarly, this perturbation can be introduced into the DNN-based systems of autonomous vehicles by unexpected roadway hazards, such as debris or roadblocks. Thus, proposing a hazardous roadway environment that can compromise the DNN-based navigational system of an autonomous vehicle, and produce an incorrect steering wheel angle, which could cause crashes resulting in fatality or injury. This implies developing a DNN-based autonomous vehicle driving system using object detection and semantic segmentation to mitigate the adverse effect of this type of hazard, which helps the autonomous vehicle to navigate safely around such hazards [71].

Among different available sensors to perform the localisation of robots or self-driving cars, is the use of vision, which has attracted recently much attention from the state-of-art, called visual place recognition (VPR) [150]. That extracted information is fed through a monocular camera to perform self-localisation in such environments (rural, urban). Thus, the current location is found by searching, among the places already visited, the one with the appearance closest to the current image.

However, there exist other navigation models proposed in the community [40, 151]. Among the accurate models that are proposed is CNN which provides accuracy for the navigation decision. Nevertheless, it is a resource-consuming, and it has to be fed by large data sets, along with off-chip memory access which is the primary bottleneck in accelerating those large-scale models, especially with large input size [154].

A visual place recognition (VPR) system always follows a similar pathway: it requires an image acquisition, followed by some image processing that allows building a representation that characterises the current location [25, 152]. Thus, passing those images through an image processing pipeline will carry out a form of information selection followed by its encoding. Due to its multidisciplinary nature, the field of VPR has been studied by several communities and used in a wide variety of applications: in machine vision, [137, 127, 151], in databases [110, 25], and in robotics, [36, 129, 59].

Therefore, CNNs and DNNs require considerable computational capability [21], their use in embedded devices places constraints in terms of power consumption as well as computational capability. However, with the recent and advanced development of new AI-decision-based algorithms, it has become viable to implement the localisation based on the visual-localisation-recognition in an energy-efficient embedded computing environment.

1.4.4 Localisation with autonomous car

To achieve the task of localisation, a self-driving vehicle must successfully handle a large number of problems simultaneously [150]. For example, the system must locate its position, identify practicable pathways, determine potential routes or prevent sources of accidents etc.

To deal with such a variety of issues, car architectures are generally composed of a wide variety of modules, specialised in solving a reduced number of problems [18]. The information extracted by these modules is successively merged to go from the raw sensor to the action on the vehicle. The pipeline is often very classical and follows a logical order: first, the information from the sensors is processed (perception system); then this information is used to localise the vehicle in its environment (localisation system); thereafter a trajectory is computed from the location of the vehicle (path planning system); finally, the trajectory is read and carried out via motor control mechanisms (motor control system) [142].

In such architecture, the responses obtained by the localisation modules have a great impact on the performance of the system. Indeed, the planning block relies heavily on the location provided by the system and requires a high degree of reliability [125]. To reach the highest possible level of performance, location blocks are generally based on the use of very powerful and accurate sensors such as LiDAR (see section 1.3), or RTK GPS [119]. The vehicles that have achieved the greatest localisation trajectories are mostly based on these technologies [33]. For example, the autonomous car proposed by the VisLab team in the VIAC project (VisLab Intercontinental Autonomous Challenge[1]) used information from GNSS and LiDAR to locate the vehicle [31].

However, these sensors remain costly, energy consumptive and heavily impacted by the environment [49, 81]. For example, GNSS (refers to a constellation of satellites providing signals from space that transmit positioning and timing data to GNSS receivers. The receivers then use this data to determine location [16]) is heavily impacted by the nature of the environment, and may not function properly around large buildings or in overcast areas. In the context of *electric* (*i.e.*, battery-powered) autonomous vehicles, this yields a significant impact [49].

This insight led to the development of new alternatives, notably the use of visual information since cameras are cheap and passive sensors that provide access to a rich space of information.

1.4.5 Visual Place Recognition

Visual Place Recognition (VPR) is a field of research that addresses the issue of locating a place from visual information. The general idea is to determine the position at which an image was taken by comparing it with a geo-referenced database of images. The proposed methods have been used in many fields such as robotics [36, 40], big data [110] or machine vision [127, 151]. Each case has very specific constraints, notably in terms of computational time, accuracy and computational cost, which do not necessarily lend themselves to every use case.

From an architectural point of view, VPR models often follow this workflow: first, an image is analysed to find its characteristic information; second, the detected information is transformed into a compact and meaningful location code; finally, the code is sent to a memory which has to store the location code (for the learning phase) or send information back (for the using phase), if the image belongs to an already known location [40, 44]. Thus, it becomes possible to create a complete representation of an environment by memorising images at regular intervals.

In general, the performance of a VPR system is evaluated according to three criteria: first, the accuracy of the model, *i.e.*, the average distance between the coordinates of an image

[1]The objective of the VIAC project was to test the capability of an autonomous vehicle at very high intensity by taking it on a track of 16,000km from Italy to China

to localise and the coordinates of the image that the model best recognises; second, the computation frequency of the model as a function of the number of images learned; and third the use of computing resources.

Currently, the best performing state-of-the-art models are deep models such as NetVlad, HybridNet or RegionVlad [151]. However, these models are very expensive in terms of computational resources[2] with higher complexity than traditional, non-Machine Learning approaches. For example, the CoHog model [151] gives comparable performances to deep models while being much less greedy in computational resources. Nevertheless, this model is a VPR model of the "Global Handcrafted Feature" family and does not need to be trained before being used.

1.5 LPMP, a bio-inspired model of localisation

Among the various existing models, the Log-Polar Max-Pi model [53, 43] (LPMP) is depicted in figure 1.16 represents our main research context for a hardware implementation of a visual localisation model (for further details and results, see section 2). Inspired by the functioning of animal cognition, the model allows for the building of a neuronal representation of an environment from visual information. In particular, it mimics a family of spatial neurons called place cells that can be observed in the hippocampal formation of mammals [64].

Initially designed to reproduce observations made in neurobiology in small mostly indoor environments, however, the model is not adapted to operate in autonomous vehicles. Preliminary works have thus been conducted and have demonstrated the need to overcome several problems before being able to use the model in the field of AVs [53, 53].

In particular, The work of Colomer *et al.* showed that a scaling up of Espada *et al.* bio-inspired neural module for visual localisation, called the LPMP model, is necessary to consider its use on a self-driving vehicle.

The main reasons why we have used the LPMP approach in our thesis project:

- As the LPMP model is the unique localisation model in the state of art of the bio-inspired models, with a few neurons in its model
- The ability of the model to operate in large environments, also the encoding used method to process visual information in order to reduce the computational cost of the system.
- The robustness to various environmental conditions, as it was demonstrated by Colomer *et al.*, the model needs to be robust to many different conditions (weather, traffic, etc.) to be able to operate for long periods of time.
- The Adaptation to AVs operation, as the model, offers a well-established navigation process (from localisation to control) for the AVs. The transition on vehicle implies the shift from a visuospatial representation to a sensorimotor one and the integration of the contributions made on the visual localisation (See Colomer *et al.* work for more details [43]).

[2] These models use a very large number of neurons to encode an image and often require the use of a graphics card.

1.5.1 Principles of the LPMP model

The model is used in two stages: The first is the learning stage, where a representation of the environment is learned in one-shot learning. During this stage, the model learns a number of images that are representative of a particular position. Depending on its version, the system learns the images, unsupervised and online, at regular intervals of distance or via a novelty detector (LPMP+vig). The second stage is the *query stage*. During this stage, the system analyses a batch of N images and returns their localisation within the learned representation. In the case of an autonomous vehicle application, the image analysed by the system is the one acquired by the camera.

Points of Interest detection To localise an image, the model follows the classic VPR system pipeline (see section 1.4.5): LPMP analyses an Image I to detect its points of interest (PoI).

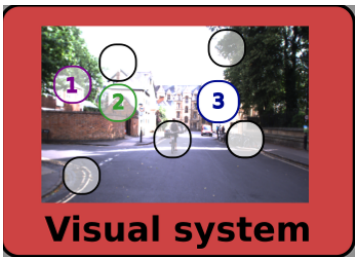


Figure 1.12: Points of Interest detection referred to as landmarks.

Saliency points filtering During this step, I is convolved with a Deriche filter, then with a DoG (Difference Of Gaussians) filter from which the most salient points are selected through a competition mechanism to retain only the most pertinent ones.

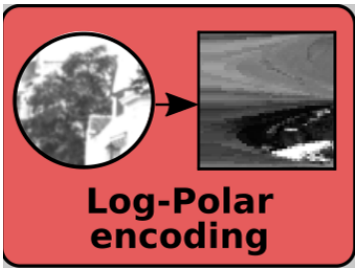


Figure 1.13: Saliency points filtering.

Points of Interest encoding Then, LPMP encodes points of interest to get a compact and meaningful code of a place. At this point, the LPMP model builds a visuospatial pattern by performing a what-where merging in a Max-Pi layer, by using the visual identity of each PoI (*via* Log-Polar encoding) with their absolute orientation angle (obtained through azimuth computation and encoding). To encode the visual identity of a landmark, the LPMP model proceeds to a log-polar transform on the thumbnail around the PoI.

Memory querying Finally, LPMP queries its memory to return the location that best matches I. To do so, the visuospatial pattern is passed to a neural memory: WTA (Winner-Take-All). This memory contains the patterns of all previously learned locations (one location

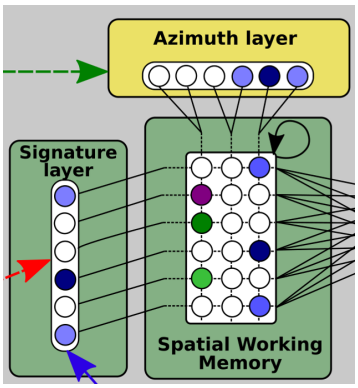


Figure 1.14: Points of Interest detection along with Azimuth layer encoding grouped in Working-space-memory.

per neuron). At the time of a request, it returns by neuron a level of activity correlated to the proximity level of the current pattern and of all those learned.

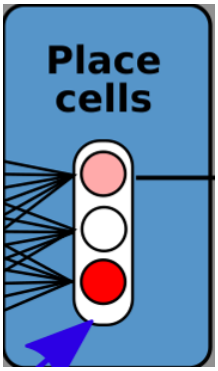


Figure 1.15: Memory querying as Place cells.

1.5.2 Some Advantages and Limitations of LPMP

LPMP offers a promise, particularly in terms of simplicity and consistency when the scene exhibits minimal variation under a specific lighting level. However, it is sensitive to abrupt changes in lighting conditions, such as transitioning from a bright sunny environment to a dark tunnel, and vice versa. Nevertheless, if the ambient light levels remain relatively stable, LPMP demonstrates high accuracy in recognising locations even in the presence of human activity, including moving objects like pedestrians, cyclists, and other vehicles. This level of accuracy is deemed sufficient, considering that LPMP has undergone validation using diverse datasets, some of which are considerably large, as demonstrated by Espada et al. [53]. LPMP has proven its competitiveness when compared to other state-of-the-art models and has achieved accurate results in localising objects in the KITTI dataset [60], even with a small sample size for image learning [43].

1.6 Conclusion

The localisation task for autonomous vehicle navigation is mandatory to provide the decision-making feature to it. A lot of projects and surveys dissect different aspects of their implementations, deployment, and energy efficiency. We have seen previously that LiDAR as a system for detecting landmarks through frames of image processing, is greedy of energy and costly. Thus, that is why we have decided to use a visual place recognition based on a bio-inspired neural architecture called LPMP, for localisation specifically. The LPMP model uses a monocular type of camera for the landmark or points of interest information.

In the next chapter, we will explore the different platforms used to accelerate such intensive calculations, and the specifications and requirements we need to set to make our application run-time, as well as, efficient. Thus, we will see the GPU and FPGA as types of boards for the application acceleration. In addition, we will present how the state of art lave-rage from those embedded efficient platforms deploy and respond to their demanded specifications, also set a comparison between those two different architectures, and then conclude which of those is the more appropriate and conveyable to our demands.

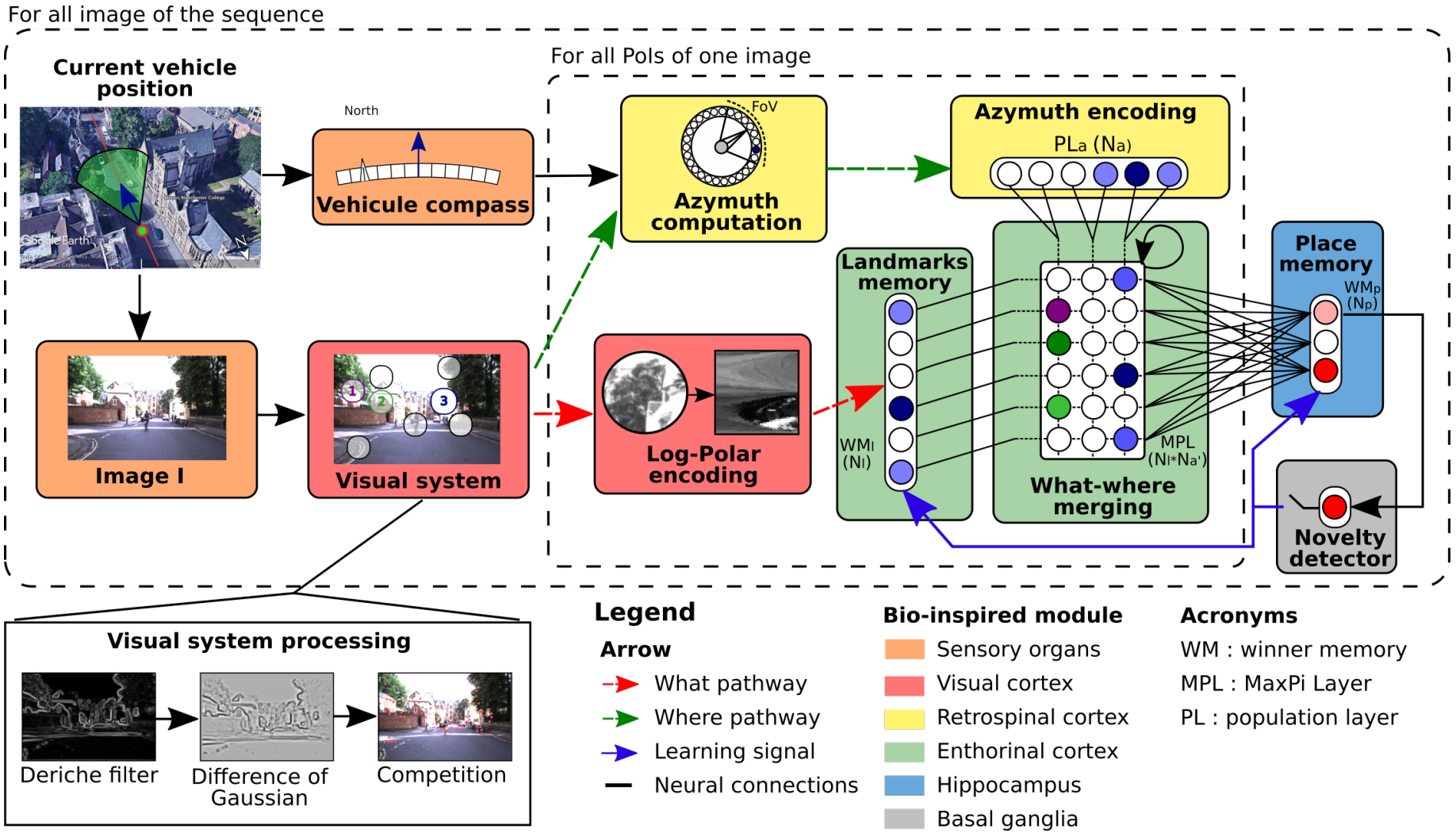


Figure 1.16: **Overview of LPMP model.** This figure illustrates how the LPMP model builds a neural representation of an environment from visual information. To do so, the system must go through several stages: Points of Interest detection ("Visual System," on the left); Saliency points filtering (Deriche filter and DoG); Points of Interest encoding (Log-Polar encoding); and finally, memory querying (access through Signature-Layer, Spatial-Working-Memory, and Place-Cells). At this stage, the neural activity of Place memory provides the best-recognised location based on what it has previously learned [51].

2

Hardware Acceleration for autonomous vehicles localisation

The use of a Convolutional Neural Network based method for object detection increases an accuracy that surpasses human visual system [133]. Because it requires considerable computational capability as seen in section 1.4.3, its use in embedded devices, which place constraints in terms of power consumption as well as computational capability has thus far been limited. However, with the recent development of graphics processing unit GPUs for use in embedded devices and open-source software libraries for machine learning, it has become viable to utilise CNNs and various machine learning acceleration means in an energy-efficient embedded computing environment [106].

These complexities (computational support for application acceleration and power consumption) impose numerous challenges for the design of autonomous driving edge computing systems (Edge computing is a distributed computing framework that brings enterprise applications closer to data sources such as the Internet of things IoT devices or local edge servers. This proximity to data at its source can deliver strong business benefits, including faster insights, improved response times and better bandwidth availability [6].). First, edge computing systems for autonomous driving need to process an enormous amount of data in real-time, and often the incoming data from different sensors are highly heterogeneous. Since autonomous driving edge computing systems are mobile, they often have very strict energy consumption restrictions. Thus, it is imperative to deliver sufficient computing power with reasonable energy consumption, to guarantee the safety of autonomous vehicles, even at high speed. Second, in addition to the edge system design, vehicle-to-everything (V2X) provides redundancy for autonomous driving workloads and alleviates stringent performance and energy constraints on the edge side [86].

Therefore, we present within this chapter the use of Heterogeneous systems-on-chip (HeSoC) based on reconfigurable accelerators, such as Field-Programmable Gate Arrays (FPGA) for implementing the localisation application based on the Visual Processing Recognition (VPR) algorithm. FPGA represents an appealing option to deliver the performance/Watt required by the advanced perception and localisation tasks employed in the design of autonomous Vehicles [86]. Contrary to software-programmed GPUs, FPGA development involves significant hardware design effort, which in the context of HeSoCs is further complicated by the system-level integration of HW and SW blocks. High-Level Synthesis is increasingly being adopted to ease hardware IP design, allowing us to quickly prototype their solutions [86].

2.1 Computational supports for Hardware implementation

2.1.1 GPU

the graphics processing unit (GPU) has emerged as a versatile platform for running massively parallel computation [113]. Graphics hardware presents clear advantages for processing huge type datasets encountered in different domains of applications, as they provide computation support with a high memory bandwidth, high computation throughput, support for floating-point arithmetic, the lowest price per unit of computation, and a programming interface accessible to the non-expert [113]. These features have raised tremendous enthusiasm in many disciplines, such as linear algebra, differential equations, databases, data mining, computational biophysics, molecular dynamics, fluid dynamics, seismic imaging, game physics, and dynamic programming [39].

2.1.1.1 Using GPU for navigation process

Perceiving the surrounding environment is crucial for autonomous mobile robots. Autonomous vehicles can use the information of the key point for navigation in an unknown environment or perceptive locomotion control over rough terrain [98]. Depending on the application, various post-processing steps may be incorporated, such as smoothing, painting or plane segmentation. Miki *et al.* [98] in their work, suggest an elevation mapping pipeline leveraging GPU for fast and efficient processing with additional features both for navigation and locomotion, with a demonstration of their mapping framework through extensive hardware experiments. The result is shown in figure 2.1. Since the processing calculation is done on GPU, the processing time remained short even for large numbers of points. While the baseline method's calculation (which uses the CPU) time grew at a steeper rate with respect to the number of points. With the use of Bpearl sensor [8] data to measure the calculation time for each component on Jetson Xavier and collected 1000 measurements, the calculation time per feature (with a number of points of 43017), is 6.857ms, without taking into consideration the data-movement between CPU DRAM and device shared memory.

However, it's true that Miki *et al.* used Xavier Jetson AGX for their acceleration. Along with a LiDAR camera, which is greedy for energy as a type of source. Thus, the overall energy consumption will be considerable, although the latency and time execution is as considerable as FPGA will deliver see chapter 5.

In another work conducted by Beyeler *et al.* in [32] which presents a cortical neural network model for visually guided navigation that has been embodied on a physical robot exploring a real-world environment. The model includes a rate-based motion energy model for area V1 of the brain, and a spiking neural network model for cortical area MT. The model generates a cortical representation of optic flow, determines the position of objects based on motion discontinuities, and combines these signals with the representation of a goal location to produce motor commands that successfully steer the robot around obstacles toward the goal. The model produces robot trajectories that closely match human behavioural data. This study demonstrates how neural signals in a model of the cortical area might provide sufficient motion information to steer a physical robot on human-like paths around obstacles in a real-world environment, and exemplifies the importance of embodiment, as behaviour is deeply coupled not only with the underlying model of brain function but also with the anatomical constraints of the physical body it controls. This work uses NVIDIA Jetson TX2 as a type of board to accelerate the calculation of the cortical neural network model for

visually guided navigation, the latency and time of getting implemented on the platform are nearly 100-150× slower than our N-LOC hardware implementation of the visual place recognition LPMP model, further details see chapter 5.

Along side, Tesla has introduced the Tesla D1, a new chip specifically designed for artificial intelligence. Powered by NVIDIA, the Tesla D1 is capable of delivering a power of 362 TFLOPs and shows great promise for turbocharging autonomous driving [15].

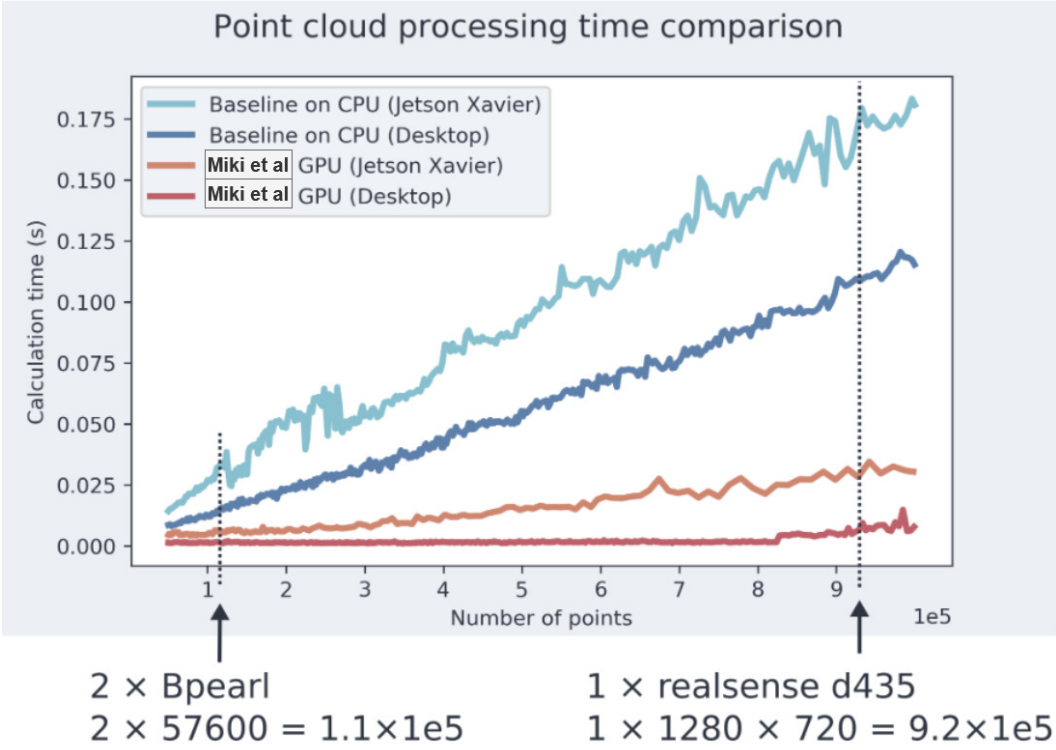


Figure 2.1: Point cloud processing time comparison between Miki *et al.* method and an existing implementation on CPU [56]. The calculation time increases more with the baseline method while it stays comparatively low with miki et aL.'s approach. The number of points is indicated with 2xBpearl [8] and 1xrealsense [7] on the plot. The proposed mapping pipeline could process the data in real-time, while the baseline method had a considerable delay on the onboard PC (Jetson). recreated from Miki *et al.* work in [98].

2.1.1.2 An example of embedded GPU: NVIDIA's Jetson TX2

The NVIDIA Jetson TX2 is an a high-end embedded heterogeneous system, which features an ARM Cortex-57 quad-core processor, an NVIDIA Pascal GPU with 2×128 cores (2 shared-memory processors, or SMPs), 8 GiB of DRAM, and 32 GiB of eMMC storage. It aims at real-time image and video processing, possibly within the context of an AI application. Edge computing is clearly one of the targets of the Jetson TX2. For more details see Figure 2.2.

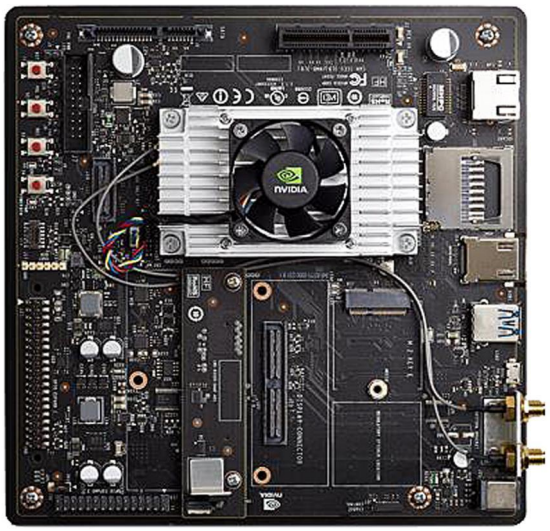


Figure 2.2: An overview of the NVIDIA Jetson TX2, which was utilised in our experiments. See section 4.6.1.

2.1.2 FPGA

2.1.2.1 Principles

In recent years, FPGA is becoming a promising solution for algorithm acceleration [68]. Compared with CPU, GPU, and DSP platforms, for which the software and hardware are designed independently, FPGA allows the developers to implement only the necessary logic in hardware according to the needed algorithm [68]. By eliminating the redundancy in general hardware platforms, FPGAs can achieve higher efficiency. Application-specific integrated circuits (ASICs) based solutions achieve even higher efficiency but require a much longer development cycle and higher cost [68].

2.1.2.2 Using FPGA for navigation process and motivation beyond

Future autonomous vehicles are directly related and depend on vision-based navigation, which imposes great computational challenges, as we have seen and explored in the chapter 1. Lentarise *et al.* in [83] develop a high-performance supporting custom computer vision algorithms of increased complexity for AVs pose tracking. At the algorithmic level, they follow a 6D pose by rendering a depth image from an object mesh model and robustly matching edges detected in the depth and intensity images. At the system level, they devise an architecture to exploit the structure of commercial system-on-chip FPGAs, *i.e.*, Zynq7000, and the benefits of tightly coupling VHDL accelerators with CPU-based functions, the figure 2.3 gives details of the implemented architecture for the tackled application. At the implementation level, they employ their custom HW/SW co-design methodology and an elaborate combination of digital circuit design techniques to optimise and map efficiently all functions to a compact embedded device. Providing significant performance per watt improvement, the resulting system achieves a throughput of 10-14 FPS for 1 Mpixel images, with only 4.3 watts mean power and 1U size while tracking in real-time with only 0.5% mean positional error. The

final system: resources on SoC FPGA *XC7Z100IFFG900-2L*, it consumes overall 36% of the Look-up-table (LUT), 77% of the BRAM, in addition, it requires ≈ 83ms.

Based on what we have presented in GPU and FPGA sections before, for their use in the navigation of mobile robots or even for AVs. Using the FPGA as a type of board to accelerate the calculation, especially for the localisation task, is justified by the comparable latency that can provide hundreds of milliseconds compared to the GPU results, also the computational challenges via a combination of techniques at the architecture and VHDL level. Thus, To decrease time and make it a run-time system, we shall employ a deep-pipelining, module replication, parallel memory organisation, tight coupling of PS and PL functions, as well as task-level parallelisation and balanced scheduling.

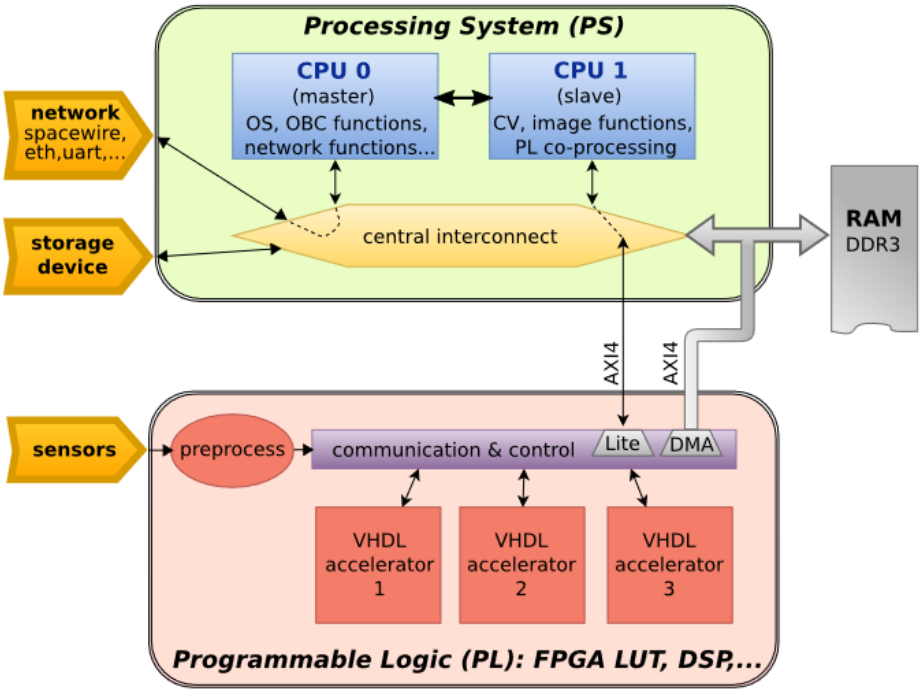


Figure 2.3: Proposed system on top of Zynq7000 SoC FPGA for the application support. Recreated from Miki *et al.* work in [83].

2.1.3 Comparing State-of-the-Art Hardware Platforms: Performance, Features, and Costs

In the realm of hardware platforms, two contenders stand tall: FPGA (Field-Programmable Gate Array) platforms, including Zynq 7000 and Ultrascale, and GPU platforms like NVIDIA Jetson TX2 and NVIDIA Jetson Xavier. When comparing these state-of-the-art options, it's important to consider the specific application, workload, and implementation at hand. To assist in the selection process, Table 2.1 presents a comprehensive overview of the metrics used to evaluate the suitability of an FPGA as an acceleration platform. These metrics include flexibility, power consumption, performance, reconfigurability, and scalability. By utilizing these metrics, we can assess the FPGA's potential for meeting our specific requirements and implementation needs.

After careful evaluation and consideration of the aforementioned metrics, the FPGA emerges as the chosen platform for acceleration. With its remarkable flexibility, low power consumption, impressive performance, reconfigurability, and scalability, the FPGA proves itself to be the true champion in this epic battle of hardware platforms. The metrics are presented and explained as follow:

- Flexibility refers to the ability of a hardware architecture to adapt and support a wide range of applications and functionalities. A highly flexible HW architecture allows for efficient implementation of various algorithms and designs by providing a diverse set of resources.
- Power consumption is a metric that quantifies the amount of electrical power consumed by an architecture during its operation.
- Performance refers to the speed and efficiency of an architecture in executing a specific task or workload. It is typically measured in terms of throughput, latency, or operations per second.
- Reconfigurability measures the ability of an architecture to be dynamically reprogrammed or reconfigured during runtime. This feature is exclusive to FPGA platforms that can effectively utilise such functionality.
- Scalability refers to the ability of an architecture to efficiently accommodate designs of varying sizes and complexities. A scalable HW architecture can support small, medium, and large designs without significant degradation in performance or resource utilisation.

Throughput
FPGAs are highly parallelisable and can achieve high throughput by implementing custom hardware designs. The actual throughput of an FPGA depends on the design and optimisation techniques employed.

GPUs are optimised for parallel processing and can also achieve high throughput for certain workloads, such as image and video processing: Tasks such as image and video encoding/decoding, object detection, image recognition, and video transcoding can be accelerated using GPUs, machine learning and deep learning: Training and inference tasks in machine learning and deep learning algorithms can be accelerated using GPUs, Computational Fluid Dynamics (CFD): CFD simulations involve solving complex equations to study fluid dynamics, financial modeling: Financial institutions often deal with vast amounts of data for risk analysis, portfolio optimisation, and pricing models, etc. The throughput of a GPU depends on factors such as the number of cores and the efficiency of the implemented algorithms.

Performance
FPGAs offer high performance due to their ability to implement custom hardware designs tailored to specific applications. They can achieve low latency and high-speed processing for tasks that benefit from hardware acceleration, such as Real-time Signal Processing: FPGAs can be utilised for real-time signal processing applications such as digital filtering, audio and video processing, radar and sonar signal processing, and wireless communication, High-Frequency Trading: In the financial industry, FPGA-based solutions are often employed for high-frequency trading (HFT), Network Packet Processing: FPGAs can be used for network packet processing tasks, such as packet parsing, deep packet inspection, and traffic classification, Cryptography and Security: FPGAs can accelerate cryptographic algorithms, including encryption, decryption, hashing, and authentication, High-Performance Computing (HPC):

FPGAs can be integrated into HPC systems to accelerate computationally intensive tasks, Machine Learning Inference: FPGAs can be utilised for high-performance machine learning inference tasks.

GPUs excel in tasks that can be parallelised, such as graphics rendering and machine learning. With their large number of cores, GPUs can process substantial amounts of data concurrently.

Power consumption
FPGAs generally consume less power compared to GPUs. This is because FPGA designs are customised to efficiently perform specific tasks, resulting in reduced power requirements.

GPUs have higher power consumption due to their large number of cores and complex architectures designed for general-purpose computing.

Cost
FPGAs can be more expensive than GPUs, particularly when factoring in development costs. FPGA development typically necessitates specialised knowledge and tools, which can contribute to higher overall expenses.

GPUs, especially those designed for embedded systems like NVIDIA Jetson TX2 and Jetson Xavier, are relatively more affordable and widely accessible.

Flexibility
FPGAs offer a high degree of flexibility as their hardware designs can be reprogrammed to adapt to changing requirements. This makes FPGAs suitable for applications that require frequent updates or customisation.

GPUs provide flexibility at the software level, enabling developers to leverage parallel processing capabilities for a wide range of tasks. However, GPUs are less flexible at the hardware level compared to FPGAs.

Based on the presented reference in table 2.1, FPGA is the architecture that offers the most favorable comparison across the different metrics listed in Table 2.1. To further quantify the comparison between FPGA and GPU, Table 2.2 presents power consumption estimates for various workloads. For instance, for a highly complex application involving both image processing IP and machine learning IP, the power consumption can be approximately 7W. In contrast, the NVIDIA Jetson TX2 can reach up to approximately 15W.

HW Plts	Flexibility	Power consumption	Performance	Reconfigurability	Scalability
CPU	[67]	[114, 67]	[15, 67]	N/A	[67]
GPU	[15]	[114, 67]	[15, 34, 67]	N/A	[15, 34]
FPGA	**[34, 75]**	**[140, 34, 114, 75]**	**[140, 34, 114, 67, 75]**	**[34]**	**[140, 34, 114, 75]**
ASIC	[75]	[75]	[75]	N/A	[75]

Table 2.1: Comparing state-of-the-art hardware platforms: Performance, features, and costs.

The throughput of a GPU relies on factors such as the number of cores and the efficiency of the implemented algorithms. They excel in tasks that can be parallelised, such as graphics

rendering and machine learning, leveraging their numerous cores to process large volumes of data concurrently. GPU power consumption varies, ranging from 800W for the latest server or workstation-oriented GPUs to as low as 15W for the Jetson TX2 SoC. This power consumption is attributed to their large core counts and complex architectures designed for general-purpose computing, even in embedded settings. GPUs, particularly those designed for embedded systems like NVIDIA Jetson TX2 and Jetson Xavier, offer relative affordability and wide accessibility. At the software level, GPUs provide flexibility, allowing developers to harness parallel processing capabilities for diverse tasks, as mentioned earlier in the context of throughput workloads. However, when considering the inherent design architecture of GPUs and FPGAs, and drawing upon the definition of flexibility as a metric, GPUs exhibit less hardware-level flexibility compared to FPGAs.

HW plts	Power consumption estimation (state-of-the-art)	Performance estimation (state-of-the-art)	Cost
FPGA Zynq 7000	**Up to** $\approx 7W$	**Up to** ≈ 10**TFOPS**	**Reasonable**
NVIDIA Jetson TX2/TX2i	7.5W / 15W	1.33 TFLOPs	Affordable
NVIDIA Javier Xavier	15W / 30W	21TOPS	Reasonable

Table 2.2: Hardware acceleration platforms: Quantitative features.

2.1.3.1 Programming FPGA using VHDL

VHDL is one of the commonly used Hardware Description Languages (HDL) in digital circuit design [17]. VHDL stands for VHSIC Hardware Description Language. In turn, VHSIC stands for Very-High-Speed Integrated Circuit [17].

VHDL was initiated by the US Department of Defense around 1981. The cooperation of companies such as IBM and Texas Instruments led to the release of VHDL's first version in 1985. Xilinx, which invented the first FPGA in 1984, soon supported VHDL in its products. Since then, VHDL has evolved into a mature language in digital circuit design, simulation, and synthesis [97].

VHDL is a programming language that has been designed and optimised for describing the behaviour of digital circuits and systems [17]. As such, VHDL combines features of the following, a simulation modelling language, a design entry language, a test language, and a netlist language [9]. VHDL is optimised for electronic circuit design, and as such their many points in the overall design process at which it can help [9]. It can help with design specification, design capture, design simulation and documentation, and so on and so forth [9].

2.1.3.2 Programming FPGA using High-level Synthesis

Specialised hardware has the potential to provide huge acceleration at a fraction of a processor's energy, the main drawback is related to its design. On one hand, describing these components in a hardware description language (HDL) (e.g. VHDL or Verilog) allows the designer to adopt existing tools for RTL and logic synthesis into the target technology. On the other hand, this requires the designer to specify functionality at a low level of abstraction, where cycle-by-cycle behaviour is completely specified. The use of such languages requires advanced hardware expertise, besides being cumbersome to develop. This leads to longer development times that can critically impact the time-to-market [103].

An interesting solution to realise such heterogeneity and, at the same time, address the time-to-market problem is the combination of reconfigurable hardware architectures, such as field-programmable gate arrays (FPGAs) and high-level synthesis (HLS) tools [27]. FPGAs are integrated circuits that can be configured by the end user to implement digital circuits. Most FPGAs are also reconfigurable, allowing a relatively quick refinement and optimisation of a hardware design with no additional manufacturing costs. The designer can modify the HDL description of the components and then use an FPGA vendor toolchain for the synthesis of the bitstream to configure the FPGA. HLS tools start from a high-level software programmable language (HLL) (e.g. C, C++, SystemC) to automatically produce a circuit specification in HDL that performs the same function [103]. HLS offers benefits to software engineers, enabling them to reap some of the speed and energy benefits of hardware, without actually having to build up hardware expertise. HLS also offers benefits to hardware engineers, by allowing them to design systems faster at a high-level abstraction and rapidly explore the design space.

2.1.4 Heterogeneous System Architecture

Heterogeneous computing is a key strategy to meet the requirements of many compute-intensive applications [34]. However, current platforms which leverage both CPUs and FPGAs are commonly underutilised, as scheduling is often constrained to a run-to-completion model or to the acceleration of a single application at a time [41]. With specifically designed hardware, reconfigurable fabrics represent the next possible solution to surpass GPUs in speed and energy efficiency [34]. Various FPGA-based accelerator designs have been proposed with software and hardware optimisation techniques to achieve high speed and energy efficiency [41].

In contrast, GPUs offer up to 10 TOP/s (*i.e.*, *tera-operations* $\approx 10{\cdot}10^{12}$ operations per second) peak performance and are good choices for high-performance neural network applications. Development frameworks like Caffe [76] and Tensorflow [20] also offer easy-to-use interfaces which makes GPU the first choice for neural network acceleration. Alongside CPUs and GPUs, FPGAs are becoming a platform candidate to achieve energy-efficient neural network processing. FPGAs can implement high parallelism and make use of the properties of neural network computation to remove additional logic. Algorithm researches also show that a Neural Network (NN) model can be simplified in a hardware-friendly fashion without hurting the model accuracy [68]. Therefore, FPGAs offer the potential to achieve higher energy efficiency compared to CPUs and GPUs [104]. However, FPGAs are also infamous for being hard to design and program energy-efficient and highly performant neural networks, especially from a software developer's perspective, compared to a CPU and/or GPU-centric approach [104]. Nevertheless, FPGAs are the best-suited devices if one requires end-to-end control, as they allow the systems engineer to design a tailored platform, provided the necessary field expertise is there to exploit them and yield a low power and energy footprint [104].

Several FPGA vendors propose AI-based or neural architecture-oriented solutions using FPGAs, including Xilinx products such as Alveo U55C [19], which is geared toward HPC and Big Data workloads; Versal ACAPs, a cloud-oriented platform, *etc.* However, such solutions rapidly become of limited use when considering unconventional solutions, *e.g.*, to implement bio-inspired algorithms in hardware, as resource utilisation can quickly lead to resource exhaustion when evaluating new algorithms. This leads the systems designer to explore other venues, such as multi-FPGA designs, and solve additional issues, *e.g.*, how to correctly synchronise FPGA systems connected through gigabit serial links, and find the best communication schedule for a given (set of) design(s).

Hence, the reasons to resort to using a multi-FPGA board with an embedded high-speed interconnect are the following:

- Necessity for prototyping complex algorithms that need to be scaled.
- Leveraging Dynamic partial reconfiguration for the aim of reducing energy, power consumption, and space locality Task placement.
- Facilitate the incorporation of such middle-ware for partitioning: if there is a need to schedule work on multiple devices, how much workload should be executed on each device? For instance, scheduling 25% of the threads on CPU and 75% of the threads on FPGA.
- Leveraging high-speed transceiver protocols as an intrinsic property of FPGA to communicate over multiple ones.

2.1.5 Exploring Hardware and Software Scheduling Strategies on Heterogeneous HW

The second issue requires the exploration of new scheduling strategies to correctly allocate parts of a NN to the target real-time offline scheduling for heterogeneous architectures, many of which get their inspiration from Topcuoglu *et al.*'s HEFT [136], an offline scheduler for periodic tasks. Yet, while autonomous vehicles may benefit from a good default allocation policy using this method, the rapid changes in the environment also require good online scheduling strategies. Some efforts, such as *e.g.*, Chen *et al.*'s [41], propose a scheduling algorithm for non-reconfigurable heterogeneous embedded platforms. Rossi *et al.* [120] propose an algorithm to make the reconfiguration process preemptive to allow higher-priority tasks to be reconfigured first.

Finally, some strategies co-schedule periodic and aperiodic tasks on heterogeneous systems [122], which requires to use of a hybrid offline/online algorithm. We wish to go in that direction for multi-FPGA systems, with "general-purpose" Partial Reconfigurable regions (PRRs) to reduce the parameter space during the decision-making phase of the scheduler.

2.1.6 Leveraging partial reconfiguration

Since a pure software implementation is not sufficient to meet an autonomous vehicle's real-time requirements, an accelerator is needed. Hardware implementations can vastly outperform software ones. However, Fiack's work also shows resources are quickly saturating when implementing NN on FPGAs [57]: around 100K neurons were implemented on a Stratix V board. Yet, while Wizarde (described in section 2.3) is arguably more powerful, it features $\approx 350K$ cells per tile—roughly half what is available in a Stratix V.

Hence, the use of dynamic partial reconfiguration (DPR) is essential to implement neural networks. Our main application which is the localisation task is large enough that it will not completely fit into the available reconfigurable fabric. Moreover, the LPMP model [43], which studies and tackles the localisation aspect of an AV, shows that it is linked directly to the capability and the performance of an AV on self-localisation, or even in the environment exploration [43], for more details about the deployment and distribution of the bio-inspired neural architecture *see* 5.

Thus, the computational stills unchangeable according to the vehicle's environment, *e.g.*, transitioning from a dense urban area to a rural one, the neurons used to decide will be the same for all kinds of environment transition [43]. As a result, relying on a full hardware solution is not reasonable or realistic. Therefore, to leverage the whole multi-FPGAs platform wisely, and to grant the implemented application a high capability to navigate indoors or/and outdoors, the system should rely on a light software layer which will provide a scheduling and resource management environment, to decide which and where hardware task to allocate, within an FPGA.

Hence, two major steps must be achieved:

- A middleware layer to provide an API to load and replace hardware tasks.
- Set of a scheduler(s) able to achieve real-time navigation using a bio-inspired approach.

Hence, scheduling algorithms' capabilities must be tested to figure out the best task allocation strategy to achieve real-time navigation using a bio-inspired approach. Through the emerging of CPU, FPGA accelerators can be managed much more efficiently with more complex strategies, which inevitably optimises and outperforms the acceleration.

2.1.7 Facilitating DPR

2.1.7.1 Using Ker-ONE as a hypervisor for DPR facility

Ker-ONE is a hypervisor proposed by Xia *et al.* [146]. It provides a Partially Reconfigurable Region (PRR) Monitor, to let the system software know which PRR is available for reconfiguration. Ker-ONE provides an API a programmer can use to load new bitstreams using this approach. Which allows us to use it as a bare-metal kind of setup. The Ker-ONE binaries yield a very small footprint: the kernel is 123KiB large; the user binary is 109KiB large. The associated static design, including Ker-ONE's partial reconfigurable region (PRR) monitor, takes 5504 LUTs, which is less than 2% of the available real estate on a Kintex-7 FPGA.

The Ker-ONE framework is shown in figure 2.4. It consists of a host micro-kernel and a user-level environment. Ker-ONE follows the principle of minimal authority and low complexity. The micro-kernel is the only component that runs at the highest privilege level in the kernel part, in the supervisor mode. Only the basic features that are security-critical have been implemented in the micro-kernel [146], such as the scheduler, memory management, and inter-VM communication. All non-required features have been eliminated so that the microkernel's Trust Computing Base (TCB) is mitigated and reduced. The Trust Computing Based corresponds to pieces of software and hardware on top of which the system security is built. As known in the operating system development, a smaller TCB size corresponds to higher security [130]. Since it reduces the system's attack surface. The user environment runs in user mode and is composed of additional system services, such as device drivers, file systems, VM bootloaders, which run as server processes, see figure 2.4. Thus, this framework is designed to be scalable and easily adaptable to extension mechanisms [146]. Multiple virtual machines (VM) run on top of the user environment and Ker-ONE is based on para-virtualisation. In this technique, a guest OS is modified to explicitly make calls (*i.e.* hyper-calls) to the hypervisor or a virtual machine monitor in order to handle privileged operations. Each virtual machine can host a para-virtualised OS (*i.e.* guest OS) or a software image of a user application, which has its own independent address space and executes on a virtual piece of hardware. Therefore, Ker-ONE relies on a virtual machine monitor(VMM)

to support the execution of guest OSs in their associated virtual machine. It then handles virtual machines' hyper-calls, emulates sensitive instructions and provides virtual resources to the virtual machines.

The design of Ker-ONE has been based on some assumptions, presented as follow:

- It is ported only on one single-core architecture.
- Ker-ONE deals with the virtualisation of simple guest OSs such as uC/OS or FreeRTOS, instead of complex systems such as Linux, since para virtualising these types of OS would be quite expensive and error-prone.
- In order to provide strong protection to critical tasks, all critical real-time tasks are executed in one specific guest real-time OS (RTOS). The less critical tasks execute in general-purpose OSs (GPOSs). Therefore, Ker-ONE is designed to cohost a single guest RTOS and one or multiple additional guest GPOSs.

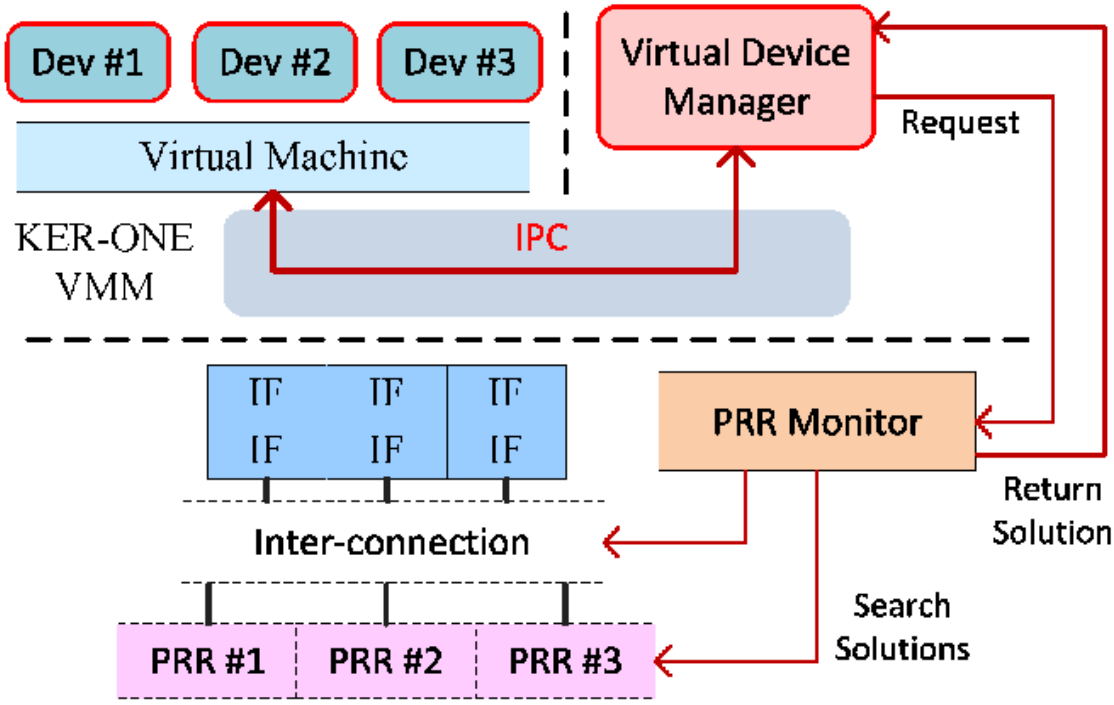


Figure 2.4: Ker-ONE consists of a micro-kernel and of a Virtual Machine Monitor running at a privileged level. The User environment executes in a non-privileged level. Recreated from XIA *et al.* work in [146].

2.1.7.2 Abstracting FPGA manipulation through adequate system-level layers FOS

While useful, DPR itself requires the application designer to provide their own custom scheduler for each application. A better way would be to somehow abstract reconfigurable regions and make them available through some kind of middleware or system software. This is what FOS (FPGA Operating System) proposes to do [141]: after providing a static design for the FPGA (of which some are "glue logic" to interact with FOS, and the rest is what the actual system designer wishes to use for their application), reconfigurable regions can be considered

"simple" accelerator slots, much like GPUs in mainstream computing, for more details about FOS architecture, see figure 2.5. The goal beyond this work is to see how FOS can be used to drive a multi-FPGA system from a scheduling policy viewpoint, as well as from a DPR perspective. In addition, care will need to be taken to add transceiver logic to the FOS infrastructure, so that various instances of FOS (one per tile) can efficiently communicate. The welcoming team at Manchester, led by Dr Dirk Koch, is behind the design and implementation of FOS and is very interested in dealing with scheduling tasks across multiple FPGAs, but also implementing dynamic scheduling policies, as well as managing various task sizes, which will probably require all parties of this project to design task descriptors which will include meta-data to describe the needs and constraints of the tasks.

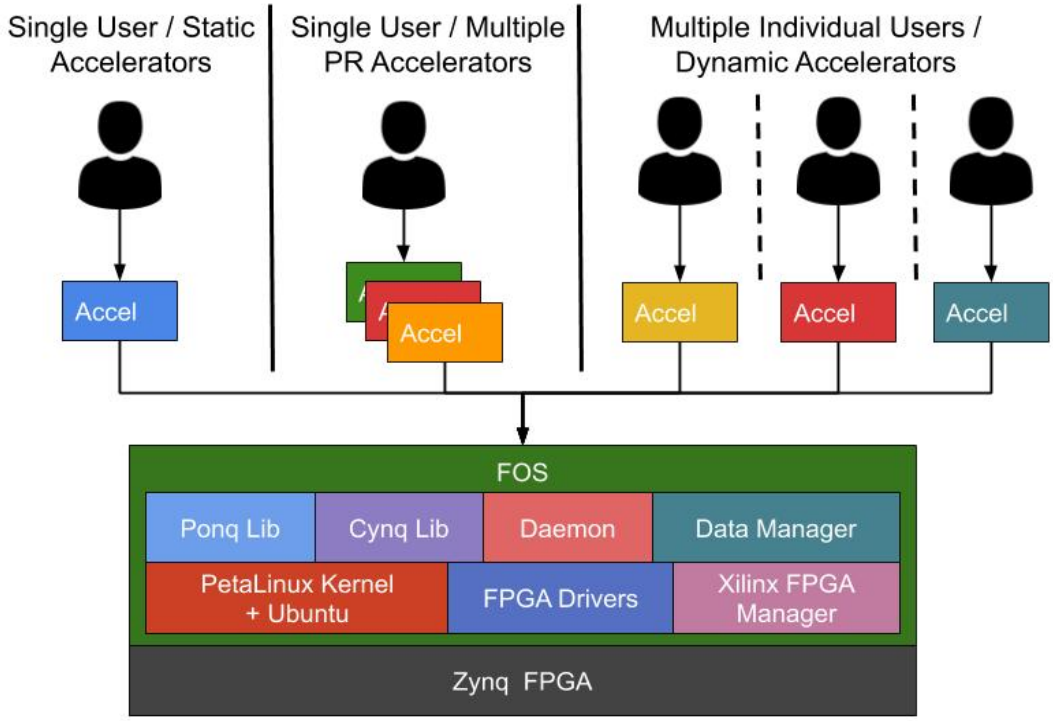


Figure 2.5: FOS extends the ZUCL framework with Linux integration, python libs, and C++ runtime management to provide support for: multi-tenancy (concurrent processes with hardware accel.), dynamic offload, GUI, network connection and flexibility. Recreated from Anuj *et al.* work in [141].

2.2 Accelerating the localisation task: FPGA *vs.* GPU

The (LPMP) localisation task requires autonomous vehicles to process visual information in hard real-time to feed the spatial working memory and ultimately, efficiently compute the vehicle's location. Quite simply put, the higher the speed of the vehicle, the higher the processed frame rate must be, in order to allow the decision-making system to perform safely and efficiently. Hence, decision-making relies on a very low-latency localisation task, especially at high speeds.

Table 2.3 compares a relatively high-end CPU+FPGA System-on-Chip (SoC), a Xilinx zc706 board, with two different GPU-based SoC embedded platforms: NVIDIA Jetson TX2 and Jetson Xavier. The table displays the host-accelerator latency overhead for data transfer

	Zynq-7045	NVIDIA Jetson TX2	NVIDIA Jetson Xavier AGX
Overhead (ms)	$7.02 \cdot 10^{-3}$	$5.70 \cdot 10^{-1}$	$1.28 \cdot 10^{-1}$

Table 2.3: FPGA *vs.* GPU comparison: Host-Accelerator data transfer overhead, in milliseconds. The latency was computed using 1000 roundtrips, each exchanging a 32-bit value.

roundtrips of very small data packets (a single 32-bit word at a time, which is representative of what our system must deal with, with an objective of very low latency). As can be seen from Table 2.3, while the Jetson Xavier system yields a much lower latency than the Jetson TX2, a 32-bit host-GPU roundtrip is still $\approx$ 100 times higher latency-wise than its SoC-FPGA equivalent. This confirms data transfers between host and accelerator tend to favour FPGA-based systems: several studies, *e.g.*, Qasaimehe *et al.* [114], observe between 100× and 10000× shorter latency or higher bandwidth when comparing CPU+FPGA *vs.* CPU+GPU (depending on which kind of NVIDIA platform is used). Likewise, power consumption is largely in favour of reconfigurable systems when compared to GPUs, as shown in Table 4.4 (see Section 5.3), which is in line with what Qasaimeh *et al.* detail in their work [114] (see Section 5.2 for more information). In addition, as we present in Table 4.4, the power consumed by N-LOC is around $\approx$ 3W for the whole system, whereas in the case of Jetson systems it is situated in the 7.5-15W range, *i.e.*, a Jetson system would consume between two and five times more power than our resulting hardware implementation of LPMP. Thus, this justifies our choice of using FPGAs as a platform to prototype and implement the bio-inspired neural architecture.

2.3 The Wizarde Platform

To help with prototyping embedded applications running on a heterogeneous system composed of multiple FPGAs, we have designed Wizarde, a custom board which aggregates several system-on-chips (SoCs). It is composed of 3 × 3 tiles, organised in a 2D mesh. Each tile is equipped with a Zynq XC7Z045, i.e., a SoC which embeds an ARM Cortex A9, which is a dual-core general-purpose processor; and a Xilinx Kintex-7 FPGA, which holds 350K programmable logic cells [52]. Wizarde's tiles are described in table 2.4. An important feature of Wizarde is the gigabit transceiver interface set between two neighbouring tiles. This will allow hardware tasks, i.e., their bitstream representation, to be mapped to different modules of the target FPGA(s) depending on the run-time context. For example, due to specific resource contention, a given reconfigurable region may not be available to a task which used to run on it. As a result, such a task may be run on a different available slot somewhere else in Wizarde. Besides, each tile is independent (equipped with USB, Ethernet, micro-SD, DRAM, etc.), but all tiles are connected to their neighbours through a gigabit transceiver.

2.4 Mixed architectures: SoC + FPGA

Heterogeneous computing is a key strategy to meet the requirements of many compute-intensive applications. However, current platforms which leverage both CPUs and FPGAs are commonly underutilised as scheduling is often constrained to a run-to-completion model or acceleration of a single application at a time [34].

Zynq SoC	A9 + Kintex-7
LUTs	218,600
FlipFlops:	437,200
Block RAMs	545
SMA connector	1 port
DDR3 SDRAM	1 GB (connected to **PS**)
DDR3 SODIMM	1 GB (connected to **PL**)
Gbit Transceivers	16
USB 2.0/UART	1 port
Gbit Ethernet	1 port

Table 2.4: Contents of a Wizarde tile, based on the Zynq xc7z045ffg900-2.

To tackle this challenge, our future aim within this dissertation project is to leverage a middleware or bare-metal software layer, in which we will include and propose a new scheduling policy to minimise the overhead caused by reconfiguration, and also to maximise the utilisation of FPGA resources by dynamically scaling the resource allocation (we aim to use Ker-ONE [146] as a Hypervisor). Moreover, FPGA technology is currently going through an exciting time as FPGAs are becoming essential components in critical applications and SoCs are now available for building sophisticated embedded systems that couple powerful ARM SoCs with FPGAs. However, despite these key advancements, heterogeneous runtime systems targeting FPGAs have not achieved their full potential, which occurs when orchestrating all the heterogenous resources together. A heterogeneous runtime system has to optimise four main scheduling problems:

- Device type selection: Which device (or combination of devices) an application should be executed on (i.e. deciding between CPU or FPGA)?.
- Partitioning: If scheduling on multiple devices, how much workload should be executed on each device? For instance, scheduling 25% of the threads on CPU and 75% of the threads on FPGA.
- Number of compute units: How many instances of compute units should be allocated for the task? For instance, the number of CPU cores or FPGA accelerators.
- Accelerator type selection: If a device has multiple implementation alternatives (e.g., FPGA accelerators with different micro-architectures or big. LITTLE CPUs), which implementation should be selected

2.4.1 Gigabit Transceivers Interface

To bring out the high-speed signals from inside the FPGA and interface with the real world, a needed demand for the use of transceivers is put in context. Compared to an approach using ordinary IO Pins for FPGA interconnection, it has several advantages: the provided bandwidth is very high while only a few wires are required [50]. Thus, to leverage this FPGA's features aspect, a pre-developed hardcore IP has been incorporated within our FPGA ecosystem development.

The LogiCORE IP 7 Series FPGAs Transceivers Wizard is a type of serial communication that will be used and already incorporated in the wizarde board, it provides the ability to automate the task of creating HDL wrappers to configure on-chip FPGA transceivers. The wizard's

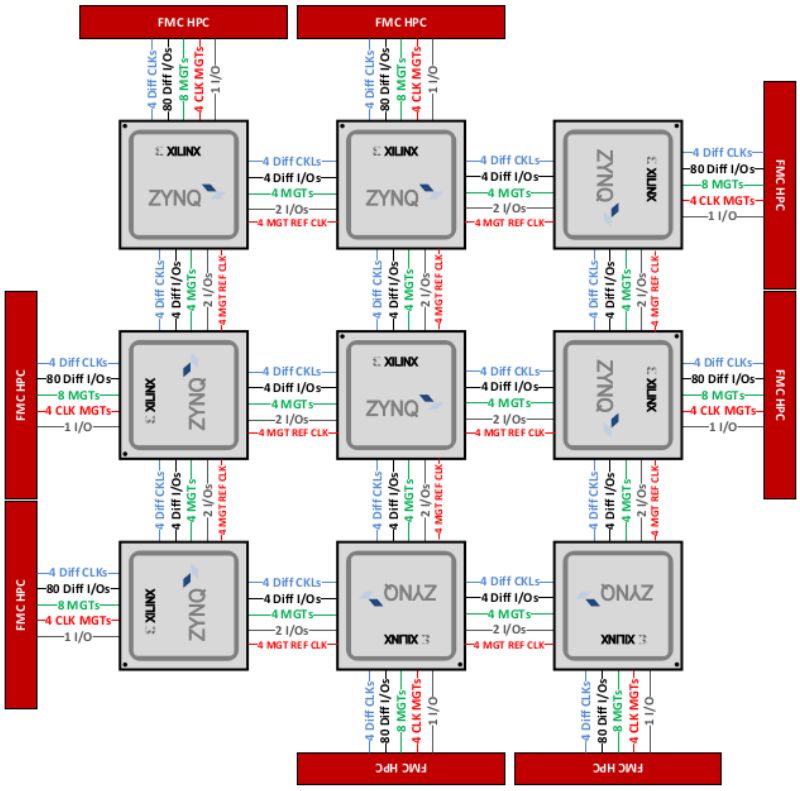


Figure 2.6: The Wizarde Platform. It is composed of 3 × 3 Zynq XC7Z045 SoCs, able to a system independently. Each tile is connected to the others through gigabit transceive 2.4).

customisation GUI allows users to configure one or more high-speed serial transceivers using either pre-defined templates supporting popular industry standards, or let us build a protocol from scratch [2].

2.4.1.1 Comparaison to related work

An interconnect framework for FPGAs based on multi-gigabit transceivers (MGTs), typically available in modern reconfigurable devices is proposed by Dreschmannetal *et al.* [50]. The framework provides higher bandwidth while using fewer pins, compared to existing approaches based on ordinary FPGA IO pins. Unlike other implementations using MGTs for device interconnection, special care has been taken to achieve high throughput and data integrity while keeping latency, resource usage and protocol overhead very low. Depending on the available MGTs, the bandwidth per connection reaches from 3.125 to 28 GBit/s, allowing large amounts of data to be moved quickly between multiple FPGAs [50].

Yangfane *et al.* present in their work a network on chip (NoC) emulation at the physical level [87], with two levels of interconnects that are adapted to mimic NoC on-chip communications: high bandwidth and low latency parallel on-board wires, and high-speed serial multi-gigabit transceivers, which is particularly important, as it helps the proposed NoC emulation platform to scale well as the NoC size increases.

Aloisio *et al.* shows that high-speed serial links are a key component of data acquisition systems for high energy physics [23]. They carry physics events data and often also a clock,

trigger and fast control signals. The authors demonstrated that the jitter on the clock recovered from the serial stream is a critical parameter since it directly affects the timing performance of data acquisition and trigger systems. While FPGAs include multi-gigabit serial transceivers, which are configurable with various options and support many data encodings.

2.4.1.2 Towards using GTX Transceivers for a data-transmission over Wizarde platform

As presented in the section 2.3, and based on different types of projects which try to incorporate the GTX Giga transmission into their implementation, see section 2.4.1.1, thus, according to the high transmission and latency of the data communication which can reach up to 12 GBit/s [50]. Therefore, a multi-FPGAs platform is constructed for prototyping embedded applications, and leveraging its features as GTX Transceivers will be an essential step to deploy the targeted localisation application.

Data in the Frame generator are sampled with ref_clk of 50 MHz before being sent to Northwest tile (NW) via MGT transceivers. The burst sending is expected with a frequency of 3.125 GHz. See figure 2.7

The different signals to be considered in the GTX transmission protocol, are as follows:

- Error_count: it should be NULL based on chosen frequency.
- DRP clock
- Rx_reset_done: should be at 1 when data are well received.

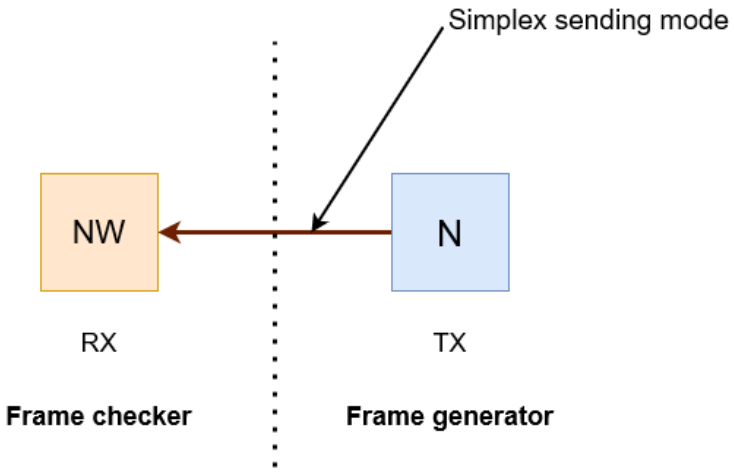


Figure 2.7: A simplex mode implemented on GTX transceivers as a first-stage protocol to communicate between two adjacent Wizarde tiles.

2.5 Accelerating Image processing and VPR model using Heterogeneous computing system

In the application domain of robot patrolling, the robot is considered as fully-autonomous and all the computing parts of the robot are designed as dedicated embedded systems [57]. The robot is not considered as a remote set of sensors but as a programmed and independent agent, taking its own decisions on the basis of learning [57]. That is the reason why Fiack *et al.* proposed a specific real-time and embedded vision system. Their system, and main contribution, are composed of two main parts:

- the Vision layer: a smart camera extracting visual features according to a bio-inspired attentional model,
- the Neural control layer: an embedded framework implementing higher cognitive tasks such as those implicated in the navigation scenario proposed in this paper.

In our thesis, we are interested in the vision layer called the Difference of Gaussian DoG, as it provides richness for this sense and and importance in the sensation-action loop [57]. Otherwise, Fiack *et al.* proposed a platform which consists in a specific hardware architecture (Zynq 700) that should provide intensive computation capabilities to deal with low-level image processing. The proposed architecture combines video sensing, image processing and communication into a System-on-Chip (SoC) embedded in the robot. This vision system is designed as a full hardware architecture deployed onto an FPGA device. Alongside, we incorporate the VPR LPMP model proposed by Colomer *et al.* in [43], as a bio-inspired neural architecture to process the keypoints or the salient points extracted from the visual processing DoG IP.

2.5.1 Difference of Gaussian DoG interface-based Image Processing for the localisation task of Autonomous Vehicles

The Intellectual Property (IP) DoG implemented by Fiack *et al.* [57] resorts to several types of operations, including gradients, as well as several differences of Gaussians (DoGs) operations. It provides pixel data of each landmark identified on the captured image, based on a sequence of differences of Gaussians. DoGs are used in multi-resolution methods to avoid expensive computations due to a filtering operation. The algorithm used to construct the processing phases of each level of resolution is detailed and evaluated by Fiack *et al.* [57] on FPGAs. Their IP is based on successive image filtering operations with 2D Gaussian kernels. It detects features in an image stream and then passes them to the central core as post-processing. More details see 2.9.

2.5.2 Integration of the Pyramid IP to a Wizarde's Tile

The DoG IP developed and implemented by Fiack *et al.* on Zynq-7000 FPGA family, determines the keypoints of images passed through a VITA-2000 camera module [57], also the project was carried out using ISE-2013 Xilinx tool.

The goal beyond this project is to leverage Wizarde platform and evaluate the logic it takes to synthesise the IP, but also how much logic is left for other designs on the same tile. Likewise, we shall plug the DoG IP in the larger context of a 3×3 SoC board called Wizarde (see section 2.3), by taking in consideration the constraint file for wizarde, thus the appropriate

device tree, internal branches & electronics configurations, and the same Hardware design and software application implemented in Zynq xc7z045ffg900-2. Likewise, The purpose beyond this project is to give the number of resources required for the Zynq ZC706 board to operate and support applications as depicted in 2.10. The architecture chosen represents one tile within 9 of Wizarde. As we have seen before, the organisation of the architecture is depicted in 2.8, It is composed of a chain of custom Intellectual Properties (IPs), written in VHDL, that takes its input from a camera through a streaming interface. The IP can be configured by the CPU part specifically by the memory-mapped interface. The IP chain generates several results which can be read back by the software part:

- A Difference of gaussian (DoG) or an intermediate processed image, selectable thanks to a dedicated register.
- The list of keypoints (the pixels information from the landmarks region of a processed image) extracted and sorted by the IP at the different frequency band
- The list of log-polar (is a coordinate system in two dimensions, where a point is identified by two numbers, one for the logarithm of the distance to a certain point, and one for an angle [10].) features associated to each key-point.

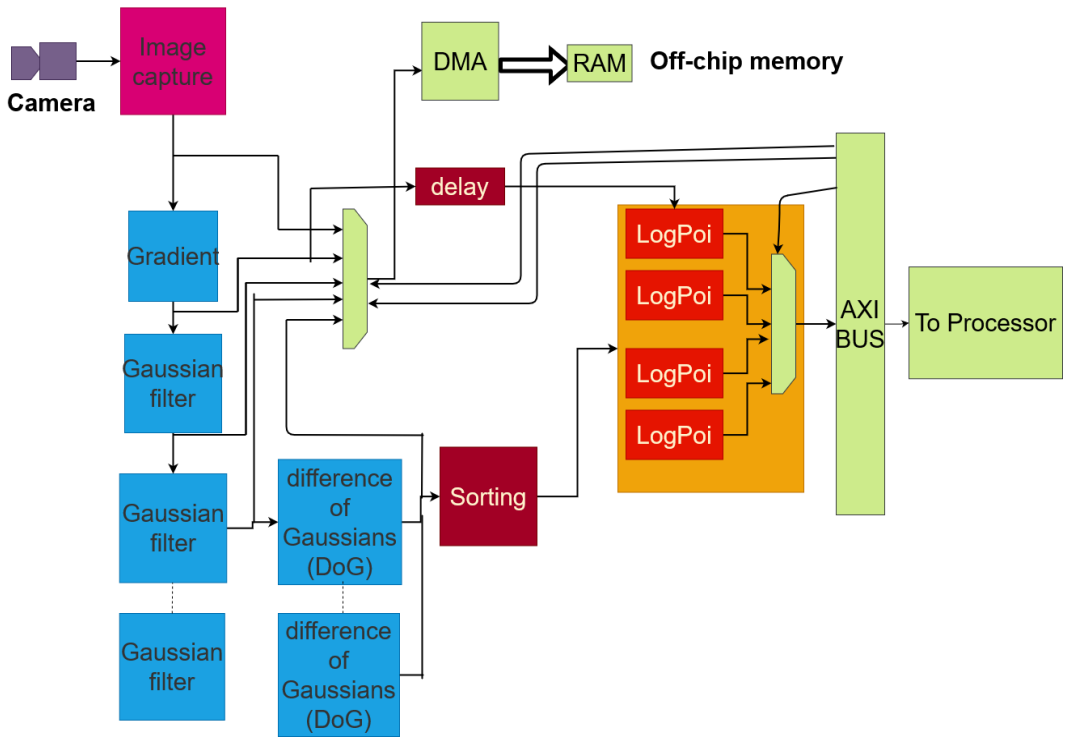


Figure 2.8: Global overview for scaling-based image-processing IP. The flow of pixels comes from the camera and goes to the CPU's memory through a DMA. An intermediate output can be selected thanks to a dedicated register. Another register allows selecting which feature to read. Finally, the keypoints can be read through the memory-mapped interface

The detected features are identified by their coordinates. Since most of the IPs follow a data-flow model of computation, they produce and consume pixels at inputs and outputs, respectively. Thus, the coordinates of the pixels are mandatory for each IP in order to carry out the needed calculation. Moreover, the learning of the visual cells for navigation needs to know the coordinates of the keypoints. The coordinates of the pixels must then be

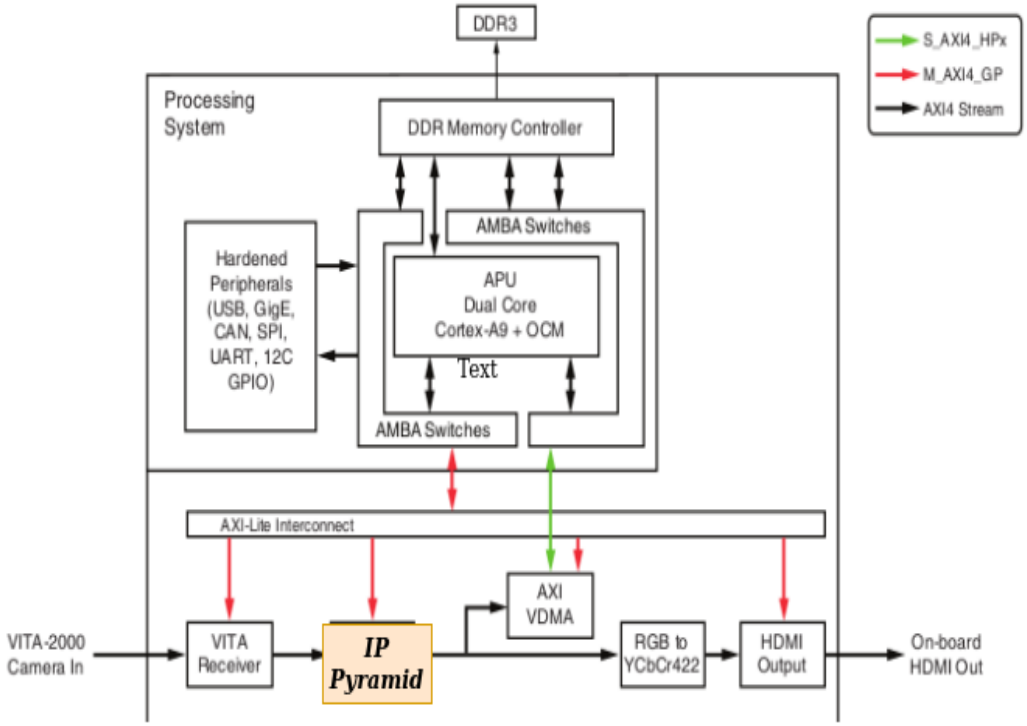


Figure 2.9: Block Diagram based on multiple chains of Hardware IPs for image processing DoG application, using IP Pyramid as a processing-core IP. Alongside other pre-existed Xilinx IPs to fed, output data as Axis-Stream-type information."

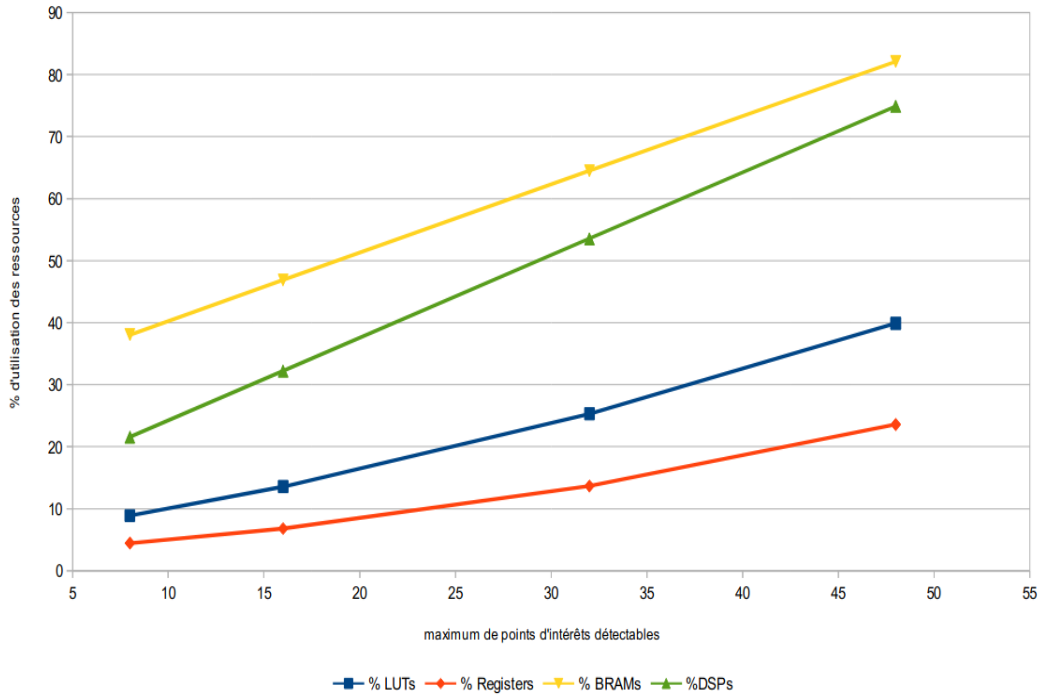


Figure 2.10: DoG IP resource consumption

transferred from one IP to one other. Across the pipeline, these data must be either delayed or regenerated by each processing IP depending on its latency. Furthermore, the results

shown below represent the first attempts our the design proposed to emulate and achieve the target specified:

2.5.3 Hardware implementation of a VPR model

Although significant improvements have been made to VPR models, they are still not widely used as the main source of localisation for navigation. Performing a navigation task on an autonomous vehicle requires a fairly high localisation frequency that most VPR models cannot achieve. For instance, a vehicle must provide a location very fast when it is travelling in fast lanes, so that it does not lose the navigation route. However, these constraints could be overcome by moving to a high-performance implementation.

In this work, we propose to leverage a VPR algorithm by moving to a hardware implementation. We are particularly interested in the VPR models using neural networks. Neural architecture has the advantage of being particularly well suited to hardware implementation, in particular by massively parallelising the computations performed by neural networks.

We thus propose to leverage the Log-Polar Max-Pi model (LPMP) [44, 43], a bio-inspired VPR model based on the use of a single conventional camera and neural networks based on a firing rate model. It has several interesting features for hardware implementation. First, LPMP is competitive with the state of the art, with high performance at small sampling scales. Second, it is a relatively simple model with a much smaller number of neurons than deep models. Third,although also based on a feedforward architecture, the model is not based on a succession of regular layers as in CNN's but is rather composed of a succession of neuronal populations with different characteristics. In addition, Colomer *et al.*'s reference implementation is purely software-based (using Python). Without any optimisation, it cannot scale to satisfy an autonomous vehicle's real-time constraints for localisation. It thus lends itself quite well to an FPGA implementation.

2.6 Conclusion

Based on what we have explored and presented previously, we have seen that FPGA is the common and appropriate choice for better prototyping of the run-time and energy-efficient application like the localisation using powerful and complex algorithms based on neural network layers, or even the bio-inspired neural architectures. Nevertheless, with the growing demand for the current approaches in terms of energy, and resource consumption for higher performance, it becomes quite urgent to find out a better strategy to implement such kinds of applications on dedicated platforms that can yield better efficiency and considerable performance. Alongside, leveraging pre-existed middlewares, or micro-kernels in which we can rely upon schedule and manage the placement of the tasks within the demanded specifications. Thus, Implementing the LPMP visual place recognition model onto a hardware IP is our next challenge to track, in the next chapter. However, we will demonstrate that even in one tile of FPGA, the localisation score accuracy still needs to be considered. Therefore, the need of scaling up the application onto a multi-FPGA board (the LPMP model in our case) is mandatory (see chapter 5).

3

Problem Statement

Electric autonomous vehicles have been the focus of many works in robotics and embedded systems over the last decade. Current solutions tend to favour LIDAR cameras for medium-range localisation, coupled with convolutional neural networks (CNN) [77]. For example, Gao *et al.* propose to fuse CNNs and vision algorithms with LIDARs [58].

However, leveraging LIDAR cameras for localisation is expensive and energy-consuming. A bioinspired artificial hippocampus algorithm was proposed by Espada *et al.* [53], which models a neural architecture based on a neurobotic model.

While the resulting neural network (NN) is large, there are not many layers between neurons, contrary to Deep Learning implementations— which simplifies the overall architecture, and gets rid of LIDARs. Espada demonstrates the potential of such an approach in unknown environments but also exposes the need to scale to be useful in a real-life context. Indeed, the original algorithm was implemented as pure software, on a (high-end) general-purpose processor. Previous work by Fiack *et al.* shows FPGA-based neural networks far outperform their software counterpart but quickly saturates available hardware resources [57]. Likewise, the image-processing part of the algorithm has already been implemented using FPGAs by Fiack *et al.*, and the results show a drastic increase in performance once specialised hardware is used [57]. Other FPGA-based solutions show they outperform other heterogeneous (CPU+GPU) solutions for (convolutional) NNs, in terms of both performance and energy consumption [85].

Hence, we intend to leverage a heterogeneous platform, composed of high-end general-purpose embedded processors and FPGAs. This will allow us to provide specialised circuits to process specific parts of the NN. The FPGAs will need to execute several tasks: extraction of salient points in images [57], as well as updating a large (pre-trained) NN implemented in hardware. Because the NN will be large, we will suggest one of the major contributions of such a scenario as well as a workbench to deploy the bio-inspired algorithm on multiple FPGAs. Furthermore, we rely on the Cortex A9s to perform local and distributed hardware and software task scheduling.

Basically, we start by implementing hardware tasks into one tile's FPGA then Measure latency & throughput in order to compare it with pure CPU implementations, then we get across multiple FPGAs / tiles. Once all of that is done, we fetch the neural network which fits into one tile's FPGA after we map a larger neuron which fits onto multiple tiles' FPGA.

Some challenges need to be overcome during our implementation, we aim to program the bio-inspired neural application using the HLS programming tool, and then, the implementation design will be carried out using the Vivado Xilinx platform. Thus, some challenges need to be tracked within this thesis, we outline them as follows:

- The main goal of my thesis is to deploy the whole neural architecture on the multi-FPGA board system. Scheduling the intra-inter communication through FPGA tiles is the main issue in our research. We will present a scenario in which we will set the CPU part as the master controller, and the FPGA tiles will carry out the accelerator IPs of the tackled application (whether it is the DoG image processing IP or the N-LOC bio-inspired neural ones). The signals on which the variables of the bio-inspired neural model need to model, have to be on fixed point values. By avoiding the float types to provide a better representation of the system, and by minimising the resource consumption of the used board. Also, the fixed-point representation gives the same degree of precisions as float one.

Part III

Contributions

4

MODELLING A BIO-INSPIRED ALGORITHM FOR FPGAS

4.1 Introduction

As we have discussed in chapter 3, Colomer *et al.* proposed a new way for the LPMP model to encode visual information based on the use of a sparse coding algorithm. It is a representation learning method which aims at finding a sparse representation of the input data (also known as sparse coding) in the form of a linear combination of basic elements as well as those basic elements themselves [14]. Inspired by the function of the visual cortex, it strongly compresses the visual information by keeping only useful information for localisation issues. In particular, this method was developed to build a code resistant to the translation undergone during the spatial movement of a vehicle [43]. Therefore, its use in the LPMP model has greatly improved the computational and memory cost of the system while slightly improving the model's localisation performance [43]. The LPMP model has been designed and deployed for large-scale environments, and also it reduces the computational and memory cost caused by its method of visual information encoding [43].

In addition, Colomer *et al.* have assessed the robustness of their model in dynamic environmental conditions (*i.e* subject to strong variations of human activity such as traffic, pedestrian activity, etc.), in order to determine whether and how the model should be improved [43]. Finally, they developed a new method for evaluating its performance based on its formulation as a model of Visual Place Recognition (VPR) and on the use of a more realistic dataset (we both use Oxford RobotCar Dataset to evaluate the model [90]).

In this chapter, our main focus will be the implementation of the localisation task on reconfigurable fabric specifically on field programmable gate array (FPGA). As discussed in chapter 3, the target neural architecture for Visual Place Recognition (VPR) will implement the LPMP bio-inspired model, as it delivers more reliability, and robustness of scores or results, and yields a low-cost energy-consumption on the term of application [44, 43].

In the remainder of this chapter, we depict the N-LOC architecture which is the LPMP model hardware implementation into three different essential blocs, the Signature Layer (SL), spatial working memory (SWM), and Place Cell (PC). Afterwards, a representation and discussion of the size of architecture alongside with resources it takes on FPGA implementation will be highlighted in detail.

4.2 Implementing N-LOC on FPGA

The LPMP mimics the hippocampus. N-LOC "only" implements the modelling LPMP which relies on a bio-inspired neural architecture implemented on the FPGA Zynq-7000 type of

family. As we showed earlier, LPMP performs visual localisation by mimicking the functioning of the mammalian brain [53, 44]. To localise an image, the architecture encodes these extracted landmarks (given by the image-processing DoG IP, see section 2.5.1) in a unique visuospatial pattern via several neural structures. Our resulting IP is composed of 3 stages: (1) the computation of the landmarks' visual signature via a winner-takes-all network (WTA) in the *signature layer* (SL), along with the computation of their angular position in the *azimuth layer* (AL); (2) merging of SL with AL via a *spatial working memory* (SWM); (3) computation of "place cell" (PC) activity via winner-takes-all for an appropriate localisation.

4.2.1 Visual signature computation

The computation of the visual signature landmarks relies on a WTA. It consists of a neural network and carries out input signals discrimination through competition. This WTA models cognitive properties, *e.g.*, decision-making, visual and auditory attention, and selective amplification. A WTA consists of a weighted average-based computation, with a post-tree-reduction selection, in order to only keeps the highest activity among activated neurons in that group of neurons. we denote SL as the signature Layer for the WTA computation.

$$S_i = 1 - \sum_{j=1}^{N_{pixels}} (|E_j - W_{ij}|)/N_{pixels} \tag{4.1}$$

Where i is the index of the neuron being considered, j is the index of the pixel being processed, W_{ij} is the weight of the i-th neuron processing the j-th pixel, and N_{pixels} is the total number of pixels per landmark.

4.2.2 Angular position computation

The computation of the landmarks angular position θ_l^{north} (or azimuth), where l is the l^{th} neuron in the AL vector and relies on the interpolation between the landmark angular position θ_{poi}^{ego} and the vehicle orientation $\theta_{Vehicle}^{north}$. It is described by eq. 4.2 [1].

$$\theta_l^{north} = \theta_{poi}^{ego} + \theta_{Vehicle}^{north} \pmod{2\pi} \tag{4.2}$$

θ_l^{north} is in radians, This information is encoded in the form of a population of neurons, a bio-inspired neural structure which encodes the current azimuth value in the form of an activity bubble. Besides, we fix the AL vector in our experiment, to focus only on the visual signature computation.

4.2.3 Spatial working memory

The spatial working memory (SWM) is a $N_S \times N_A$ pixel matrix. N_S is the number of neurons in the SL vector, and N_A is the number of angles considered in the model. In figure 4.3, $N_A = 4$, where each angle of 45° is coded by a group of neurons in the AL vector. In our actual experiments (see section 4.6.1), we set $N_A = 3$. We denote N_{SWM} the total number of values which compose the SWM. A is the number of groups in AL, with $A = 4$ (*i.e.*, each group holds 60 neurons). Eq. 4.4 gives the intensity of the il neuron where $s_i(t)$ is the signature layer's neuron activity, $w_{i,il}(t)$ is the weight value between SWM and SL, $a_j(t)$

[1] Each neuron encodes 2°. We need a population of 180 neurons for 360° of total camera angle.

represents the AL's neuron value, $w_{j,il}(t)$ is the weight value between SWM and AL, eq. 4.3 yields its final activity, where f is the activation function of Sigmoid which is applied to normalise the final results, it is used to determine the output of neural network like yes or no, thus, It maps the resulting values in between 0 to 1.

$$x_{il}(t) = f(x_{il}(t-1) + I_{il}(t) - x_{il}(t-1).In(t)) \tag{4.3}$$

$$I_{il}(t) = (s_i(t).w_{i,il}(t)).(\max_{j \in N_a l}(a_j(t).w_{j,il}(t))) \tag{4.4}$$

The connection weights between AL and SL are assigned to 1 in the SWM. The execution workflow of the second WTA is described in figure 4.3.

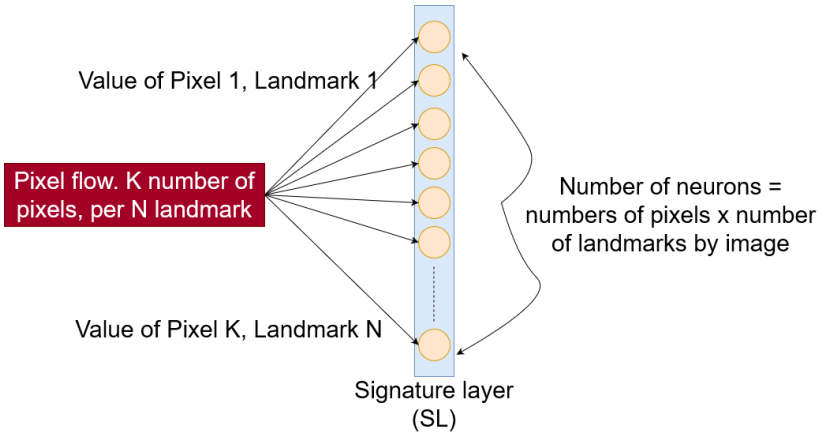
4.2.4 The place cell neurons group

The activity of the SWM matrix characterises the current location. It is memorised in a *place cell vector* (PC) of N_P neurons, which will then be fed to another WTA. This is in reference to "place cells neurons" found in the hippocampus which has a close activity [102]. N_P refers to the maximal number of images that N-LOC can memorise. Eq. 4.5 gives the activity of the neuron i at time t, we denote that by $P_i(t)$, as $W_{ij}^{\mathrm{SWM}} - x_j(t)$ is the weight value between SWM and PC, and $x_{il}(t)$ is SWM's neuron value obtained from the Eq. 4.4. Each neuron in PC holds connections related to learned images: the activity of one neuron gives the similarity between a learned place and an image. Thus, it eventually provides the appropriate information about the current location to the localisation system.

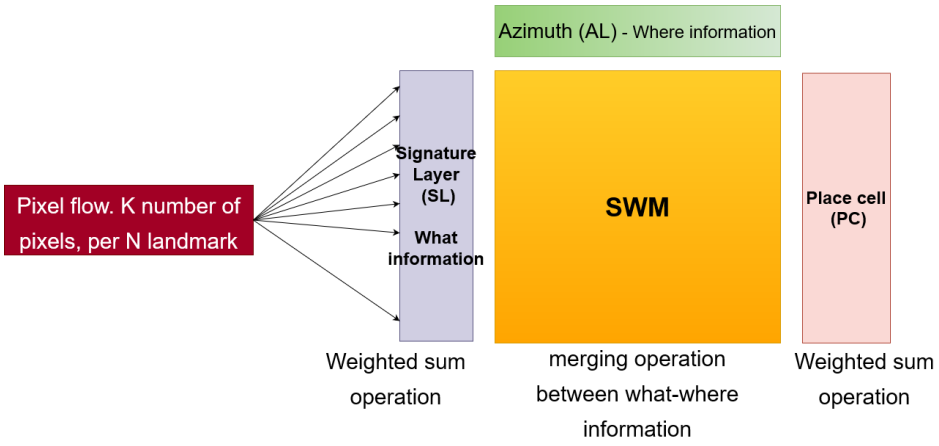
$$P_i(t) = 1 - \sum_{j=1}^{N_{\mathrm{SWM}}} (|W_{ij}^{\mathrm{SWM}} - x_j(t)|)/N_{\mathrm{SWM}} \tag{4.5}$$

4.2.5 Modes of operation

The N-LOC model has two modes of operation for the localisation process. The first is *learning*, where the connection weights of its different components can be updated to memorise new images. It is triggered when the autonomous car starts the localisation process, or when the car enters a new location. Thus, a set of images must first be streamed to the neural IP to extract features and compute the weights of the connections between neurons and pixels at each stage. See figure 4.1 for more details about the learning mode. The second is *using*, where the connection weights are fixed. This is where the actual assessment and evaluation of the accuracy and performance measurement of the environment localisation per each captured image throughout camera VITA [30] occurs. Thus, we aim to have the maximum score rate for the selected sparked neuron, which represents a previously learned image. See figure 4.1 for more details about the using mode.



(a) Learning mode



(b) Using mode

Figure 4.1: The figure 4.1a refers to the learning mode where the values of weights between the signature layer and pixel flow are provided by the values of the pixel of each landmark that the image contains. The figure 4.1b indicates the using mode, where the values of all weights between layers are fixed, however, the neurons are invited to do different types of calculations, in the signature layer the weighted sum operation is performed, in SWM layer an activation function with correlation between signature layer' neuron value, and azimuth layer' neuron value, the place cell is weighted sum type of operation to choose the score and appropriate landmark according to the evaluated input.

4.3 Evaluation of the LPMP model on real-time constrained environments

Colomer *et al.* [43] conducted different types of experimentations on real-time constrained environments (three types of weather were encountered: sunny, cloudy, overcast) with the VEDECOM vehicle on the Satory track, to evaluate the navigation performance of the model.

Table 4.1 shows that 5 sessions of experiments have been performed in total, after stabilisation of a functional version of the model, in order also, to evaluate and assess the LPMP model by Colomer *et al.*. Among the five experiments, four types of weather were encountered: sunny (once), partly cloudy (once), overcast (once) and cloudy (twice). It also shows that

the attendance of the track has always been basic, except for experiment 4 where various vehicles were moving on the track.

The table 4.1 shows that out of the 11 trials performed, 6 can be considered as successful. A trial on each of the target trajectories has been passed, proving that the model can work in all three environments. It also shows that most failed tests occurred when the weather was cloudy (experimental sessions 3 and 5), illustrating the difficulty of the model to work during this weather. Cloudy weather is particularly hard for a visual navigation system, which has to deal with very important light variations in a short period of time. Difficulties were encountered in adjusting *the BlackFly camera* [4] to keep the brightness level more or less constant (the used camera, although being quite powerful, is very sensitive to the variation of luminosity which strongly modifies incoming images).

Trajectory name	Trial index	Experiment session index	Weather	Total distance in m.	Total duration in s. (learning duration)	Number of corrections	Number of place cells	Best distance in m.	Average place recognition accuracy in m. (std)	Mean distance to target trajectory in m. (std)
nloop	1	1	Partly cloudy	2672	2221 (933)	10	51	1367	4.36 (3.69)	1.54 (1.8)
	2	2	Sunny	X	X	18	48	∼2734	X	X
	3	3	Cloudy	628	355 (121)	3	49	156	4.51 (8.74)	1.14 (0.93)
	4	3	Cloudy	691	401 (115)	4	47	128	7.38 (10.39)	1.77 (1.18)
	5	4	Overcast	1108	567 (112)	6	41	228	4.66 (5.46)	1.36 (1.05)
	6	4	Overcast	1163	657 (162)	8	44	147	4.23 (4.29)	1.22 (1.04)
	7	4	Overcast	3154	1531 (171)	11	45	1660	4.43 (3.64)	1.54 (1.21)
	8	5	Cloudy	1701	755 (60)	17	46	194	4.94 (4.92)	1.44 (1.19)
HallA	9	1	Partly cloudy	3600	1613 (168)	6	97	422	4.52 (12.76)	0.8 (0.75)
	10	5	Cloudy	1625	584 (172)		100	90	2.74 (1.59)	1.4 (1.1)
Parking	11	5	Cloudy	1103	1218 (171)	10	65	79	3.68 (2.53)	1.86 (1.66)

Table 4.1: **Results of the LPMP model on the Satory circuit type of testing environment at VEDECOM institute** . Three different trajectories were tested: a looping trajectory *nloop*, a long trajectory *HallA* and a trajectory in a closed area "Parking". Std stands for standard deviation. Recreated from Colomer *et al.* in [44].

4.4 Fixed-point arithmetic

Traditionally, when targeting reconfigurable hardware, fixed-point values are used to represent decimal values. Our implementation uses 8-bit values, with 2 bits for the integer part, and 6 bits for the fractional one. This coding is motivated by the fact that all neurons' weights and values are between 0 and 1. We have experimented with different resolutions, i.e., 8, 10, 12, and 16-bit fixed-point numbers. Our experiments showed no significant improvement in accuracy for place cells activation. Our target device is maintained for the same project as in the image processing, Xilinx Zynq-7000 SoC ZC706.

Using a fixed point representation is proposed rather than choosing the floating-point type, which consumes and requires a lot of hardware resources even when dealing with small types of operations. Therefore, fixed-point representation is efficient and justified both in terms of accuracy and resources consumption [45, 22], the normal way of representing the split between integer, fractional bits in a vector with a fixed-point number is x,y where x represents the number of integer bits and y the number of fractional bits.

For our implementation, we have selected 8 bits as the total representation of signals type, "2,6" represents 2 integer bits and 6 fractional bits. This format is often called Q format, sometimes denoted $Q_{m.n}$, where m represents the number of integer bits, and n represents the number of fractional bits. Figure 4.2 illustrates the explained Q format. According to our experimentation, we have noticed that even though we change the representation of the Q format (increasing the bits in both integer and fractional), will maintain the same score and accuracy in the final results, with $\approx 93\%$ for all different Q format representations (according to our experimentations). However, it will drastically impact resource consumption, thus, the "2,6" fixed-point(2 integer bits and 6 fractional bits) is the optimal thread-off for this selection. We use the *ap_fixed* hls library for fixed-point features.

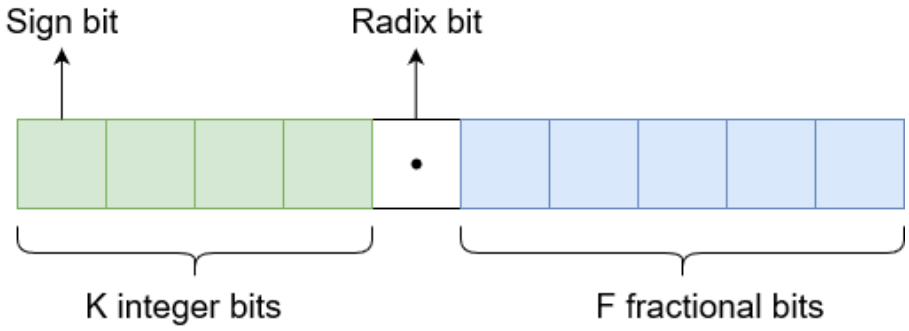


Figure 4.2: An example is shown for the fixed-point format of Q_{KF} based on two's complement.

4.5 Use of HLS to implement N-LOC

Several academic and industrial efforts have been devoted in order to increasing the productivity of FPGA-based designs by means of using High-Level Synthesis (HLS) tools [22]. The HLS approach in Electronic Design Automation (EDA) is a way in the design flow aiming at moving the design effort to a higher abstraction level. Although the first generations of HLS tools failed to produce efficient hardware designs, different reasons have motivated researchers to continue improving these tools [45]. Among these reasons, we can mention:

- The emergence of IP-based design approaches.
- Trends towards using hardware accelerators on SoCs.
- The time-to-market constraint usually presses to reduce the design time.

As FPGA offers a tremendous number of logic cells on a single chip, digital design for such huge hardware resources under time-to-market constraint urged the evolution of High-Level Synthesis (HLS) tools [22]. Today, several existing HLS tools have shown their efficiency in producing acceptable design performances and shortening time-to-market [103, 94]. Thus,

several HLS optimisation steps have been proposed in the state of the art, in order to improve the system's performance.

We outline here a pre-evaluation and initiative to go forward by using the VHDL programming tool first. However, according to the complexity of design and the difficulty of leveraging the appropriate pragmas to carry out the different possible optimisation. Moreover, to explore the possible optimisation steps that could be done in order to achieve an efficient hardware implementation, further see table 4.2.

4.5.1 Methodology for Implementing the LPMP Model on FPGA Platform

The VHDL code generation process for the LPMP model leverages High-Level Synthesis (HLS) techniques to facilitate efficient development. The implementation of the LPMP model involves two crucial phases.

Firstly, the storage of pixels is accomplished by utilizing the available Block RAMs (BRAMs) and Look-Up Tables (LUTs). This phase focuses on efficiently organizing and managing the pixel data.

Secondly, the acceleration of neuron calculations occurs during the "Using" phase by parallelizing the computations with the aid of
`#pragma HLS unroll factor=NUMBER_NEURONS_IN_LAYER`.
This pragma directive enables the efficient utilisation of hardware resources and enhances the computational performance. `NUMBER_NEURONS_IN_LAYER` represents the signature layer vector which contains neurons that carry out same calculation.

To optimize the signals and vectors of neurons and weights, specific directives are employed. For example, to utilize the LUTRAM efficiently, the code incorporates the directive
`#pragma HLS resource variable=NEURONS_LAYER1 core=RAM_1P_LUTRAM`
in the appropriate section of the code. Additionally, for efficient memory allocation, the code implements
`#pragma HLS ARRAY_PARTITION variable=NEURONS_WEIGHT block factor=10 dim=1`
. . . to partition the array into different BRAM blocks.
Furthermore, to ensure proper synchronisation between neuron layers, the following directive is employed:
`#pragma HLS pipeline`.
This directive enables efficient pipelining of computations, enhancing the overall performance of the design.

4.6 Experimental results

This section presents the results of our implementation: resource utilisation, latency, and throughput performance. Results were obtained using Vivado's HLS framework v2016.4, coupled with a custom workbench. The design was implemented on a ZC706 board, which features the same Zynq-7045 SoC that Wizarde uses, See chapter 3. This board was the target used in Vivado to obtain performance measurements, resource consumption, and power consumption estimates. We used the parameters described in section 4.

4.6.1 Experimental setup and implementation parameters

The target hardware platform is based on the Zynq-7045 system-on-chip, which features a dual-core ARM Cortex A9 processor, coupled with a Kintex-7 FPGA (see Table 2.3 for additional details). The Processing System (PS, featuring the dual-core Cortex A9) runs a bare-metal executable which streams pixels to the programmable logic (PL). N-LOC was designed using Vivado HLS, i.e., using the OpenCL language with FPGA-specific *pragmas* to specify bus sizes, etc. The PL implements our N-LOC IP, which was synthesised and implemented on the FPGA part of the Zynq SoC. Hence the performance results shown below are obtained using an actual implementation of our design (not a simulation). Moreover, we used Vivado 2016.4 to synthesize the IP on our FPGA. The Wizarde board was validated with this toolchain and we have not yet qualified and validated Wizarde with a more recent Vivado version (in our future work, we will generate a device tree for a Linux distribution which is compatible with a more recent version of Vivado). Resource-wise, we compared the synthesis reports of Vivado 2016.4 and 2019.3. The difference is below 4%. While the synthesis process of a more recent version of Vivado may yield better resource usage in general, we believe the nature of our IPs would see only marginal gains compared to Vivado 2016.4 in our specific case. Moreover, the number of DSPs would remain the same, even in parts of the IP which can take advantage of them (e.g., the WTA part).

Additionally, for the purpose of software reference evaluation, we utilised the NVIDIA Jetson TX2 as the hardware platform to execute the software application. The characteristics of the hardware platform, as well as the versions of the different packages employed to run the Python-based software code on it, are defined below:

- NVIDIA Pascal™ Architecture GPU, 2 Denver, 64-bit CPUs + Quad-Core A57 Complex.
- DDR4 Memory & 8 GB L128 bit.
- OS (Ubuntu 18.04).
- CPython 3.0, OpenCV 4.6.

Ideally, we would like to plug the image-processing IP (which performs the pyramidal decomposition of the acquired images) directly to N-LOC. However, in order to correctly isolate the N-LOC part of the processing chain, we opted to use a small bare-metal executable which will stream images taken from the Oxford dataset [90]. This will also prove useful when we evaluate the distributed N-LOC architecture, as it also requires a minimal runtime system to orchestrate data synchronisation between multiple N-LOC instances (see Section 5.5.1).

Hence, a set of parameters is given to obtain around 90% accuracy when comparing the value of computed place cells after the second WTA and the value in pre-learned images. As explained in Section 4.4, accuracy does not change significantly when increasing the fixed-point number format to a higher number of bits. The initial training/learning phase uses as many images as there are place cell neurons. Our bio-inspired IP uses the following parameters:

- On the target Zynq-7045 chip, the signature layer (SL) can fit at most 1600 neurons; the SWM 4800 neurons; the place cells vector 100 neurons, with 16 landmarks per image.
- The azimuth layer (which represents the angle orientation of each captured image) contains 180 neurons.

- The values of azimuth neurons are fixed for this experimentation; varying azimuths will be added in the future.

- We parameterised N-LOC with three configurations: 30, 60 and 90 place cell neurons. They represent the number of images that need to be learned in the localisation process.

- We used 100 images to "feed" N-LOC.

- The maximum of neurons that we can accommodate on FPGA Zynq-7045 is 180, as the last one, saturates the resources in both LUT and BRAM, see section 4.2 for more details.

We compare our bio-inspired neural IP with Colomer's Python-based application [44] to build the place cells vector. As will be detailed in Section 4.6.2, we considered two versions of the software implementation: the baseline and its optimised version. Both make use of Cython, numpy and OpenCV to speed up computations. However, the optimised version of the software implementation is also parallelised and makes better use of Cython to help guide it toward more efficient code generation. As detailed in Section 2.5.1, image pre-processing is performed either using OpenCV in the Python script or in the image processing IP. A previous version of this model was validated on large datasets, as shown by Espada et al. [53]. Iterations over LPMP led to an optimised code, which was tested in a mobile robot in a real-life environment (closed tracks) to evaluate the performance of a software LPMP implementation. As Colomer et al. show [44, 43], the algorithm is accurate and correctly identifies landmarks. However, this implementation does not go fast enough to scale at higher speeds. In addition, only the *learning* and *using* phases leading to building the place cell vectors are evaluated; i.e., we do not time the video pre-processing which does the gray-scaling and applies difference of Gaussians and Log-Polar conversion performed by the previous component in the pipeline. As we described above, the reference code is written in Cython, with uses of NumPy and OpenCV where appropriate. Our application directly embeds all (grayscaled) images to exploit and as a result we subtracted the time taken by the Python application to perform all the processing prior to the actual *learning*/*using* phases, i.e., I/O operations to read image files, putting colour images to gray scale, dividing images into landmarks, etc. These operations represent roughly 20% of the total execution time in the baseline and optimised software implementations. We used different learning configurations: the system had to learn using 30, 60 and 90 gray-scale images taken from the Oxford dataset [90] to train the system (*learning* mode), then used 100 images in total in the *using* mode. Each image has a resolution of 640×400 pixels. Python results were obtained with NVIDIA Jetson TX2 and followed the same principles for the learning/using ratio.

Colomer et al. already showed how the LPMP model behaves with large datasets [44]. In particular, real-life environment data obtained on close tracks using a mobile robot running the software version of LPMP were gathered and validated the LPMP model's accuracy and precision [43]. The (optimised) reference code correctly identifies the right landmarks in a real-life context. Moreover, our hardware implementation also selected the same landmarks in our experiments as the software reference implementations. Hence, to compare our hardware implementation to the software implementation used by Colomer, we consider it is sufficient to resort to the Oxford dataset and run it on both software and hardware implementations to compare the two and check how faithful N-LOC is to the original software implementation.

4.6.2 Resource utilisation

This implementation requires ≈ 26% of available LUTs, and ≈ 50% of BRAM blocks. The amount of BRAM used is consistent with the format used for the SWM, which is essentially a dense 1600×3 matrix of neurons yielding 8-bit values. Table 4.2 provides details for specific resources required to implement our neural IP to process 8 or 16 landmarks, respectively. We cannot consume more resources to increase the number of neurons because space must be made to also fit the image processing IP.

Results numbers were obtained using Vivado's HLS framework v2016.4, coupled with a custom workbench. The design was implemented on a ZC706 board, which features the same Zynq-7045 SoC that Wizarde uses. This board allowed us to obtain performance measurements, resources consumption, and power consumption estimates. We used the parameters described in Section 4.2. Further, we also used those parameters to implement the neural IP.

8 landmarks							
# neurons	50	80	90	100	150	170	180
LUTs (%)	11.68	15.56	18.08	19.79	29.12	31.99	27.41
BRAMs (%)	16.70	31.38	49.72	53.39	86.42	93.76	132.29
FLIP FLOP (%)	2.69	1.67	2.85	2.89	3.16	3.27	2.03
DSPs (%)	1.33	0.33	1.33	1.33	1.33	1.33	1.44
16 landmarks							
# neurons	50	80	90	100	150	170	180
LUTs (%)	16.73	24.51	28.37	31.41	41.75	46.67	52.32
BRAMs (%)	35.05	60.73	97.43	104.77	110.28	124.95	150.41
FLIP FLOP (%)	3.21	2.55	3.45	3.50	2.13	4.16	4.76
DSPs (%)	1.33	0.33	1.33	1.33	1.67	2.37	2.89
32 landmarks							
# neurons	50	80	90	100	150	170	180
LUTs (%)	33.62	48.21	55.45	60.90	88.19	99.08	104.53
BRAMs (%)	24.04	94.50	94.50	94.50	376.33	376.33	376.33
FLIP FLOP (%)	2.68	1.33	2.71	2.71	2.75	2.75	2.76
DSPs (%)	1.89	2.44	1.89	1.89	1.89	1.89	1.89

Table 4.2: Resource consumption estimation for 8, 16, and 32 landmarks, depending on the number of neurons in the place cells layer. Only 50 and 80 neurons are shown for 32 landmarks: resources saturate beyond that number. The results were obtained using Vivado HLS 2016.4, and they provide an estimation of the post-synthesis outcome.

We conducted our experimentation using three different *learning* configurations, to compare both software and hardware approaches. We set the numer of place cell neurons at 30, 60, and 90 respectively, and then test and evaluate the *using* phase on 100 images. Our FPGA-based implementation outperforms the baseline Python-based reference implementation [44], with an average throughput of 52 images/s *vs.* 7 images/s. Moreover, the LPMP algorithm performs 50961 operations per image for 16 landmarks of one processed-image. Thus, by

multiplying those values, we get 0.032 GFLOPS/s for the Python implementation), and 2.3 GFLOP/s for our FPGA-based solution. The learning phase latency is 6× shorter, and for the using phase, the latency is 9× shorter, with a total throughput ≈ 7× larger. See Table 4.3 for more details about the performance comparison.

A traditional Python implementation is around 10 times slower than sequential implementation with close-to-the-metal languages such as C. The first implementation of Colomer *et al.* is not pure Python: it resorts to Cython, which translates Python code into C and compiles it natively, and also makes use of numpy and OpenCV, which are written in C and C++. In their second implementation, processing neurons in SL, SWM, and PC structures is parallelised in the *using* mode. As a result, it is much faster than the sequential reference implementation we use as our baseline.

In term of *learning* latency, the optimised LPMP reference improves significantly its processing performance compared to the baseline. N-LOC still yields better throughtput and latency compared to the optimised version (≈ 9× lower). The optimised version stores all of the image pixels in one shot, whereas we stream them one by one. However, while the optimised version does perform twice as well as the baseline for the *using* latency, NLOC still outperforms this optimised version, as its latency is $\approx 2-3\times$ lower ($\approx 9\times$ compared to the baseline). The *learning* phase is dominated by data (pixel) transfers, whereas the *using* phase is significantly more intensive computationally speaking. Thus, as the number of image comparisons grows in the *learning* phase, the *using* phase takes progressively longer in time [44]. Our implementation of the *using* phase's latency is ≈ 9× lower than the baseline, and $\approx 2-3\times$ lower than the optimised version. Consequently, the throughput of the optimised LPMP version is $\approx 1.5-2\times$ faster than the baseline when considering total throughput, but N-LOC's total throughput is itself $\approx 3-4\times$ higher than the optimised LPMP version (and ≈ 7× higher than the baseline). All the details of our experiments are shown in Table 4.3. Each image is composed of 16 landmarks, of resolution 12×12 pixels.

The figure 4.4 illustrates the increase in the number of resources used in the FPGA for each experimentation in which we varied the number of neurons in place cells for each N-LOC IP during the design implementation using the Vivado tool. Our analysis revealed that the LUT and BRAM were the primary resources consumed. This can be attributed to the N-LOC architecture's reliance on both storing pixel values on neuron weights and performing simple mathematical operations on LUTs rather than using DSP.

Table 4.2 and figure 4.5 provide the various metrics we have considered. Multiple experiments have allowed us to assess the accuracy of our visual recognition process compared to prelearned images following Colomer's model. It is predicated on the number of landmarks per image. Table 4.5 shows resource utilisation of the overall application implementation. Hence, a resource consumption trade-off must be considered to make the design fit the board.

Table 4.4 details the power footprint of the whole hardware-based application (image processing and neural IPs). The overall footprint (image processing IP *and* N-LOC IP) consumes 2.75W. This is to be compared to the nominal power consumption of NVIDIA Jetson TX2 used for our experiments (with a power consumption of 7.5W–15W).

Number of neurons in place cells	30	60	90
N-LOC			
Learning: Latency (ms)	2.6	2.6	2.6
Using: Latency (ms)	11.4	20.3	29.2
Total Throughput (img/s)	82	45	31
Total Power consumption static + dynamic (W)	≈ 2.8	≈ 2.8	≈ 2.8
Baseline reference			
Learning: Latency (ms)	17.76	17.86	18.99
Using: Latency (ms)	100.32	123.93	146.66
Total Throughput (img/s)	9	7	6
Total Power consumption static + dynamic (W)	$\approx 7.5 - -15$	$\approx 7.5 - -15$	$\approx 7.5 - -15$
Optimised reference			
Learning: Latency (ms)	17.16	17.10	17.48
Using: Latency (ms)	61.11	66.05	72.60
Total Throughput (img/s)	13	12	11
Total Power consumption static + dynamic (W)	$\approx 7.5 - -15$	$\approx 7.5 - -15$	$\approx 7.5 - -15$

Table 4.3: N-LOC: Performance and efficiency. The system is configured with different number of place cell neurons, which corresponds to the number of neurons to be learned. The system is then evaluated and tested with 100 images for different place cell configurations. Results are generated using NVIDIA Jetson TX2 platform for software reference (using all 6 CPU cores when possible for the optimised version), and an FPGA ZC706 board for N-LOC Hardware implementation. The used frequency in FPGA in both 30 and 60 PC neurons is 100 MHZ. For 90 neurons, the frequency is set it 70 MHZ to satisfy timing constraints.

(IP) Block	Description	Power (W)
Image processing	Image acquisition and landmark identification	0.783
Bio-inspired Neural accelerator	Neuron activation ; place cell recognition	0.141
Processing system	Hardcore processor (ARM Cortex A9)	1.567
Total power on chip (static + dynamic)		2.749

Table 4.4: Total power consumption (static + dynamic) generated by all integrated IPs. Results obtained with 100 place cell neurons (maximum of neurons to be trained), and 16 landmarks per image. We used Vivado's power estimator.

IP block	Slice LUTs	BRAM Tile	FLIP FLOPs	DSPs
Image processing	29647 (13.6%)	255 (46.8%)	30553 (7.0%)	298 (33.1%)
Single N-LOC	62377 (28.5%)	164 (30.0%)	7879 (1.8%)	4 (0.4%)

Table 4.5: Resource utilisation for each integrated IP, implemented on one tile of Wizarde, for 100 neurons of place cell neurons group, and 16 landmarks per each image. The raw values are provided, along with the percentage they represent between parentheses.

4.7 Conclusion

We proposed a low-footprint and high-performance accelerator for feature and image recognition in the context of autonomous vehicle navigation. It leverages a bio-inspired algorithm which extracts pixel values to perform feature extraction and image recognition, by porting a previously software-based process and designing a hardware-based one, which relies on the difference of Gaussian operations. Compared to previous (highly accurate) implementations, ours provides not only accuracy, but also very low latency (13× shorter than the Cython reference implementation), low power consumption (30× lower), and high frame rate (10× higher).

Our experimental results show that the proposed accelerator achieves a speedup of **10x** compared to Cython-based software and a power dissipation of approximately **2.141W** which is 30× more efficient than a high-end CPU.

Unlike previous proposals, our solution tends to a high run-time speed compared to a software implementation, and with comparable accuracy. Furthermore, we propose a novel solution to implement this new bio-inspired neural IP that has the lead to give functionality control for a mapped system over a tested environment.

In the next chapter, we will be demonstrating the capability of duplicating the neural architecture on one FPGA, and some performance results from that perspective. Then, the interest of duplicating the architecture on multiple FPGAs, using the Gigabit transceivers GTX provided by Xilinx vendor.

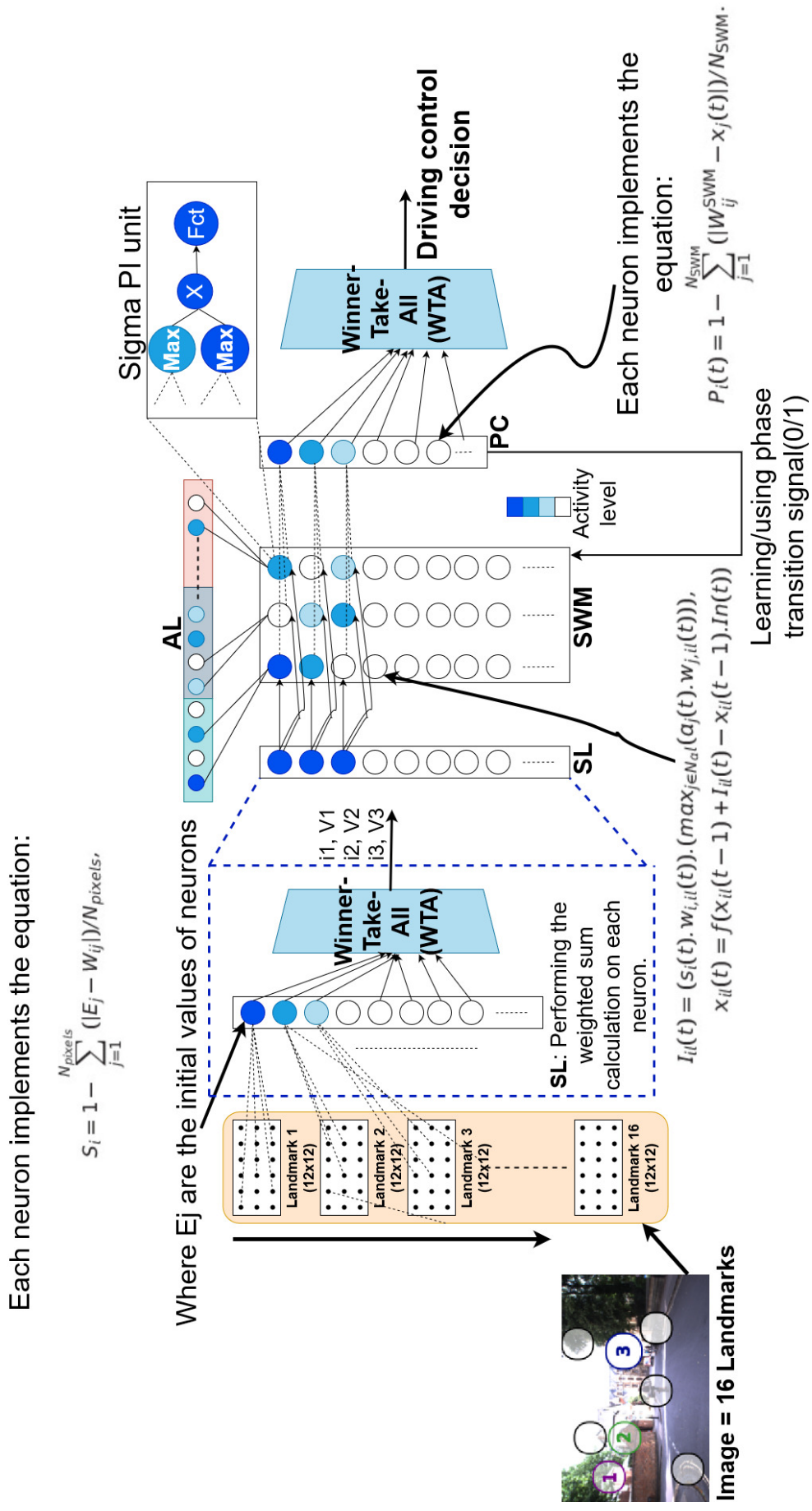


Figure 4.3: Overview of our bio-inspired neural architecture. On the left-hand side, the Pyramid IP identifies and sends keypoints information to the bio-inspired neural IP, pictured on the right-hand side of the figure. Data is sent pixel by pixel, for each landmark of each image. The Learning Mode processes 10 images for each time period. If a consensus is found (*i.e.*, the measured error is acceptable), the system then switches to Using Mode.

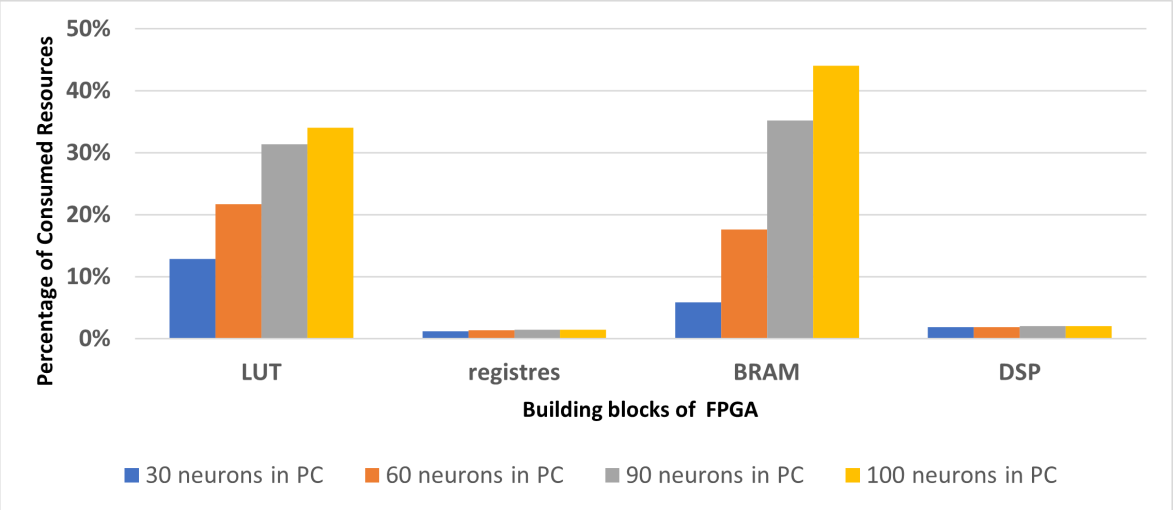
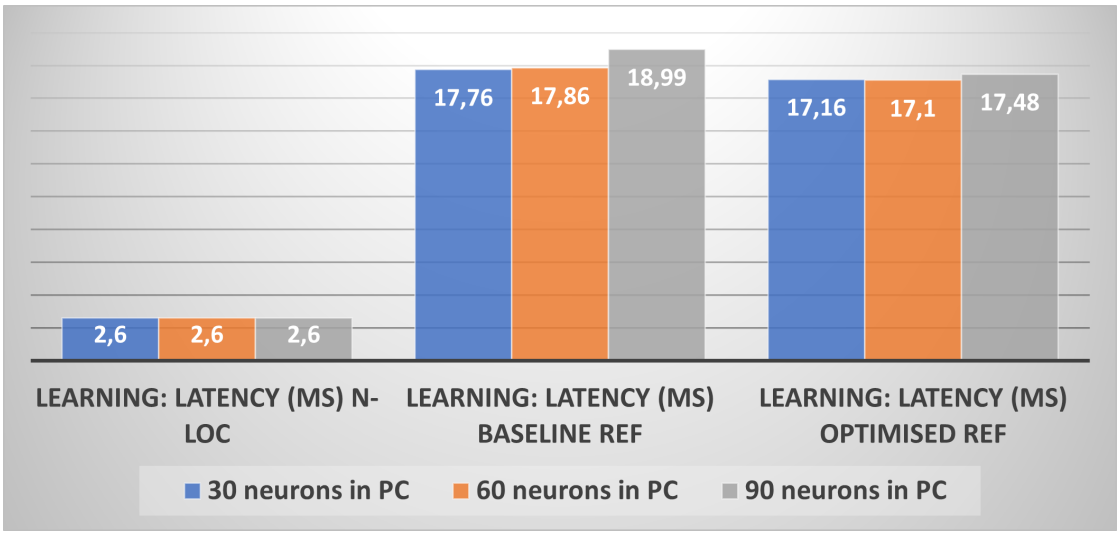
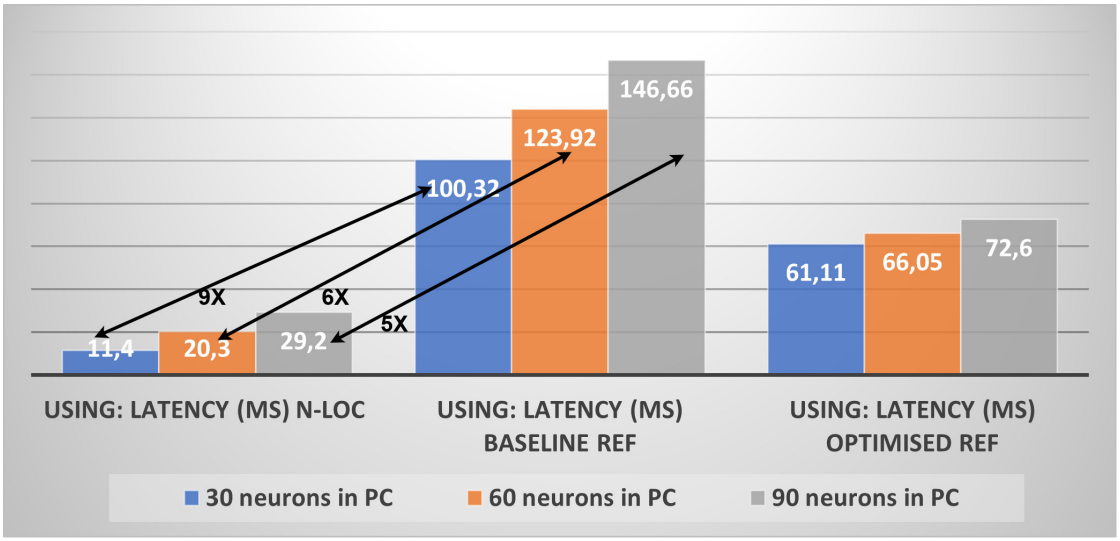


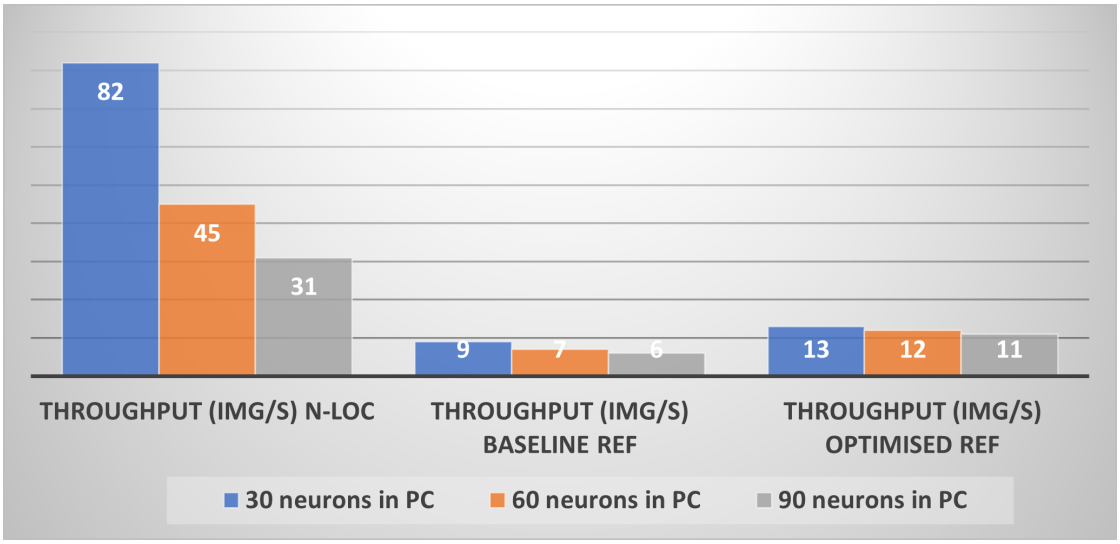
Figure 4.4: Resource consumption in term of LUT, BRAM, registers and DSP, for configurations with different numbers of place cells.



(a)



(b)



(c)

Figure 4.5: Figure (a) illustrates the comparison of learning latency using different neurons in PC. Figure (b) demonstrates the comparison of latency using different neurons in PC. Figure (c) presents the comparison of throughput performance. The learning latency for N-LOC is 2.6 ms, while it remains consistent at 18.99 ms for both the baseline and optimised reference. Based on our experimentation, we have achieved a navigation speed of up to 200 km/h specifically for N-LOC processing.

5

A DISTRIBUTED N-LOC ARCHITECTURE

5.1 Introduction

The LPMP bio-inspired neural network (NN) architecture was implemented on one FPGA board, and as seen in table 4.2, the maximum number of neurons (in the Place cells group) that we can fit in one FPGA board is 180 neurons (after many optimisations on the HLS-based programming application), with an accuracy $\approx$ 95 and $\approx$ 30FPS. As that number of neurons in PC, is the maximum that we can accommodate in our FPGA board ZC706, therefore a need to scale up the NN architecture and deploy it through multiple ones is a must.

First and foremost, expanding the neural architecture of the targeted localisation application means increasing the capacity of self-driving vehicles or mobile robots to self-localise themselves based on a pre-learning of such environment that they could find themselves in it. Thus, the efficiency of the model relies strictly on the number of neurons that contains. Meanwhile, a large neural architecture implies a wide and efficient place-recognition figure 5.1.

Therefore, we need to exploit a large FPGA board to prototype the same application with a large number of neurons, for instance, providing 500 neurons in place cells, see chapter 4. As said before, as maximum as we provide a large FPGA board, it will be limited by the size of the application, thus, a need for multi-FPGAs to explore the portability and how we can leverage those boards to communicate the data between each FPGA. In addition, Wizarde will be used as a targeted platform (see figure 2.6) to mimic our proposed solution for prototyping. In this chapter we then demonstrate the possibility of how we deploy the N-LOC IP on multiple tiles of FPGAs, we will as a result, it is necessary to explore the possibility of producing a *distributed* version of N-LOC, which could scale as needed, by GTX Transceivers which requires communication between FPGA tiles.

A reminder of this contribution and chapter is as follows: we present, and demonstrate the need for scalability of the bio-inspired neural architecture by distributing the N-LOC IP on different FPGAs. We start by exploring and prototyping the proposed concept on one tile of FPGA, then we show how it is possible to do so on multiple ones. As our bio-inspired neural architecture hardware implementation is the first ever attempt to propose such an implementation. as well as for our distributed N-LOC hardware IP on a single zc706 FPGA board, including its software controller. We will present also a distributed N-LOC for a multi-FPGA board with gigabit transceivers type of communication to communicate data through a certain type of specified and established scenarios.

5.2 Comparison with related work

Overall, a growing autonomous vehicle market needs to implement tasks such as visual navigation, object detection, etc. Hence, a broad summary with various software and hardware-based implementations, running on CPU, GPU and/or FPGAs, is detailed in various surveys [100, 155, 70, 128, 115].

One of the major venues to deal with autonomous vehicle navigation is the use of Machine Learning and Deep Learning. Deploying a Deep Learning model directly to edge devices comes with many advantages compared to traditional cloud deployments: by eliminating communications, inter- and intra-processes can reduce latency and reliance on the network connection. Since the data never leave the device, edge-inference helps with maintaining user privacy. Moreover, since the amount of cloud resources is drastically reduced, edge-inference can also reduce ongoing costs [79, 144].

The porting of ML applications running on edge devices both drives and is driven by the development of specialised hardware accelerators such as GPUs, ASICs or FPGAs. FPGAs are dominating and attracting people to this research domain, thanks to their steadily improving performance, internal expanded bandwidth and high throughput [26]. A lot of works and benchmark instances are proposed to implement CNN, NN circuits with all required features for, e.g., Xilinx FPGA platforms [101, 21], which define a benchmarking approach to co-design, construct and optimise any such algorithm into an inference accelerator IP [140, 34].

Another way to implement the navigation process is to resort to bio-inspired models and algorithms. In this context, spiking neural networks (SNNs) and visual place recognition (VPR) models serve different purposes. SNNs, such as the temporal neural encoder (TNE) proposed by Kheradpisheh et al. [99], are inspired by the spiking neurons in the brain and can encode sensory information in the form of spike trains. This allows SNNs to process and recognise temporal patterns and sequences, which are particularly useful for navigation tasks that require tracking of moving objects or path integration. LPMP, as currently designed, does not include temporal sequences (yet), but provides a much simpler model, which in turn makes it easier to follow a hardware–software co-design approach, like the one we used for this work, as the complexity of the neural network is lessened compared to SNNs. VPR models are particularly useful for global localisation tasks in which the robot needs to determine its position relative to a known map of the environment [139].

Hence, our approach relies on a bio-inspired VPR model, which, by contrast with ML/DL models, has a "neural circuitry" which is closer to what can be found in nature, i.e., the way we model individual neurons is not significantly closer to what DL models do, but the structure of the network itself follows more closely what can be found in a mammal's brain: there are no hidden layers, etc. The resulting neural network is simpler in its structure, but may result in a less memory-efficient way of storing information if implemented naïvely. The LPMP approach (and its hardware implementation) also differs from more "traditional" bio-inspired spiking algorithms in that it relies on recognising visual similarities.

Cuperlier et al. have shown how it could be beneficial to implement a neural processing unit as an IP onto FPGA-based reconfigurable fabrics for an embedded navigation application [47, 28]. This is what led us to propose a hardware-based implementation of their bio-inspired algorithm.

Beyond the use of an accurate and precise model, there is the question of providing an implementation that is sufficiently fast to be useful in real life. Hence, the use of accelerators such as

GPUs and FPGAs is an important area of research for navigation algorithm implementation to be embedded in vehicles, with performance and energy-efficiency in mind. Qasaimehet et al. in [114] conducted a comprehensive benchmark of the run-time performance and energy efficiency of a wide range of vision kernels in order to determine which embedded platform is most suitable for their application. The conducted study is performed for three commonly used hardware accelerators for embedded vision applications, ARM57 CPU, Jetson TX2 GPU and ZCU102 FPGA using the vendor-optimised vision libraries OpenCV, VisionWorks and xfOpenCV. The results show that the GPU achieves an energy/frame reduction ratio of 1.1×–3.2× compared to the others for simple kernels. However, for more complex kernels and more complete vision pipelines, the FPGA outperforms the others with energy/frame reduction ratios of 1.2×–22.3×. They report also that the FPGA performs increasingly better as a vision application's pipeline complexity grows.

A publicly available chart summarising neural network accelerator performance and power consumption has been made available by the Energy Efficient Computing Group at Tsinghua University, China [67]. It would be interesting to see where our system fits in this chart.

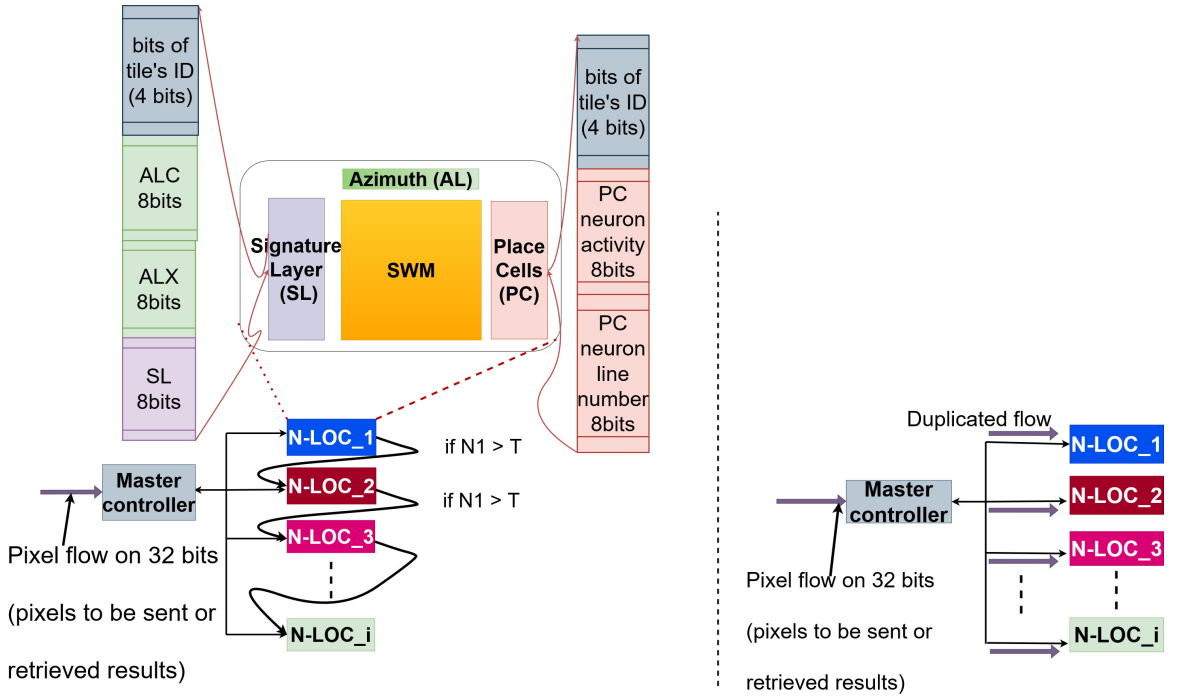
5.3 Distributed N-LOC: Principles of Learning and Using modes

In this section, the possibility of distributing the N-LOC architecture across one and/or multiple FPGA tiles is explored. As discussed in the previous introduction, expanding the neural architecture of the localisation tasks will give rise to the AV's capacity to localise itself mechanically. For instance, a larger neural network implies a larger pre-learning environment and a larger online learning capacity in general (see section 4 for details about how the bio-inspired algorithm implemented in N-LOC works). Hence, a large neural architecture results in a wider and more efficient place recognition task.

5.3.1 Principles of the LPMP models

The LPMP model is presented in section 2 in section 1.5, and implemented in Hardware N-LOC IP (see previous chapter). The last implemented one resort to two main phases which are the learning and phasing modes. We will describe each in detail, from a distributed point of view.

As detailed in chapter 4, these two phases help the N-LOC IP extract the features from pixel streams/image landmarks, copy some of them in the NN architecture, and finally choose which place cell should be activated in the scene. By introducing two modes of how the N-LOC IPs can process and communicate the data with each other, the master controller which is a C-based programme implemented on the ARM A9 cortex part of the Zynq will be responsible for organising the data path and sending commands into all the post-implemented N-LOC IPs. Basically, we have two different modes of operations within this N-LOC deployment, the *Learning phase* and *using phase*, as seen in section 4, the N-LOC IP extracts the features from the image landmarks, and copy it to the weights values of the NN architecture, then, we have the using mode through which we assess the application and by which the mobile-robot or the AV can explore the environment.



(a) Learning mode (b) Using mode

Figure 5.1: Expanding bio-inspired neural network architecture N-LOC over Wizarde multi-tiles. In the learning phase, the data copy of each current landmark will be held on one bloc IP. If the number of learned neurons in N-LOC1 is overloaded (superior to a fixed threshold T), we will switch to the next available bloc IP which is the second one. Then, it will be the same rule for all different bloc IPs. In the process phase, all bloc IPs work simultaneously, the best score among the three represents the accurate and appropriate localisation of a given image.

Learning phase in a distributed N-LOC environment, the *Learning* phase workflow is represented as follows, the N-LOC IPs are duplicated and distributed on different reconfigurable regions. After setting that up, we connect the pixel stream which contains the pixel information of the landmark, to the appropriate N-LOC block if its neurons' weights are not saturated (we set T as the maximum of neurons that a bloc N-LOC IP can contain). If an N-LOC block is saturated during the learning phase (*i.e.*, the maximum number of neurons to initialise has been reached), we then switch to the next available N-LOC block, to carry on the ongoing or further learning of different captured images.

Using phase likewise, its workflow is: the pixel stream is connected through all N-LOC blocks simultaneously. Then, all N-LOC blocks *simultaneously* will perform different computations based on different pre-learned information. Thus, a threshold-based comparison is set by the master controller, to select the highest activated neurons among the N-LOC blocks. See figure 5.1, the Using mode specifically.

In order to organise this proposed scenario, and ensure the distribution of the N-LOC through the different reconfigurable fabric, also the way learning and using modes alternates between each other according to the need of the vehicle exploration over the environment. Thus, a master controller is proposed, who has the responsibility to be in charge of communicating with all N-LOCs, using a bidirectional communication protocol. See figure 5.1, the learning mode specifically.

A proof of concept of this distributed architecture was implemented on a single Zynq-7045's programmable logic, with 3 N-LOC block instances. We next discuss the possibility of using gigabit transceivers to enable fast communications between GTX users.

5.3.2 Implementation

We conducted some different experiments to show how possible we can distribute the N-LOC hardware architecture over multiple N-LOC IPs. First of all, we evaluated the proposed scenario of duplicating and dividing the whole neural architecture, as described and illustrated in figure 5.2, on one FPGA fabric. The obtained results were compared to the results of one whole implementation of N-LOC, as referred to in section 4.6. In the next section, we show that it will be as promising to use a distributed version by duplicating it through multiple FPGA fabrics as the number of resources, as well as latency performance, are considerable compared to one single N-LOC implementation.

The master controller will be responsible to schedule and organise the communication and implementation of those IPs. It will be at the processing part of the Zynq. In the algorithm explained here 1, we dissect the adopted policy to make this arrangement.

In order to ease the implementation, we used AXI-LITE as a type of bus to interface between the three N-LOCs IPs and Processor Zynq in the PS part. As the AXI-LITE is a type of bus through which we send data per clock cycle. Afterwards, we incorporated the AXI-STREAM as a type of communication between IPS, also, the transceivers are equipped with such interface, thus, it is required to make all the interfaces in similar communication.

Algorithm 1 Algorithm for a master controller of N-LOC distribution

Result: Algorithm for a master controller of N-LOC distribution

```
while Image's landmarks are successfully processed do
    if mode learning is activated then
        We duplicate the same neural architecture onto different tiles.
        Pixel flow is connected to the appropriate N-LOC bloc if its neurons' weights are
        not overloaded.
        Moreover, once we saturate the neural architecture, we switch to the next available
        one, to carry on the ongoing or further learning of different captured images.
        The master controller is responsible to communicate with all N-LOCs, in bidirec-
        tional communication.
        switch to the using mode.
    else
        Pixel flow is connected through all N-LOC blocs simultaneously.
        All N-LOCs perform different computations based on different pre-learned informa-
        tion, simultaneously.
        Threshold-based comparison will be set by the master controller, to select the highest
        activated neurons among the N-LOCs.
        if score of activated neuron in PC < FIXED_THRESHOLD then
            switch to the learning mode
        else
            keep navigating in the using mode
        end
    end
end
```

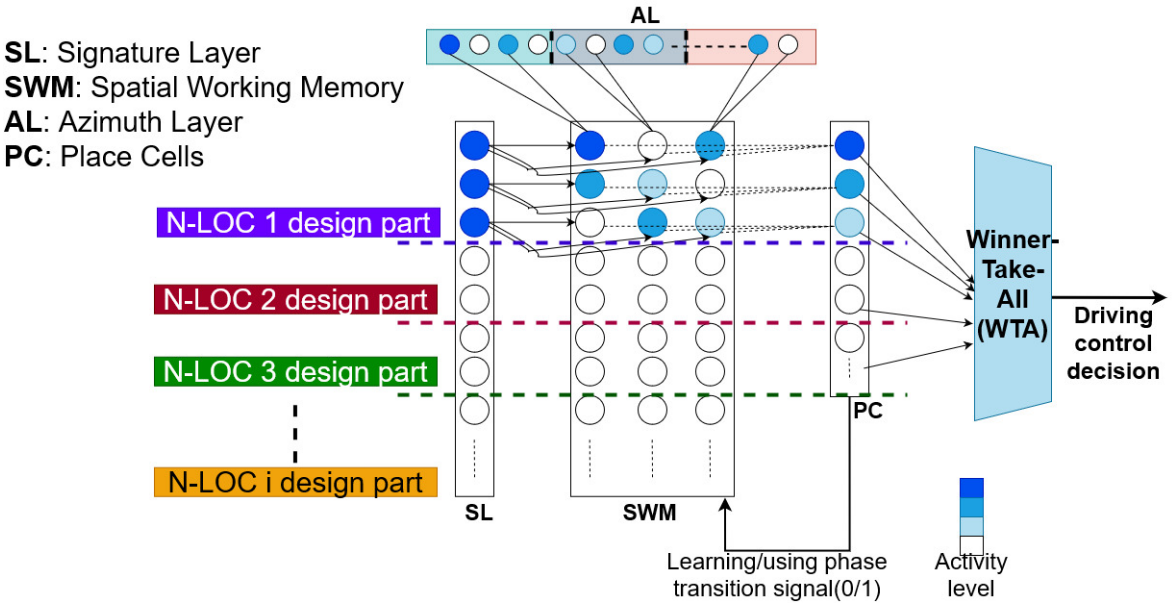


Figure 5.2: The overall N-LOC distribution model over different reconfigurable fabrics within Wizarde. Each N-LOC IP takes a part of the whole architecture, as all the N-LOC IPs have the same number of neurons in their decision.

5.4 Experiments with distributed N-LOC on a single FPGA

Experimental setup: we experimented with our distributed N-LOC model by instantiating three N-LOC blocks on a single Zynq-7045 SoC. The controller is implemented in software on a bare-metal ARM Cortex A9 microprocessor. The FPGA and Cortex A9 are linked through an AXI lite bus. Our distributed N-LOC system is composed of 3×N-LOC instances of 30 place cell neurons each, for a total of 90 PC neurons. As demonstrated in chapter 4, we show that with up to 180 neurons we can saturate our FPGA. For ease of implementation and to reduce the time of synthesising and experimentation in the Vivado tool, we decide to implement less than 100 of neurons in N-LOC IP. Hence, the total number of neurons in these 3 place cells of N-LOCs is 90, a three blocks IPs of 30 of each, will be compared to the total one which contains 90 neurons in its place cells.

As described earlier, we compare our results with a Python program which makes use of NumPy, OpenCV, and is compiled with Cython. The experimental conditions are the same as described in section 4.6.1.

Experimental results Table 5.1 shows the latency and throughput calculated from both the Learning phase (where a single N-LOC is active at a time) and the Using phase (where all N-LOCs compute in parallel) are assessed at the same time. The latency performance in one learning phase is roughly the same, whether the N-LOC system is distributed or not, as the learning phase pixels path is actually performing sequentially. The Using phase fares better with a distributed N-LOC system when processing a single image, the performance gain is caused by the simultaneous calculation made by the three N-LOCs at the same time, in contrast to a single full N-LOC. In both cases, the global throughput to both learn images and use this knowledge to localise the vehicle is at least an order of magnitude better than with the reference application. We propose here a merit factor metric

	Baseline Ref	Optimised Ref	1×90 N-LOC	3×30 N-LOC
Learning Latency (ms)	18.99	17.6	2.6	3.57
Using Latency (ms)	146.66	72.60	29.2	9.81
Total Throughput (img/s)	6	11	31	70
Merit factor (Throughput/Power)	$\approx 0.4 - 0.8$	$\approx 0.73 - 1.46$	14.75	26.53

Table 5.1: Timings for 90 place cell neurons: Python reference code ("Baseline Re"), optimised multicore version ("Optimised Ref"), a single large N-LOC instance, and a distributed 3×N-LOC architecture (3×30 neurons), implemented on a single Zynq-7045 SoC's FPGA part. The controller is implemented on the Cortex A9 as bare-metal software.

to assess the performance of the system in terms of throughput per power consumption. The merit factor serves as a valuable indicator, reflecting the efficiency and effectiveness of the targeted system or application. We have observed that a higher merit factor corresponds to significant gains in throughput while minimising power consumption. In our experiments, we compared the performance of different configurations, specifically the 1×90 N-LOC and 3×30 N-LOC setups, against an optimised software reference. We noticed that both the 1×90 N-LOC and 3×30 N-LOC configurations outperformed the optimised version by substantial margins. The 1×90 N-LOC configuration exhibited a considerable improvement, achieving a merit factor that was up to 10× higher than the optimised version. Similarly, the 3×30 N-LOC configuration displayed a significant increase in performance, surpassing the optimised version by a factor of up to 18×. These results highlight the significant advantages of the proposed configurations in terms of merit factor. Hence, this metric provides a comprehensive evaluation of system performance, enabling us to make informed decisions about system design and optimisation strategies for current and future designs.

Moreover, while the Learning phase only copies pixel values as weight into neurons' edges, the Using phase performs much more computationally intensive operations, as there is a winner-take-all (WTA) stage to update the signature layer (SL), then an update of the spatial working memory (SWM), and once the image has been fully processing – *i.e.*, in our case, once all sixteen landmarks which compose an image have been processed, yet another WTA operation takes place to select the most active place cell neuron and decide if the measured score is high enough (*i.e.*, has reached the preset value threshold). Hence, the Using phase latency is bound to be much higher than the Learning one.

Speedups	1×90 N-LOC	3×30 N-LOC
Learning Latency	8	4
Using Latency	3	7
Total Throughput	**4**	**6**

Table 5.2: Single and distributed N-LOC: speedups. Baseline: the optimised reference Python application compiled with Cython.

Table 5.2 provides speedups of a monolithic and a distributed N-LOC systems *vs.* the optimised software implementation. Compared to the optimised reference application, N-LOC

is 4-6× faster, but compared to the individually measured Learning and Using latency, this performance is rather low. It is important to note that in the reference code, as the number of place cell neurons increases, the processing time also increases dramatically: the learning latency reported in Table 4.3 is 60% higher than the 1×90 configuration shown in Table 5.1; the Using latency is 16% higher; and the total image throughput is 7% lower. While we must make use of additional FPGA units to extend the size of our network, the intrinsic parallelism used in the various phases ensures that image processing latency remains relatively constant; the only true bottleneck is the communication between the processing system (PS) and the programmable logic (PL).

In general, the main bottleneck in the N-LOC hardware implementation is the naïve implementation we made, where we isolated the N-LOC instances as much as possible, but which results in multiple AXI-Lite roundtrips between the processing system (PS) and the programmable logic (PL). A more involved architecture would have the PS only send messages once for broadcasting, with a hardware-based broadcasting designed internally to carry the data frames to each N-LOC instance. However, this approach also has drawbacks: it makes the overall architecture more "rigid," which in turn may hamper the capacity of the system to scale with several N-LOC instances, *e.g.*, up to eight, or even nine, if we target the Wizarde platform. Further, one of the inherent difficulties dealing with FPGAs stems from inherent issues related to (reconfigurable) hardware and the use of HLS: as we grow from 60 to 90 place cells configuration, in order to maintain acceptable timings and clock distribution within the system, we must reduce the clock frequency from 100MHz to about 70MHz. This is a limitation tied to a relatively naïve approach in our own design, and we plan on exploring ways to increase the clock frequency to improve performance.

Power-wise, both the PS and PL parts of the target Zynq SoC see a slight power consumption increase, as shown in Table 5.4. This is not unexpected: on top of sending pixels to the FPGA, the Cortex A9 core is now also tasked with selecting N-LOC instances during the Learning phase, but also to send the pixel stream to all instances during the Using phase. Likewise, each N-LOC instance requires proportionally more FPGA resources compared to their single N-LOC counterpart. The resulting total power consumption (static + dynamic) is around 2.8W, with ≈ 0.2W for the hardware part. Compared to NVIDIA Jetson TX2 board used to run the reference program, this is a 5.5-6× improvement.

# Neurons	Slice LUTs	BRAM Tile	FLIP FLOPs	DSPs
1×90 N-LOC	68209 (31.2%)	192 (35.2%)	3039 (0.7%)	18 (2.0%)
1×30 N-LOC	28869 (13.2%)	32 (5.9%)	2614 (0.6%)	17 (3.4%)
3×30 N-LOC	84198 (38.51%)	96 (17.61%)	8199 (3.75%)	51 (10.2%)

Table 5.3: N-LOC: resource usage. The percentage of a given resource usage on the Zynq-7045 is given between parentheses. Each processed image contains 16 landmarks.

Table 5.3 shows the resource utilisation of the overall application implementation. The 1×90 N-LOC instance requires fewer resources than its 3×30 N-LOC counterpart: it requires 20% fewer LUTs, 63% fewer flip-flops, and 65% fewer DSPs. However, the relatively large dense memory matrix required by a monolithic 90 place cell neuron network requires a complex

BRAM usage by synthesizer, and BRAM usage is twice as large as with 3×30 neurons. For the DSP part, this is a limitation tied to the needs for computations of a single IP: with a single 1×30 N-LOC block, we reach almost the same amount of DSPs used than with 1×90.

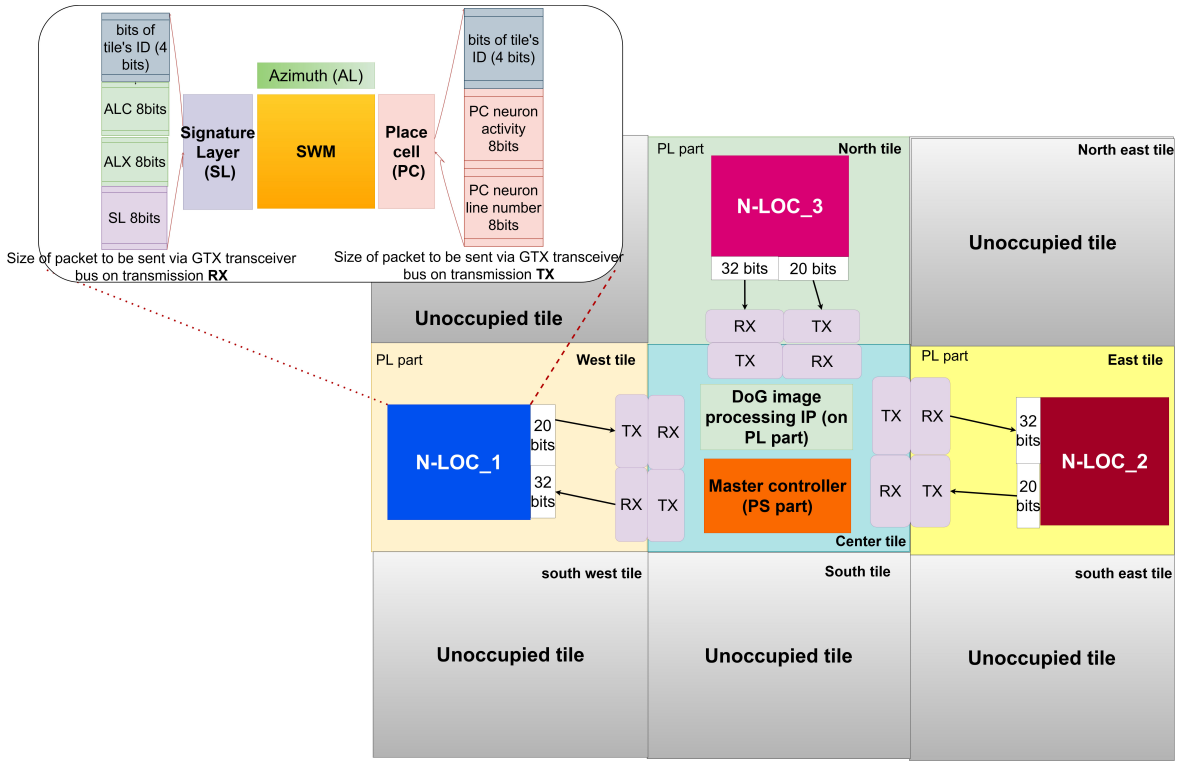


Figure 5.3: **Overall architecture for a design leveraging 3 N-LOC instances.** The design is implemented throughout multi-N-LOCs architecture based on Wizarde (see section 3). TX/RX pairs can be implemented following multiple means: gigabit transceivers, GPIOs, Ethernet, *etc.* An N-LOC IP (detailed in the upper-left corner) awaits an x Azimuth coordinate (*i.e.*, its row number), an x landmark coordinate (*i.e.*, also its row number), the current pixel to process, and its tile ID within Wizarde. Conversely, an N-LOC instance sends the score obtained in the local WTA, its line number in the local Place Cell Memory, and its tile ID. The image acquisition and processing IP is implemented in the central tile, and a lightweight resource manager collects local WTA winners, and performs the final WTA, in parallel with scheduling communications.

.

Table 5.2 and 5.4 summarise all the synthesis results generated and presented within our works, along with some ratio comparisons in terms of latency, throughput, and power consumption.

Table 5.5 provides the performance per Watt of several configurations, one with 100 place cell neurons, and the other with 90 neurons. The performance ratio with power consumption when comparing the reference code with N-LOC instances varies from 4× to 7× (for 30, 60 and 90 place cell neurons according to results showed in Table 4.3). This metric is obtained by computing the following: (1) Each image is partitioned into 16 landmarks, each composed of 12×12 pixels, *i.e.*, there are 2304 pixels to process in each image. (2) During the *Using*

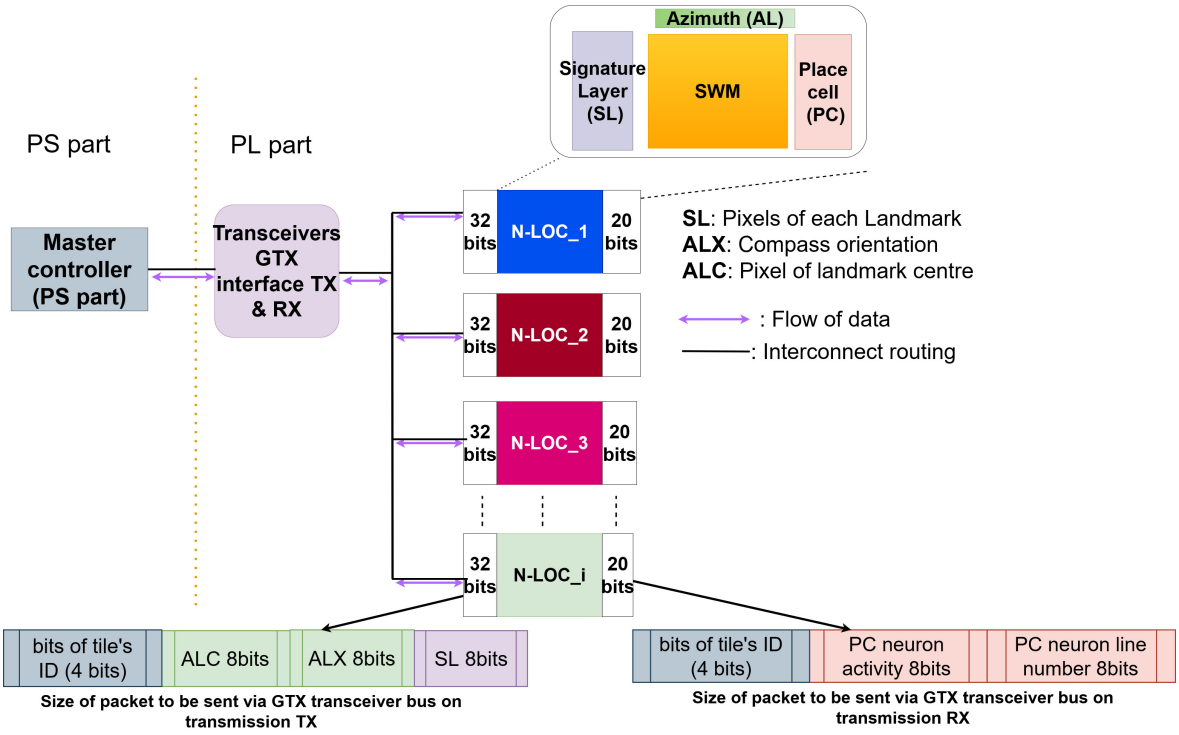


Figure 5.4: Data transmission protocol using GT transceivers implemented throughout a multi-N-LOC architecture based on Wizarde multi-tiles.

phase, there are three distinct types of operations: For the first winner-take-all, each pixel is broadcast to each element of the vector of neurons. To process all pixels for a single neuron, there are $[(12 \times 12) \cdot 16] \cdot 3 = 6912$ operations to perform. Since there are $16 \times 100 = 1600$ neurons in the Signature Layer, the total number of operations to perform in the neurons vector is $6912 \cdot 1600 = 11,059,200$ operations. Once this is done, there is a MAX computation to be performed between all 1600 neurons, *i.e.*, 1601 additional operations; to update the Spatial Working Memory, a single neuron is updated according to all azimuth neurons for the image orientation, resulting in 362 operations for that step; and there are 101 max operations to execute to perform the second winner-take-all. (3) We sum all operations required to perform both WTAs and the SWM update, which yields $\approx 11.1 \cdot 10^6$ operations $= 11.1$ MOPs. To get the performance per Watt, we compute the following: $Perf\_per\_Watt = \frac{N_{\text{ops}}}{Power}$, using the throughput values reported in Tables 4.3 and 5.1, as well as the power consumption of

IP Block	1×90 N-LOC	3×30 N-LOC	3×30 *vs.* 1×90
N-LOC	0.472	0.993	2.10
N-LOC (Learning)	0.182	0.283	1.55
N-LOC (Using)	0.289	0.708	2.44
Processing System	1.629	1.639	1.00
TOTAL	**2.101**	**2.638**	**1.25**

Table 5.4: N-LOC: Power consumption (in Watts) for 90 place cell neurons. The last column computes the power consumption ratio between a 1×90 and a 3×30 configuration. We used Vivado power estimator to evaluate the power consumption of each IP.

1×100 Ref	769 KOPS
1×100 N-LOC	239 MOPS
1×90 Ref	820 KOPS
1×90 N-LOC	269 MOPS
3×30 N-LOC	299 MOPS

Table 5.5: N-LOC: Performance (number of thousands of operations per second).

Table 4.4 and 5.4.

Hence, the first winner-take-all step is overwhelmingly more computationally intensive than the other steps. The total number of operations to process a single image is $\approx 11.1 \cdot 10^6$ operations, or 11.1 MOPS. The reference code runs on NVIDIA Jetson TX2 with a thermal design power (TDP) up to 15W, which we used as the baseline to compute the performance per Watt of various configurations. As the table shows, there is a $4\times$ to $6\times$ ratio in favor of our N-LOC design.

We have demonstrated in previous sections, the possibility of implementing distributed N-LOC IPs over one FPGA tile. The approach has shown a gain in performance in both latency, and power of consumption compared to just one full N-LOC IP, see table 5.1, and table 5.4.

The actual aim is to enhance and increase the capacity of the self-driving vehicles or mobile robots to self localise, thus, it implies expanding our bio-inspired neural architecture, and that requires creating, distributing, and implementing a lot of N-LOC IPs over FPGA tiles, as Wizarde see chapter 2, is our targeted platform. Therefore, to do so, we need at first, well-established communication management and scheduling in this part. The fastest means of communication on wizarde is through the use of gigabit transceivers.

5.5 Communication protocol via GTX transceivers

5.5.1 Highspeed transceivers on the Wizarde platform

As seen in chapter 2, the eventual platform on which to run N-LOC is Wizarde, a 3×3 tile board, with a 2D mesh communication network composed of gigabit transceivers (GTX). Hence, we must define a protocol and a communication scheduling policy to leverage its GTXs. The data transfer policy relies on streaming pixels at each rising edge into our N-LOC blocks (when the data are sent to the SL layer). Then, all the information required by each block is sent to it accordingly. We have evaluated that 32 bits is the maximum packet size required to transfer data for both the RX and TX sides of the GTX interface, for more details see the figure 5.4. Our design is illustrated in the upper-left corner of figure 5.3, including the various required word sizes for TX and RX.

5.5.2 GTX micro-benchmarking in Wizarde

We use Aurora, a LogiCORE IP [3] designed to enable easy implementation of Xilinx transceivers while providing a lightweight user interface on top of which we can build our own protocol.

This IP offers sufficiently low overhead for our needs and will allow us to build our own higher-level protocols in the future while maintaining a high scalability potential.

Specifically, we leverage an 8B/10B encoding, a protocol for high-speed serial data transmission. It provides a good clock recovery on reception and balances the number of zeroes and ones to avoid the presence of a direct current (DC) on the line. It is used in some versions of Ethernet-based network links [3].

Aurora exposes an interface with an AXI4-stream bus, which will allow us to send high-speed data, *e.g.*, *via* its external DDR memory and a DMA, from the processing system part of the Zynq to its programmable logic part.

We implemented tests to validate that tile-to-tile data transfers are indeed correct on the Wizarde platform. We specifically targeted communications between the North and North-West tiles. The benchmarks are carried out at 3.125 Gbps and 6.25 Gbps, as the maximum admissible frequency for the Aurora 8B/10B IP is 6.6 Gbps. The Aurora configuration is shown in table 5.6 for 6.6Gbps.

	North FPGA	North-W FPGA
Lane Width (Bytes)	2	2
Line Rate (Gbps)	6.25	6.25
GT Refclk (Mhz)	125	125
Init clk (Mhz)	50	50
DRP clk (Mhz)	50	50
DRP clk (Mhz)	TX-only simplex	RX-only simplex

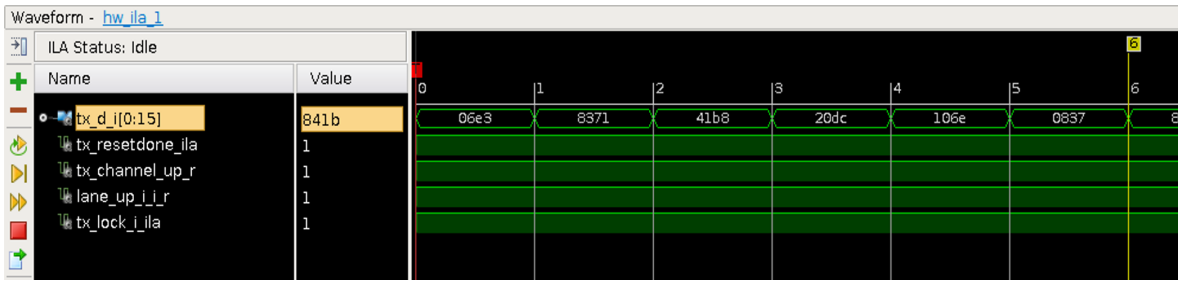
Table 5.6: Aurora IP configuration. The line transmission rate is set to 6,25Gbps, and the GT reference clock is set to 125 Mhz on both TX and RX of adjacent tiles (North, and West-North).

We use the IP in simplex mode (i.e, one-directional data transfers). The North tile will be in the transmit mode while the North-West will be in the receive mode. Our clock reference on Wizarde is set to 125 MHz, to be able to boost the frequency up to 6.25 Gbps. Each tile has a pair of GTX links connected to its nearest neighbours (*e.g.*, the central module has 4 pairs of MGTs to provide a high-speed transmission to each of its immediate neighbours).

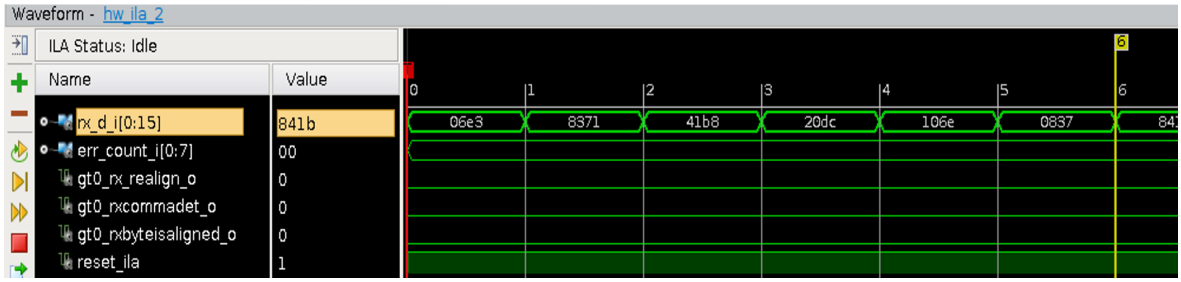
Finally, to carry out the tests we used the example design with the dedicated core IP, which has modules for frame generation (on the TX side) and frame verification (on the RX side). The frames are composed of pseudo-random numbers sent in the AXI4-stream format.

Our tests show the data we send (TX) are identical to the received data (RX), with a delay overhead of ($\approx clk\_cycl/10$). As shown in figure 5.5, we send arbitrary fixed numbers from side to side, and then evaluate the received data (registered in BRAM memory), according to the transmitted ones on the TX register.

To ensure proper activation and reset of the GTXs on both sides of the transmission, we have added some functionalities to send and receive data through DMA, by activating a streaming mode, instead of using framing mode, according to the Xilinx [3]. For more documentation on the chronogram and the VHDL codes used for both sides of TX and RX and for better efficient control, see page 55 [3].



(a) North-tile.



(b) North-West-tile.

Figure 5.5: Waveform data acquisition. We trigger the data acquisition on the chip scope from the pseudo-random value 0x06E3 and we can verify that the reset and activation signals of the GTX are valid and that the received data are in conformity with those sent. We also check the error-accumulator signal remains at 0. This benchmark is set up with a throughput of 6.25Gbps.

Figure 5.6 and 5.7 showcase the resource utilisation for post-implementation, using two different boosting clock frequencies 3.125 Gbps and 6.25 Gbps. The illustrated results are generated and exposed from both sides of tiles on North and West-North.

Site Type	Used	Fixed	Available	Util%
Slice LUTs	2552	0	218600	1.17
LUT as Logic	2315	0	218600	1.06
LUT as Memory	237	0	70400	0.34
LUT as Distributed RAM	24	0		
LUT as Shift Register	213	0		
Slice Registers	3596	0	437200	0.82
Register as Flip Flop	3596	0	437200	0.82
Register as Latch	0	0	437200	0.00
F7 Muxes	142	0	109300	0.13
F8 Muxes	63	0	54650	0.12

(a) North-West-tile (RX-tile), for 3,125 Gbps

Site Type	Used	Fixed	Available	Util%
Slice LUTs	2570	0	218600	1.18
LUT as Logic	2333	0	218600	1.07
LUT as Memory	237	0	70400	0.34
LUT as Distributed RAM	24	0		
LUT as Shift Register	213	0		
Slice Registers	3594	0	437200	0.82
Register as Flip Flop	3594	0	437200	0.82
Register as Latch	0	0	437200	0.00
F7 Muxes	142	0	109300	0.13
F8 Muxes	63	0	54650	0.12

(b) North-West-tile (RX-tile), for 6,25 Gbps

Figure 5.6: Vivado report: Resource usage per tile. The usage rates are almost equal, as such 18 more logic LUTs and less than 2 Flip Flops were recruited for the 6.25Gbps frequency upgrade.

5.5.3 Toward a distributed N-LOC architecture on Wizarde

This section discusses the possibility to implement a distributed N-LOC architecture using a multi-FPGA platform. We will use Wizarde (see chapter 2) as our target. Our reasons to resort to Wizarde are three-fold: (1) Beyond the intrinsic overhead induced by a distributed architecture and its associated control signals, one of the reasons our first attempt at implementing distributed N-LOC does not perform as well as a single N-LOC block is the very

Site Type	Used	Fixed	Available	Util%
Slice LUTs	1820	0	218600	0.83
LUT as Logic	1661	0	218600	0.76
LUT as Memory	159	0	70400	0.23
LUT as Distributed RAM	24	0		
LUT as Shift Register	135	0		
Slice Registers	2525	0	437200	0.58
Register as Flip Flop	2525	0	437200	0.58
Register as Latch	0	0	437200	0.00
F7 Muxes	129	0	109300	0.12
F8 Muxes	63	0	54650	0.12

(a) North-tile (TX-tile).

Site Type	Used	Fixed	Available	Util%
Slice LUTs	1825	0	218600	0.83
LUT as Logic	1666	0	218600	0.76
LUT as Memory	159	0	70400	0.23
LUT as Distributed RAM	24	0		
LUT as Shift Register	135	0		
Slice Registers	2523	0	437200	0.58
Register as Flip Flop	2523	0	437200	0.58
Register as Latch	0	0	437200	0.00
F7 Muxes	129	0	109300	0.12
F8 Muxes	63	0	54650	0.12

(b) North-tile (TX-tile).

Figure 5.7: Vivado report: Resource usage per tile. The usage rates are almost equal, as such the 5 more logic LUT's and the less of 2 Flip Flops less were recruited for the 6.25Gbps frequency upgrade.

small size of each neural network, which can be alleviated if a sizeable portion of each tile involved in the design can be leveraged (as one tile is roughly able to store 4600 neurons with 100 place cell neurons) (2) the GTX links, coupled with the AXI Stream protocol should offer a more asynchronous way of transferring data between the controller (still implemented in software) and its neighbouring tiles, which should reduce communication overheads (for both throughput and latency); and (3) this is the only way we can eventually implement a large neural network – large enough to be useful in a self-driving car. Moreover, such a network could be grown dynamically and on-demand, according to the computational needs of the current context in which the car is situated.

We target four tiles in the Wizarde board: for instance, the central tile, as well as the North, East, and West tiles. The latter tiles implement an instance of the N-LOC IP, combined with a GTX interface (see figure 5.3). The central tile implements the image processing IP (see chapter 2), and orchestrates communications across all tiles *via* the GTX interface. The communication scheduler is implemented in software on the central tile, using the ARM Cortex A9 processor.

The N-LOC data exchange of buffer size is detailed such the compass value (image orientation, *i.e.*, azimuth values) is sent once for each image to process, also the Azimuth values are computed locally in each N-LOC block. For each landmark (12×12 pixels), the x coordinate of the keypoint is sent to the N-LOC block, besides, for each pixel, the value of the most active neuron in SL (and its x coordinate, *i.e.*, its "line number") is sent to the relevant N-LOC block. Thus, Once all 16 vignettes have been processed, the value of the most active neuron in the PC layer of each N-LOC is sent back to the controller (PS), for more details and illustration see figure 5.3 and figure 5.1.

As a result, each N-LOC block must receive a new compass value every 16 vignettes. The word size for the azimuth buffer takes 8 bits for each period of $16 \cdot 144$ cycles of ref_clk cycle. In addition, it must receive a new x coordinate value for each new vignette The word size of the Azimuth landmark's x coordinate also takes 8 bits for every 144 periods of ref_clk cycle ($144 \cdot$ ref_clk period). It must then send its most active place cell value every time a full image has been processed. The word size of place cell (block's output) takes 8 bits for each ($16 \cdot 144 + cst$) ref_clk cycle). cst is a constant which varies with each target system.

5.5.4 Conclusion

Finally, we have seen the necessity of expanding the bio-inspired neural architecture as it is linked directly to the capability and the performance of an AV on self-localisation, or

even in environment exploration. N-LOC is a type of VPR and in particular that hardware implementation of LPMP model [43], on an FPGA board. In chapter 4, we have presented some different results of the implementation of the N-LOC on FPGA, such as the latency, and power consumption, comparing it to the Python-based reference implementation. Thus, promising results from this implementation were generated. Furthermore, in this chapter, a need for distributed N-LOC architecture is highlighted, with the feasibility of distributing the IPS over one FPGA board, we have also demonstrated and presented the need for multi-FPGAs to do so, as Wizarde will be our targeted platform. We then concluded by showing that in order to deploy the N-LOC IPS over the FPGA tiles, we need to establish and leverage GTX gigabit transceivers as it is the protocol communication that all tiles of FPGA incorporate. However, there will be respect for both N-LOC data-path between N-LOC IPs, which is already demonstrated and proposed in this section. As well as, we need to establish and propose how the data shall be transferred and communicated between all tiles of FPGA, we thus talk about this aspect in the next section.

5.6 Implementing distributed N-LOC using Dynamic Partial Reconfiguration DPR system

The distribution and implementation of the N-LOC onto one or multiple reconfigurable fabrics in an efficient manner are what every researcher or engineer tries to seek. Until now, we have proposed a new scenario with a static aspect, which means that every N-LOC IP in such an application can't be changed or modified in real-time. Thus, leveraging DPR techniques as a tool in creating and designing the dynamic accelerators and static part of the design is required for that purpose. Even though, we have demonstrated that the maximum of N-LOC instances we create and deploy over FPGAs reconfigurable fabrics, the more efficient the localisation will be in different exploring environments. In the next sections, we present some conducted experiments, we have done using both Ker-ONE a software programmer-friendly, hypervisor with a very small footprint (for more details see chapter 3), and FOS a hardware programmer friendly, a Linux + kernel modules developed using HLS (for more details see chapter 3). Thus, this section described the various experiments we ran to test both systems.

In conclusion, Ker-ONE requires a deep technical understanding of its architecture (both the user part which is made for software development, as well as for the kernel part which is basically made upon both hardware control registers and the Linux file system), in addition to, back to the year when we have started by exploring and testing that hypervisor, a lot of updates and modifications were in time, thus, that what we led us to find out other alternative solutions in the state of art.

5.6.1 Experiments with Ker-ONE

Ker-ONE relies its powerful control on its hypervisor, to leverage and use the partial reconfiguration that Xilinx architectures provide in their architecture, for more details, see the chapter 3. We have conducted some experiments in which we present the generated overheads caused by the switch between different accelerators (filters based applications), on the same relocated region. Figure 5.8 presents the architecture we have proposed to test and evaluate ker-ONE the hypervisor, with a different type of partial regions that we have created to switch and change between the hardware accelerators. We manage to create all the

bitstreams and BIN files corresponding to each IP, also, by using the AXI-Timer to measure the elapsed time of each IP, separately. Thus, through the communication between DMA and partial regions, we must specify that the choice of size of the reconfigurable blocks and the memory address is very important for control registers that Ker-ONE has in its architecture, see figure 2.4 in section 2 for more details. Moreover, we present in the figure 5.7 the elapsed time to upload and switch the partial accelerators based on bitstream files, to the targeted platform Zynq XC7Z045.

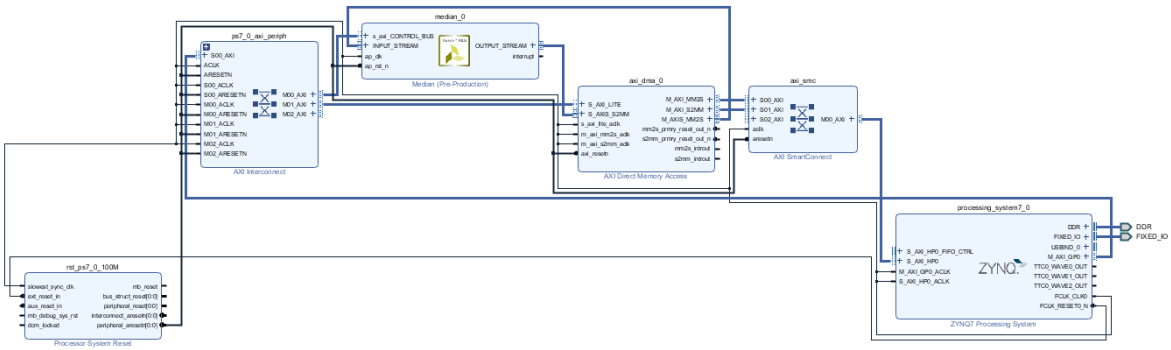


Figure 5.8: Vivado design based on Vivado tool 2016.4. As the partial region contains rather the median or sobel filter, and the Arm Cortex A9 on which the Ker-ONE hypervisor will be ported. Zynq7000 SoC FPGA is the type of board used for the evaluation.

Type of filter	Nombre de LUT necessaire	Execution time (ms)	resource consumption (LUT)	time of reconfiguration (ms)
Mediane filter	9	467	44830	1.172
Sobel filter	30	2826	44835	1.129

Table 5.7: Nombres de cellules logiques nécessaires et temps de la reconfiguration des blocs reconfigurables.

5.6.2 Experiments with FOS

Byteman enables a user to build a bitstream for the larger device using Vivado and then relocate it as if were the smaller FPGA and possibly bypass the software limitations, leveraging FOS as the scheduler system part. For more details about Byteman and FOS, see the chapter 3.

The capabilities of partial reconfiguration (PR) are the desired features to be extracted from the reconfigurable fabrics, and to be fully explored for dynamic execution pipelines at runtime [91]. Besides, it's accepted that it is most complicated to materialise PR at scale, and FPGAs are only used as updatable ASICs [91]. Manevet *et al.* in their work [91] proposes a resourceful FPGA bitstream manipulation framework called Byteman. The proposed tool provides means for parsing, modification, and generation of bitstream files, and it has been open-sourced and demonstrated in a working system. As a distinguished feature, it supports multi-die FPGAs (among the 106 Xilinx 7 Series, UltraScale, and UltraScale+ devices), and enables datacenter FPGAs to be used for relocatable PR. Bundled with an efficient bitstream manipulation core,

the efficiency is demonstrated by two case studies where the obtained performance giving by 58-377x higher bitstream merging throughput than a current state-of-art tool.

In 2021, we had a chance to get a grant for a HiPEAC mobility, through which we aimed to develop and rise a new promising collaboration between what we are doing in our laboratory at ETIS, with a new laboratory called APT Advanced Processor Technologies Research Group of Computer Science at the University of Manchester, United Kingdom. This is the first time the University of Manchester and the ETIS laboratory are collaborating. The research interests of both teams complement, and this will be an excellent way to foster further collaboration in the future. The core of this collaboration is to integrate the Byteman partial configuration tools and FOS (the FPGA operating system) from Manchester into a framework around the Wizarde heterogeneous multi-FPGA platform for autonomous vehicle navigation, which is developed by ETIS.

We have proposed to explore scheduling policies and inter-FPGA task placement (and potential migration) by leveraging FOS, in the context of multi-grain task scheduling, i.e., where hardware tasks may vary in size and hardware resource requirements. This will be done through the following steps:

- Run the *vanilla* FOS on UltraScale and look at how to add a new scheduling policy.
- Add transceiver IPs and a software layer for multiple instances of FOS to communicate.
- Work on scheduling policies to take into account inter-FPGA task and data migration

First of all, We aimed to use Byteman an open-source high-performance bitstream relocation and manipulation tool, to demonstrate the capability of implementing and switching the accelerator modules through their partial regions. See figure 5.9 for more details about the proposed hardware design. The workflow is as follows:

- Develop and create the static and partial bitstreams using the Vivado tool.
- Feed the partial and static bitstreams to Byteman to stitch between the pre-generated static design and reconfigurable modules. Byteman tool helps us to relocate, search, and map for appropriate resources for our bitstreams.

Furthermore, by selecting the N-LOC instance in the Netlist pane, we then draw a tall narrow box on the clock region. The exact size and shape do not matter at this point, but we keep the box within the clock region. The General tab of the Pblock Properties pane can be used to add these if needed. The Statistics tab shows the resource requirements of the currently loaded reconfigurable Module. See figure 5.10 in which we present the floorplanning, the reconfigurable region as we use one region for DPR.

The relocation and the partial regions management handled by Byteman was not completed as it has been meant to be, for the provided accelerators instances at that time, when the distribution of N-LOC through FPGA fabric (see figure 5.9) was ongoing of development. Thus, exploring and leveraging the scheduling policy that FOS uses, was a big deal for us. We aim further to re-implement the same execution pipeline to create the final relocatable partial regions along with the static part (which contains the GTX transceivers and image processing DoG IPS) on Vivado IDE and hardware accelerators (distributed N-LOCs). Promising results will be the goal of the publication.

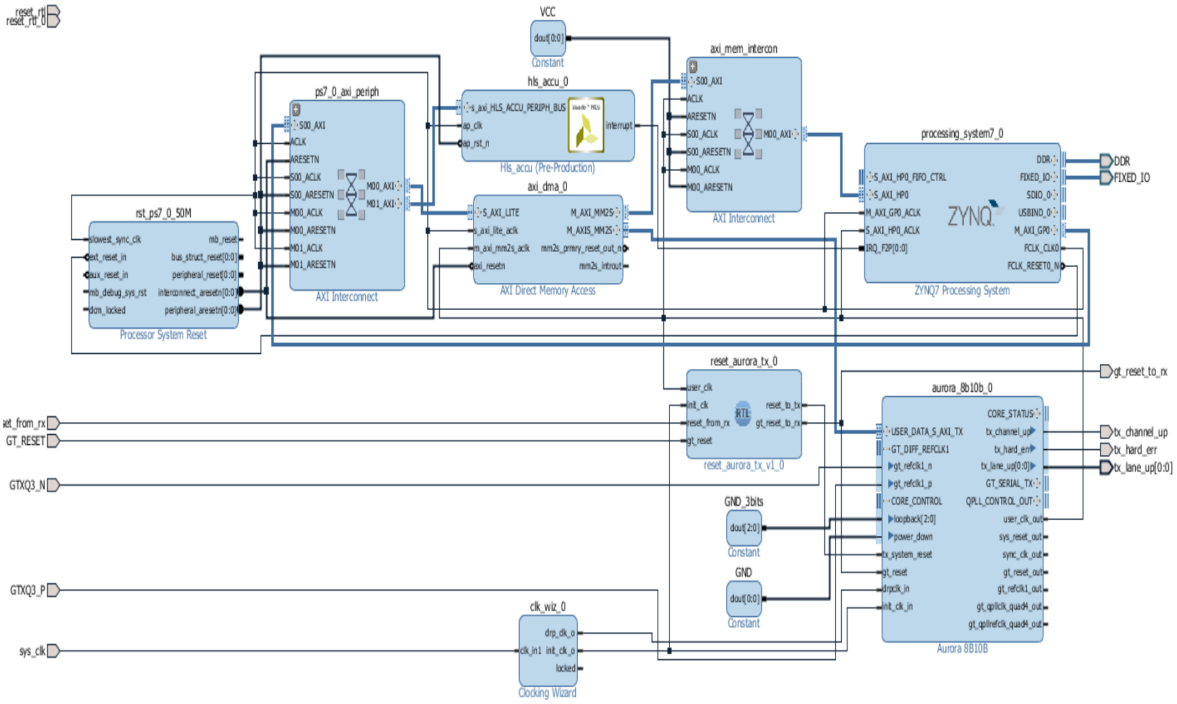


Figure 5.9: **Overview of Vivado design**. This figure illustrates the different required IPs components to communicate between two different tiles, for more details see section 5, in specific, see section 5.5. The main goal of this design is to leverage Byteman relocatable and manipulation tool.

5.6.3 Discussion and comparison: FOS *vs.* Ker-ONE

FOS incorporates Byteman which is a partial relocation tool. FOS uses Round-Robin as a type of scheduling algorithm to place the partial accelerator. The Byteman tool shows its capability to place, and relocate the accelerators on very fined and appropriate hardware resources, as well as the partial regions (created by VHDL or HLS) with more flexibility and efficiency than what exists in the state of the art. Manev *et al.* in their experimental results [91], observed that 58×-377× higher bitstream merging throughput than current state-of-art tools can yield. Unlike Ker-ONE, which is a hypervisor and considered middleware, also its hardware side lacks hardware control registers (according to our Ker-ONE hands-on experimentation in 2019), and it follows a high complexity (according to our evaluation of Ker-ONE) rather than what was presented by Xia *et al.* in [146]. In addition, FOS demonstrates its flexibility, and high efficiency when it comes to complex accelerators that can leverage the high throughput of a system or platform.

In conclusion, FOS proposes a new custom hardware tool-chain (Byteman) to perform the partial reconfiguration placement as efficiently as possible, along with, using a soft middleware to incorporate pre-existed scheduling algorithms. However, Ker-ONE uses a vivado hardware toolchain to carry out and create the partial reconfigurable regions, and in contrast to FOS policy, Ker-ONE invests in the software part by proposing user and kernel modes, in which it incorporates specific software registers, drivers, etc.

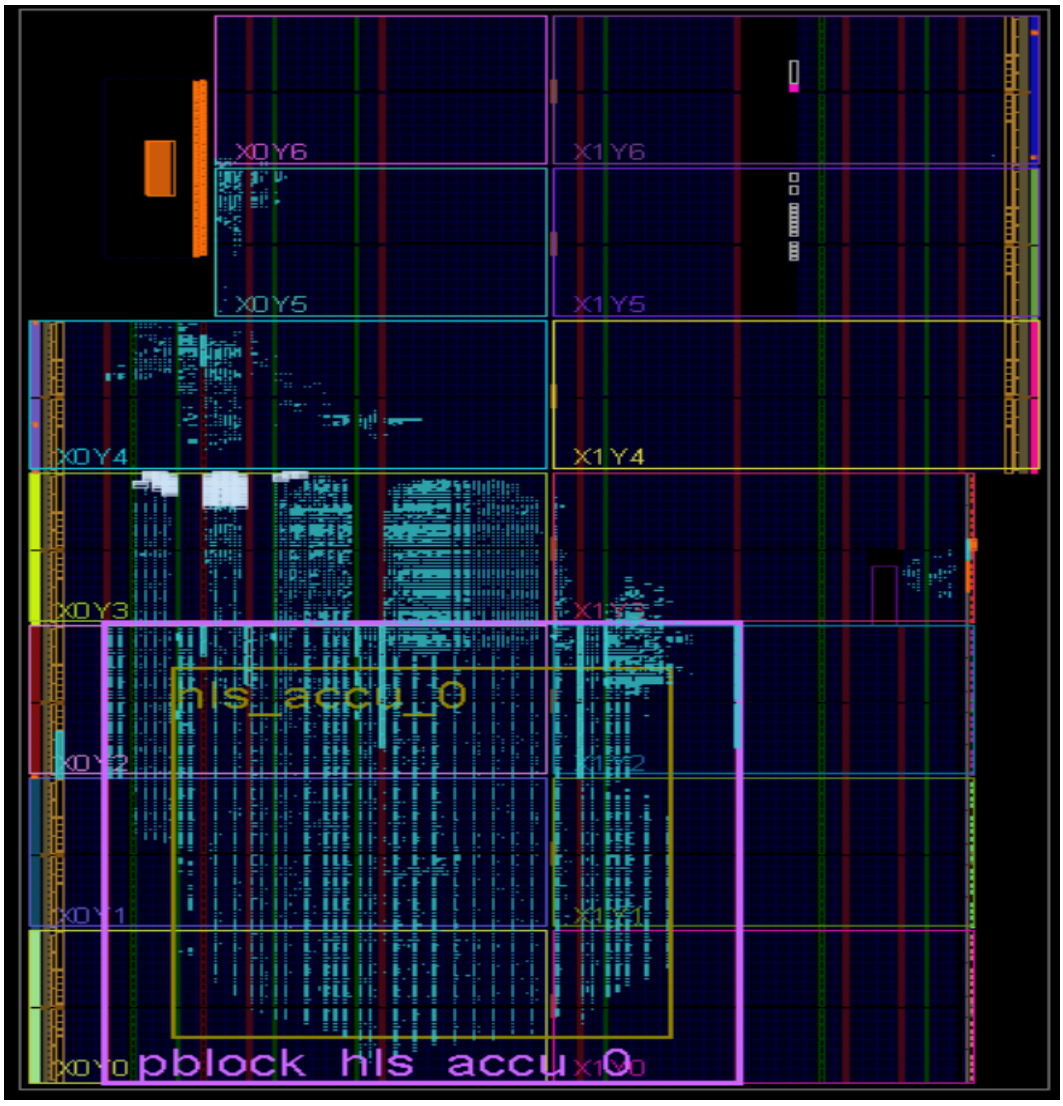


Figure 5.10: The placement of the hardware resources is indicated by blue, as the ARM cortex part is in orange. The rectangle in yellow represents the needed surface of hardware resources to relocate the hardware implementation N-LOC based on bio-inspired architecture, for more details, see chapter 4. As byteman will be responsible for hardware resources implementation by providing the needed LUTs, BRAMs, DSPs, and FF registers, also it stitches between static part and dynamic relocatable region with high efficiency according to [91].

5.7 Wizarde data-path communication and protocols for N-LOC's neural network deployment

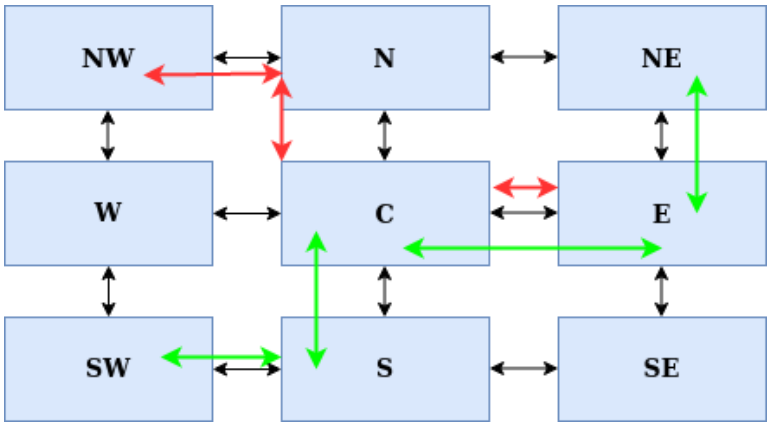
5.7.1 Different tasks communication scenarios on Wizarde's tiles

In this chapter, The possibility of implementing two different scenarios for data-path and tasks communication through **Wizarde** platform will be discussed. Many previous works have proposed different methodologies to measure the bandwidth and the latency between multiple FPGAs [153, 65, 108].

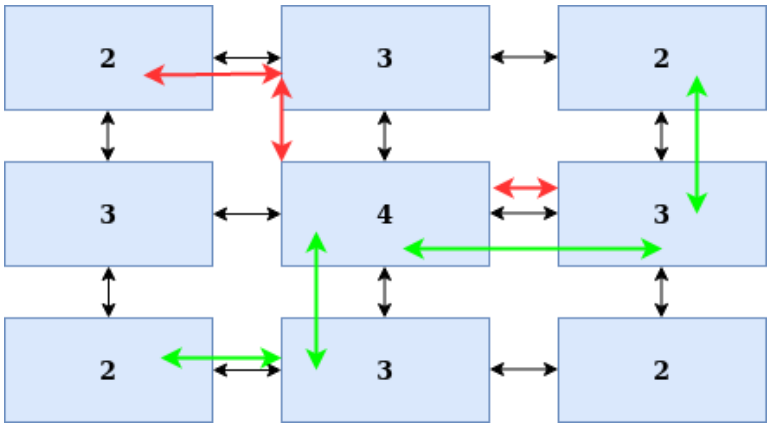
As seen in chapter 4, the necessity of duplicating the N-LOC IP for the localisation task is essential to increase and grant the mobile robot or AV the efficient way of how it can explore the environment in resilience, and run-time. Therefore, we need also to focus and show through which type of communication the data needs to be moved along.

We thus present two essential scenarios that we aim to incorporate and deal with in future, after modelling and partitioning our bio-inspired neural application N-LOC [51].

- Communication between two **adjacent** tiles of Wizarde:
  - Using GTX Transceivers will help us reach high-performance communications and carefully-designed communication optimisation strategies.
- Communication between two **Non-adjacent** tiles of Wizarde
  - The first scenario to be proposed, we suppose that all traffic and data communication control have to pass through the central tile, see figure 5.11.



(a) Central tile as traffic roundabout for the data-path.



(b) Number of transceivers for each Wizarde tile.

Figure 5.11: Waveform data acquisition. We trigger the data acquisition on the chip scope from the pseudo-random value 0x06E3 and we can verify that the reset and activation signals of the GTX are valid and that the received data are in conformity with those sent. We also check the error-accumulator signal remains at 0. This benchmark is set up with a throughput of 6.25Gbps.

The throughput and data path could create a bottleneck in the central tile. Nonetheless, we should not consider it as the router of our communication data. Likewise, to avoid the problem caused by the non-adjacent tiles scenario, we set some policies and premises:

- The control signals for data communications belong to the scheduler.
- Every FPGA tile has its identification number, to get the communication well established.
- If the data path is vertically or horizontally then the inter-FPGAs communication will be set linearly; otherwise, the scheduler will prioritise the central tile through which the data will be passed into. When the lanes of transceivers are IDLE or available, else, it chooses another available path of a specified tile to cross over.

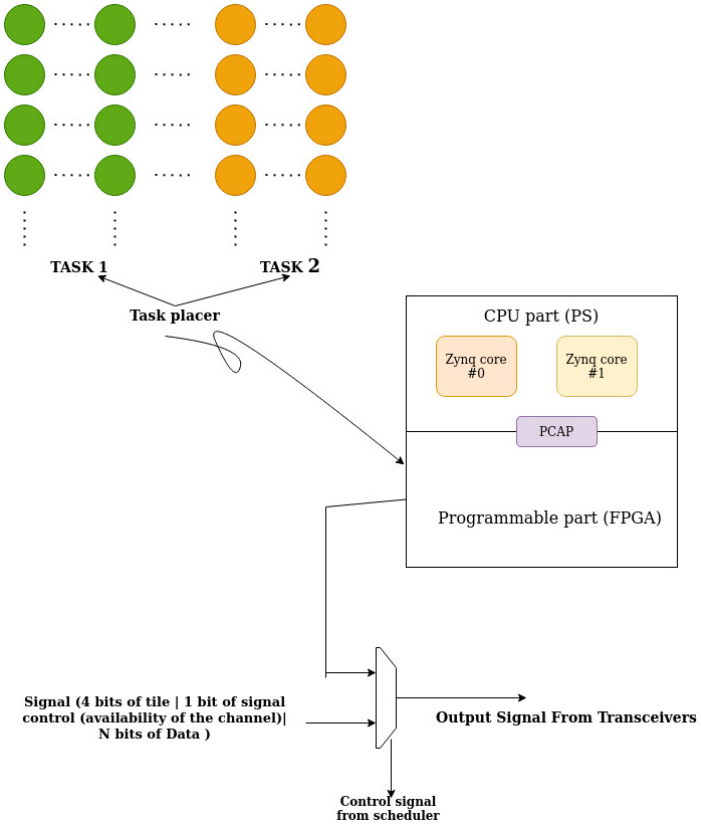


Figure 5.12: Placement and protocol of switch control on PL Part.

We are currently trying to analyse all possible cases that could happen during the communication process and put forward our first algorithm for this data path and inter-FPGAs management communication, based on several previous works and according to Wizarde's schematic design. A short latency is required to minimise physical distancing between communicating tiles, transmission time, and power consumption. Thus, we present in the Algorithm 2 the transfer-data-processing as explained in the previous conditions.

5.7.2 Tasks communication policies-formulas on Wizarde platform

Some previous works in the field of massively-parallel FPGA-based platforms have been proposed by Mencer *et al.* in [96]. The aim of creating this cubic FPGA platform in this work

is to outperform the estate-locality challenge, latency, and throughput (intra and/or inter-FPGA communication), and further to give the best performance ratio for the application systems. In our case study, we are going to highlight the experimental studies in such levels, so our study will focus on one board of Wizarde.

The proposed scenarios will be presented, with the aim of establishing task communication over tiles for the bio-inspired neural application.

To ease the process of implementation, we have to presuppose some assumptions that would help us to reduce the real-time constraints. Therefore, the scheduler must ensure that task allocation has to occur near each other, thus, affording a fast data transfer between FPGAs.

For the case of having just one wizard board and if we consider the non-adjacent data path communication scenario which will be established between tiles (SW & NE). Thus, the transfer time between those FPGA tiles and the data path are calculated based on the equations 5.1 and 5.2. The d_{min} is the minimum required time we can have for such a data transfer, as well as the d_{max} is the maximum time we can get, such as T_t2t refers to the measured time between two adjacent tiles, and T_t2t refers to the waiting time during the routing communication inside an FPGA tile.

$$d_{min} = 4 * T_{t2t} + T_{wr} \tag{5.1}$$

$$d_{real} = 6 * T_{t2t} \tag{5.2}$$

However, we should take into consideration that a priority in the data transmission must be pre-established, hence, to make sure that the channel is well allocated for data-path transmission, otherwise, an overlapping issue would occur in the traffic. Moreover, some registers are proposed to retrieve and store the streaming data (on-chip memory), besides, a FIFO pipeline is needed during each process of memory fetching, in order to manage the consumed and loading data.

As mentioned, the waiting time is required to minimise a critical path, transmission time, and power consumption. We present in the algorithm 2 the transfer data processing as explained in the previous conditions:

Algorithm 2 Algorithm for Tasks communication inter-tiles

Result: Algorithm for Tasks inter-tiles communication
Placement of Tasks & sub-tasks over Wizarde;
Associate an ID number into each tile of Wizarde;

while *Placement is successfully done* **do**
  instructions **if** *non-adjacent sub-groups transfer demand* **then**
    the scheduler chooses a private locked path-transfer
    Measuring the transfer time (overhead)
    free the line-path transfer
  else
    chose a linear line for transfer
    calculate the transfer time correlated with waiting for & overhead
  end
end

5.7.3 Bio-inspired neural network tasks placement using DPR

In order to grant our localisation system better capabilities in terms of accuracy-based wide-range navigation, we need to expand the bio-inspired neural network architecture implemented as hardware-accelerator-based N-LOC. However, the resource consumption and memory footprint of that purpose is very costly. As we have seen in table 4.2, the percentage of resources is limited over ≈180 neurons Place-Cell for each FPGA tile. Therefore, leveraging dynamic partial reconfiguration (DPR) is essential to implement scalable neural networks. As implemented, the localisation task is large enough that it will not completely fit into the available reconfigurable fabric. Moreover, the computational needs may change according to the vehicle's environment, *e.g.*, transitioning from a dense urban area to a rural one, with a possible shift in available light. Thus, the neurons used to decide, will not be the same and will yield different weights. As a result, relying on a full hardware solution is not reasonable or realistic.

Instead, the system should rely on a light software layer which will provide a scheduling and resource management environment, to decide which and where hardware tasks to allocate, within an FPGA. Hence, a major step must be achieved, by providing a layer to provide an API to load and replace hardware tasks.

As a future work, post-scheduling algorithms' capabilities (Fixed priority with Round-Robin, along with, priority-driven scheduling in real-time) must be tested to figure out the best task allocation strategy to achieve real-time navigation using a bio-inspired approach. Thus, using a CPU scheduling system, FPGA accelerators can be managed much more efficiently with more complex strategies, which inevitably optimises and outperforms the acceleration.

5.8 Conclusion

We proposed a low-footprint and high-performance accelerator for feature and image recognition in the context of autonomous vehicle navigation. It leverages a bio-inspired algorithm which extracts pixel values to perform feature extraction and image recognition, by porting a previously software-based process and designing a hardware-based one, which relies on the difference of Gaussian operations. Compared to previous (highly accurate) implementations, ours provides not only accuracy, but also very low latency (9× shorter than the Python reference implementation), low power consumption (5.5× lower), and high frame-rate (7× higher). In addition, our experimental results show that the proposed accelerator yields a much lower power consumption footprint (0.257W for the LPMP implementation; 2.741W for the whole system) compared to the pure software reference implementation running on a high-end embedded system.

However, increasing the number of neurons in Place cells can be beneficial to have a long-term distance vehicle navigation capability with the same accuracy as architecture with fewer neurons. Otherwise, we can have a higher accuracy of the system, with short-distance navigation that we can have with an architecture that contains fewer neurons, according to [43]. Thus, the bigger the bio-inspired neural architecture size will be, the more precise and able to explore and navigate the vehicle will be, as it depends on the number of images that we have learned, and their pixels are stored on the weights of the Signature Layer. In addition, we have shown that over 180 neurons in the Place cell (which corresponds to the number of images we can process and learn in our architecture), can easily saturate FPGA resources.

Therefore, the need to resize the whole application is a must for us. We then consider doing that, by distributing multiple N-LOCs over multiple FPGA tiles.

We demonstrated Wizarde's multi-FPGA capability to implement the whole neural network (≈1000 neurons) over multi-tiles, by leveraging Wizarde's gigabit transceivers. The software processing part will be deployed on the FPGA centre tile to communicate and control all FPGA tiles, by receiving and assessing the localisation score of each captured image from different NLOC accelerator modules.

Future work includes implementing LPMP on the Jetson TX2's GPU, as well as modifying our architecture to increase its clock frequency to improve its performance, and compare it to GPU-based embedded systems in terms of performance and power consumption. We also aim to embed our bio-inspired neural IP into a mobile robot to test its limits and perform a runtime assessment of the implemented navigation approach. Furthermore, a dynamic scheduling scenarios based on pre-existed software platform will be proposed to efficiently deploy the whole application by delivering a high performance run-time circuit.

Part IV

General conclusion and Perspectives

To implement bio-inspired algorithms and their underlying neural network on heterogeneous systems relying on reconfigurable fabric, a novel hardware platform called Wizarde was proposed and used to prototype and deploy the NN architecture onto it. besides, system software and scheduling strategies will need to be designed.

In chapter 4, we have demonstrated the capability of modelling the bio-inspired neural architecture on FPGA. The architecture has been proposed by [43]. The authors Colomer *et al.* have shown the limits of the vehicle speed in all different environments, by implementing and testing via different scenarios at the VEDECOM institute. We have shown that our custom original hardware implementation of the LPMP model, outperforms the reference application by yielding up to 9× lower latency times, 7× higher throughput (frames/second) than the reference software implementation, and power footprint as low as 2.741W for the whole system, *i.e.*, up to 5.5× less than a regular high-end computer system on average.

However, the need to yield more powerful results and well efficient run-time system is a necessity for the autonomous vehicle to be able to auto-drive in any kind of environment. Therefore, according to [44], the more we have neurons in the Place cell to represent and memorise images during the learning phase, the more that the vehicle can make a trustful result and explore widely. thus, the need of scaling the application is required in this situation.

In chapter 5, we have started by displaying the capability of duplicating the targeted hardware application through multiple accelerators or IPs, such as each different IP has the opportunity and permission to learn specific images (refereed as Learning mode), and all of them will be evaluated during the navigation process (refereed as using mode), by feeding same images that have been learned before. According to our results, the distribution through multiple IPs on the same FPGA can provide a speedup up to 2× compared to a monolithic N-LOC system, likewise, it keeps yielding up to $4-6\times$ faster than the original software implementation, however, compared to the individually measured Learning and Using latency, this performance is rather low. It is important to note that in the reference code, as the number of place cell neurons increases, the processing time also increases dramatically.

In the same chapter, we have conducted some real-time experimentations through which the gigabit transceivers are shown to be our best solution to communicate between wizarde's tiles, with a frame rate which can't overpass 6 GB/s, in order, to decrease the overhead time communication between FPGAs.

As a future work, and after porting the artificial hippocampus's NN to the Wizarde platform to test its ability to reconfigure its various partial reconfiguration regions, we will leverage the Dynamic partial reconfiguration DPR to take care of minimizing task relocation when possible. Once this is done, the next steps will include designing simple IPs to start to evaluate efficient multi-FPGA scheduling strategies with real-time deadlines.

Besides, leveraging a middleware or hypervisor [141, 146], with an online scheduling policy would help us to facilitate and achieve our goal, in addition, We plan on running several scheduler instances (one per tile) to efficiently perform DPR locally, but also to allow hardware and software task migration across tiles when necessary, and one all of that is done, we change scheduling policy to make it convenient for our objectives.

We start by scheduling hardware tasks into one tile's FPGA then Measure latency & throughput to compare it with pure CPU implementations, then we get across multiple FPGAs / tiles. and once all of that is done, we fetch the neural network which fits into one tile's FPGA

after we map a larger neuron which fits onto multiple tiles' FPGA. further, to overcome spatiality and estate issues, we associate reconfigurable regions to each available FPGA such as each tile's FPGA being configured in the same way.

6

Bibliography

[1] The 6 levels of vehicle autonomy explained. https://www.synopsys.com/automotive/autonomous-driving-levels.html. Accessed: 2023-07-06. 12

[2] 7 series fpgas transceivers wizard v3.6. https://www.xilinx.com/content/dam/xilinx/support/documentation/ip_documentation/gtwizard/v3_6/pg168-gtwizard.pdf. Accessed: November 30, 2016. 39

[3] Aurora 8b/10b v11.0, logicore ip product guide. https://docs.xilinx.com/v/u/11.0-English/pg046-aurora-8b10b. Accessed: 2016. 74, 75

[4] Blackfly s usb3. https://www.flir.fr/products/blackfly-s-usb3/?vertical=machine+vision&segment=iis. Accessed: 2023-07-06. 52

[5] A comprehensive guide to the backpropagation algorithm in neural networks. https://neptune.ai/blog/backpropagation-algorithm-in-neural-networks-guide. Accessed: 2023-07-06. 16

[6] Edge computing acts on data at the source. https://www.ibm.com/cloud/what-is-edge-computing. Accessed: 2023-07-06. 24

[7] Intel realsense (2021, april). https://www.intelrealsense.com/. Accessed: 2023-07-06. ix, 26

[8] Intelrs-bpearl (2021, april).. https://www.robosense.ai/en/rslidar/RS-Bpearl. Accessed: 2023-07-06. ix, 25, 26

[9] An introduction to vhdl. http://www.uco.es/~ff1mumuj/h_intro.html. Accessed: 2023-07-06. 31

[10] Log-polar coordinates. https://en.wikipedia.org/wiki/Log-polar_coordinates. Accessed: 2023-07-06. 42

[11] Multi-layer perceptron. https://scikit-learn.org/stable/modules/neural_networks_supervised.html. Accessed: 2023-07-06. 16

[12] posegraph, create 2-d pose graph. https://en.wikipedia.org/wiki/Sparse_dictionary_learning. Accessed: 2023-07-06. 11

[13] Residential street definition. https://www.lawinsider.com/dictionary/residential-street. Accessed: 2023-07-06. 7

[14] Sparse dictionary learning. https://www.mathworks.com/help/nav/ref/posegraph.html. Accessed: 2023-07-06. 48

[15] Tesla's new chip promises to turbocharge autonomous driving. https://dug.com/teslas-new-chip-promises-to-turbocharge-autonomous-driving/. Accessed: 2023-07-06. 26, 30

[16] What is gnss? https://www.euspa.europa.eu/european-space/eu-space-programme/what-gnss. Accessed: 2023-07-06. 18

[17] What is vhdl? getting started with hardware description language for digital circuit design. https://www.allaboutcircuits.com/technical-articles/hardware-description-langauge-getting-started-vhdl-digital-circuit-design/. Accessed: 2023-07-06. 31

[18] *Handbook of Intelligent Vehicles*. Springer London, London, 2012. 18

[19] Alveo u55c, 2015. Accessed: 2021. 32

[20] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2015. 32

[21] Kamel Abdelouahab, Maxime Pelcat, Jocelyn Sérot, and François Berry. Accelerating cnn inference on fpgas: A survey. *ArXiv*, abs/1806.01683, 2018. 17, 65

[22] Karim M. A. Ali, Rabie Ben Atitallah, Nizar Fakhfakh, and Jean-Luc Dekeyser. Exploring hls optimizations for efficient stereo matching hardware implementation. In Stephan Wong, Antonio Carlos Beck, Koen Bertels, and Luigi Carro, editors, *Applied Reconfigurable Computing*, pages 168–176, Cham, 2017. Springer International Publishing. 53

[23] Alberto Aloisio, Francesco Cevenini, Raffaele Giordano, and Vincenzo Izzo. Characterizing jitter performance of multi gigabit fpga-embedded serial transceivers. In *2009 16th IEEE-NPSS Real Time Conference*, pages 96–101, 2009. 39

[24] Elissa M. Aminoff, Kestutis Kveraga, and Moshe Bar. The role of the parahippocampal cortex in cognition. *Trends in Cognitive Sciences*, 17(8):379–390, Aug 2013.

[25] Relja Arandjelović, Petr Gronat, Akihiko Torii, Tomas Pajdla, and Josef Sivic. Netvlad: Cnn architecture for weakly supervised place recognition. *arXiv:1511.07247 [cs]*, May 2016. arXiv: 1511.07247. 17

[26] Arslan Arif, Felipe Barrigon, Francesco Gregoretti, Javed Iqbal, Luciano Lavagno, Mihai Teodor Lazarescu, Liang Ma, and Manuel Palomino. Performance and energy-efficient implementation of a smart city application on fpgas. j real-time image proc 17, 729–743 (2020). In *Journal of Real-Time Image Processing 17, 729–743 (2020)*, 2020. 65

[27] I Augé and F Pétrot. High-level synthesis: From algorithm to digital circuit, 2008. 32

[28] Saeed Safari Behrooz Abdoli. A reconfigurable real-time neuromorphic hardware for spiking winner-take-all network. In *Int J Circ Theor Appl. 2020; 48: 2141– 2152*, 2020. 65

[29] Abdelmoudjib Benterki, Moussa Boukhnifer, Vincent Judalet, and Choubeila Maaoui. Artificial intelligence for vehicle behavior anticipation: Hybrid approach based on maneuver classification and trajectory prediction. *IEEE Access*, 8:56992–57002, 2020. 13

[30] Mario Bergeron, Steve Elzinga, Gabor Szedo, Greg Jewett, and Tom Hill. 1080p60 camera image processing reference design. In *XAPP794 (v1.3) December 20, 2013*, 2013. 50

[31] Massimo Bertozzi, Luca Bombini, Alberto Broggi, Michele Buzzoni, Elena Cardarelli, Stefano Cattani, Pietro Cerri, Alessandro Coati, Stefano Debattisti, Andrea Falzoni, Rean Isabella Fedriga, Mirko Felisa, Luca Gatti, Alessandro Giacomazzo, Paolo Grisleri, Maria Chiara Laghi, Luca Mazzei, Paolo Medici, Matteo Panciroli, Pier Paolo Porta, Paolo Zani, and Pietro Versari. Viac: An out of ordinary experiment. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, page 175–180, Baden-Baden, Germany, Jun 2011. IEEE. 18

[32] Michael Beyeler, Nicolas Oros, Nikil Dutt, and Jeffrey L Krichmar. A gpu-accelerated cortical neural network model for visually guided robot navigation. *Neural Networks*, 72:75–87, 2015. 25

[33] Keshav Bimbraw. Autonomous cars: Past, present and future - a review of the developments in the last century, the present scenario and the expected future of autonomous vehicle technology:. In *Proceedings of the 12th International Conference on Informatics in Control, Automation and Robotics*, page 191–198, Colmar, Alsace, France, 2015. SCITEPRESS - Science and and Technology Publications. 18

[34] Michaela Blott, Nicholas J. Fraser, Giulio Gambardella, Lisa Halder, Johannes Kath, zachary Neveu, Yaman Umuroglu, Alina Vasilciuc, Miriam Leeser, and Linda Doyle. Evaluation of optimized cnns on heterogeneous accelerators using a novel benchmarking approach. *IEEE Transactions on Computers*, 70(10):1654–1669, 2021. 30, 32, 37, 65

[35] Federico Boniardi, Tim Caselitz, Rainer Kümmerle, and Wolfram Burgard. Robust lidar-based localization in architectural floor plans. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3318–3324, 2017. viii, 11, 13

[36] Guillaume Bresson, Zayed Alsayed, Li Yu, and Sebastien Glaser. Simultaneous localization and mapping: A survey of current trends in autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 2(3):194–220, Sep 2017. 17, 18

[37] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986.

[38] Gail A. Carpenter, Stephen Grossberg, and John H. Reynolds. Artmap: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network. *Neural Networks*, 4(5):565–588, 1991.

[39] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, and Kevin Skadron. A performance study of general-purpose applications on graphics processors using cuda. *Journal of parallel and distributed computing*, 68(10):1370–1380, 2008. 25

[40] Yutian Chen, Wenyan Gan, Lei Zhang, Chong Liu, and Xianlei Wang. A survey on visual place recognition for mobile robots localization. page 187–192, Nov 2017. 17, 18

[41] Y. Chen, H. C. Liao, and T. Tsai. Online real-time task scheduling in heterogeneous multicore system-on-a-chip. *IEEE Transactions on Parallel and Distributed Systems*, 24(1):118–130, Jan 2013. 32, 33

[42] Kanika Chourasia. Autonomous vehicles: challenges, opportunities, and future implications for transportation policies. *International Journal of Research*, 6:459–463, 2019. 2

[43] Sylvain Colomer, Nicolas Cuperlier, Guillaume Bresson, Philippe Gaussier, and Olivier Romain. Lpmp: A bio-inspired model for visual localization in challenging environments. *Frontiers in Robotics and AI*, 8, 2022. 2, 3, 13, 19, 21, 33, 34, 41, 44, 48, 51, 56, 78, 86, 89

[44] Sylvain Colomer, Nicolas Cuperlier, Guillaume Bresson, and Olivier Romain. Forming a sparse representation for visual place recognition using a neurorobotic approach. In *ITSC 2021*, Indianapolis, United States, Sept. 2021. xii, 18, 44, 48, 49, 52, 56, 57, 58, 89

[45] Jason Cong, Muhuan Huang, Peichen Pan, Yuxin Wang, and Peng Zhang. *Source-to-Source Optimization for HLS*, pages 137–163. Springer International Publishing, Cham, 2016. 53

[46] Travis J. Crayton and Benjamin Mason Meier. Autonomous vehicles: Developing a public health research agenda to frame the future of transportation policy. *Journal of Transport and Health*, 6:245–252, 2017. 6

[47] Nicolas Cuperlier, Frederic Demelo, and Benoît Miramond. Fpga-based bio-inspired architecture for multi-scale attentional vision. In *2016 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, pages 231–232, 2016. 65

[48] Kumar M. Ayyagari M. Kumar G. Dargan, S. A survey of deep learning and its applications: A new paradigm to machine learning. archives of computational methods in engineering. *IEEE Access*, 27(4):1071–1092, 2020. 15, 16

[49] Qi Deng, Hao Sun, Fupeng Chen, Yuhao Shu, Hui Wang, and Yajun Ha. An optimized fpga-based real-time ndt for 3d-lidar localization in smart vehicles. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 68(9):3167–3171, 2021. 18

[50] Michael Dreschmann, Jan Heisswolf, Michael Geiger, Jürgen Becker, and Manuel HauBecker. A framework for multi-fpga interconnection using multi gigabit transceivers. In *2015 28th Symposium on Integrated Circuits and Systems Design (SBCCI)*, pages 1–6, 2015. 38, 39, 40

[51] Tarek Elouaret, Sylvain Colomer, Frederic Demelo, Nicolas Cuperlier, Olivier Romain, Lounis Kessal, and Stephane Zuckerman. Implementation of a bio-inspired neural architecture for autonomous vehicle on a reconfigurable platform. In *2022 IEEE 31st International Symposium on Industrial Electronics (ISIE)*, pages 661–666, 2022. viii, 23, 83

[52] Tarek Elouaret, Stéphane Zuckerman, Lounis Kessal, Yoan Espada, Nicolas Cuperlier, Guillaume Bresson, Fethi Ben Ouezdou, and Olivier Romain. Position paper: Prototyping autonomous vehicles applications with heterogeneous multi-fpgasystems. In *2019 UK/ China Emerging Technologies (UCET)*, pages 1–2, 2019. 37

[53] Yoan Espada, Nicolas Cuperlier, Guillaume Bresson, and Olivier Romain. From Neurorobotic Localization to Autonomous Vehicles. *Unmanned Systems*, 07(03):183–194, July 2019. 2, 13, 19, 21, 45, 49, 56

[54] Mikel Etxeberria-Garcia, Mikel Labayen, Maider Zamalloa, and Nestor Arana-Arexolaleiba. Application of computer vision and deep learning in the railway domain for autonomous train stop operation. In *2020 IEEE/SICE International Symposium on System Integration (SII)*, pages 943–948, 2020. 17

[55] Daniel J Fagnant and Kara Kockelman. Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations. *Transportation Research Part A: Policy and Practice*, 77:167–181, 2015. 2

[56] Péter Fankhauser, Michael Bloesch, and Marco Hutter. Probabilistic terrain mapping for mobile robots with uncertain localization. *IEEE Robotics and Automation Letters*, 3(4):3019–3026, 2018. ix, 26

[57] Laurent Fiack, Nicolas Cuperlier, and Benoît Miramond. Embedded and real-time architecture for bio-inspired vision-based robot navigation. *Journal of Real-Time Image Processing*, 10(4):699–722, Dec 2015. 3, 33, 41, 45

[58] H. Gao, B. Cheng, J. Wang, K. Li, J. Zhao, and D. Li. Object classification using cnn-based fusion of vision and lidar in autonomous vehicle environment. *IEEE Transactions on Industrial Informatics*, 14(9):4224–4231, Sep. 2018. 45

[59] Sourav Garg, Tobias Fischer, and Michael Milford. Where is your place, visual place recognition? In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, page 4416–4425, Montreal, Canada, Aug 2021. International Joint Conferences on Artificial Intelligence Organization. 17

[60] A Geiger, P Lenz, C Stiller, and R Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, Sep 2013. 21

[61] Farouk Ghallabi, Ghayath El-Haj-Shhade, Marie-Anne Mittet, and Fawzi Nashashibi. LIDAR-Based road signs detection For Vehicle Localization in an HD Map. In *IV'19 - IEEE Intelligent Vehicles Symposium*, Paris, France, June 2019. IEEE.

[62] MA Goodale and AD Milner. Separate visual pathways for perception and action. *Trends in Neurosciences*, 15(1):20–25, 1992.

[63] S Gourichon, J A Meyer, S H Ieng, L Smadja, and R Benosman. Estimating ego-motion using a panoramic sensor: Comparison between a bio-inspired and a camera-calibrated method. page 11, 2003.

[64] Roddy M Grieves and Kate J Jeffery. The representation of space in the brain. *Behavioural processes*, 135:113–131, 2017. 19

[65] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, and J. Cong. Fp-dnn: An automated framework for mapping deep neural networks onto fpgas with rtl-hls hybrid templates. In *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 152–159, 2017. 82

[66] Erico Guizzo. How google's self-driving car works. *IEEE Spectrum Online*, 18(7):1132–1141, 2011. 2

[67] K Guo, W Li, K Zhong, Z Zhu, S Zeng, S Han, Y Xie, P Debacker, M Verhelst, and Y Wang. Neural network accelerator comparison [online, last checked: 2021-11-10.]. 30, 66

[68] Kaiyuan Guo, Shulin Zeng, Jincheng Yu, Yu Wang, and Huazhong Yang. A survey of fpga-based neural network accelerator. *arXiv preprint arXiv:1712.08934*, 2017. 27, 32

[69] Michael E Hasselmo. The role of acetylcholine in learning and memory. *Current Opinion in Neurobiology*, 16(6):710–715, Dec 2006.

[70] Muhammad Imad, Muhammad Abul Hassan, Hazrat Junaid, Izaz Ahmad, et al. Navigation system for autonomous vehicle: A survey. *Journal of Computer Science and Technology Studies*, 2(2):20–35, 2020. 65

[71] Mhafuzul Islam, Mashrur Chowdhury, Hongda Li, and Hongxin Hu. Vision-based navigation of autonomous vehicles in roadway environments with unexpected hazards. *Transportation Research Record*, 2673(12):494–507, 2019. 14, 17

[72] Seigo Ito, Felix Endres, Markus Kuderer, Gian Diego Tipaldi, Cyrill Stachniss, and Wolfram Burgard. W-rgb-d: Floor-plan-based indoor global localization using a depth camera and wifi. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 417–422, 2014.

[73] Pierre-Yves Jacob, Giulio Casali, Laure Spieser, Hector Page, Dorothy Overington, and Kate Jeffery. An independent, landmark-dominated head-direction signal in dysgranular retrosplenial cortex. *Nature Neuroscience*, 20(2):173–175, Feb 2017.

[74] Malte Jaensch and Hannes Bantle. 100 experts, 1 opinion: Predicting future electric vehicle and powertrain component sales. In *CTI SYMPOSIUM 2018*, pages 196–209. Springer, 2020. 2

[75] Junekyo Jhung, Ho Suk, Hyungbin Park, and Shiho Kim. Hardware accelerators for autonomous vehicles. In *Artificial Intelligence and Hardware Accelerators*, pages 269–317. Springer, 2023. 30

[76] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014. 32

[77] James Kuffner. Systems and methods for detection by autonomous vehicles, Mar. 14 2019. US Patent App. 15/701,695. 2, 7, 45

[78] Vaibhav Kumar and ML Garg. Deep learning as a frontier of machine learning: A. *International Journal of Computer Applications*, 975:8887. 14, 15

[79] Liangzhen Lai and Naveen Suda. Rethinking machine learning development and deployment for edge devices. *arXiv preprint arXiv:1806.07846*, 2018. 65

[80] Bengio Y. Hinton G. LeCun, Y. Deep learning. *Nature 521*, page 436–444, 2015. 15

[81] Sanghoon Lee, Dongkyu Lee, Pyung Choi, and Daejin Park. Accuracy–power controllable lidar sensor system with 3d object recognition for autonomous vehicle. *Sensors*, 20(19):5706, 2020. 18

[82] Sang Wan Lee, John P. O'Doherty, and Shinsuke Shimojo. Neural computations mediating one-shot learning in the human brain. *PLOS Biology*, 13(4):1–36, 04 2015.

[83] George Lentaris, Ioannis Stratakos, Ioannis Stamoulias, Dimitrios Soudris, Manolis Lourakis, and Xenophon Zabulis. High-performance vision-based navigation on soc fpga for spacecraft proximity operations. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(4):1188–1202, 2020. ix, 27, 28

[84] Jesse Levinson, Jake Askeland, Jennifer Dolson, and Sebastian Thrun. Traffic light mapping, localization, and state detection for autonomous vehicles. In *2011 IEEE International Conference on Robotics and Automation*, pages 5784–5791, 2011. viii, 8

[85] Yixing Li, Zichuan Liu, Kai Xu, Hao Yu, and Fengbo Ren. A gpu-outperforming fpga accelerator architecture for binary convolutional neural networks. *J. Emerg. Technol. Comput. Syst.*, 14(2):18:1–18:16, July 2018. 45

[86] Shaoshan Liu, Liangkai Liu, Jie Tang, Bo Yu, Yifan Wang, and Weisong Shi. Edge computing for autonomous driving: Opportunities and challenges. *Proceedings of the IEEE*, 107(8):1697–1716, 2019. 24

[87] Yangfan Liu, Peng Liu, Yingtao Jiang, Mei Yang, Kejun Wu, Weidong Wang, and Qingdong Yao. Building a multi-fpga-based emulation framework to support networks-on-chip design and verification. *International Journal of Electronics*, 97(10):1241–1262, 2010. 39

[88] D.G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, page 1150–1157 vol.2. IEEE, 1999.

[89] Yifang Ma, Zhenyu Wang, Hong Yang, and Lin Yang. Artificial intelligence applications in the development of autonomous vehicles: a survey. *IEEE/CAA Journal of Automatica Sinica*, 7(2):315–329, 2020. 14

[90] Will Maddern, Geoffrey Pascoe, Chris Linegar, and Paul Newman. 1 year, 1000 km: The oxford robotcar dataset. *IJRR*. 48, 55, 56

[91] Kristiyan Manev, Joseph Powell, Kaspar Matas, and Dirk Koch. byteman: A bitstream manipulation framework. 11 2022. xi, 79, 81, 82

[92] Patrick Mannion. Vulnerable road user detection: state-of-the-art and open challenges. *arXiv preprint arXiv:1902.03601*, 2019. 6

[93] John Markoff. Google cars drive themseleves, in traffic. *The New York Times*, 9, 2010. 2

[94] Wim Meeus, Kristof Van Beeck, Toon Goedemé, Jan Meel, and Dirk Stroobandt. An overview of today's high-level synthesis tools. *Design Automation for Embedded Systems*, 16(3):31–51, 2012. 53

[95] Bartlett Mel and Christof Koch. Sigma-pi learning: On radial basis functions and cortical associative learning. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1990.

[96] O. Mencer, K. H. Tsoi, S. Craimer, T. Todman, W. Luk, M. Y. Wong, and P. H. W. Leong. Cube: A 512-fpga cluster. In *2009 5th Southern Conference on Programmable Logic (SPL)*, pages 51–57, 2009. 84

[97] I. Mezei and V. Malbasa. Using vhdl to improve an fpga based educational microcomputer. In *EUROCON 2005 - The International Conference on "Computer as a Tool"*, volume 1, pages 799–802, 2005. 31

[98] Takahiro Miki, Lorenz Wellhausen, Ruben Grandia, Fabian Jenelten, Timon Homberger, and Marco Hutter. Elevation mapping for locomotion and navigation using gpu. *arXiv preprint arXiv:2204.12876*, 2022. ix, 25, 26

[99] Maryam Mirsadeghi, Majid Shalchian, Saeed Reza Kheradpisheh, and Timothée Masquelier. Stidi-bp: Spike time displacement based error backpropagation in multilayer spiking neural networks. *Neurocomputing*, 427:131–140, 2021. 65

[100] Payal Mittal, Raman Singh, and Akashdeep Sharma. Deep learning-based object detection in low-altitude uav datasets: A survey. *Image and Vision Computing*, 104:104046, 2020. 65

[101] Sparsh Mittal. A survey of FPGA-based accelerators for convolutional neural networks. In *Neural Comput and Applic 32, 1109–1139 (2020)*, 2020. 65

[102] May-Britt Moser, David C. Rowland, and Edvard I. Moser. Place cells, grid cells, and memory. *Cold Spring Harbor Perspectives in Biology*, 7(2):a021808, Feb 2015. 50

[103] Razvan Nane, Vlad-Mihai Sima, Christian Pilato, Jongsok Choi, Blair Fort, Andrew Canis, Yu Ting Chen, Hsuan Hsiao, Stephen Brown, Fabrizio Ferrandi, et al. A survey and evaluation of fpga high-level synthesis tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(10):1591–1604, 2015. 31, 32, 53

[104] Tan Nguyen, Samuel Williams, Marco Siracusa, Colin MacLean, Douglas Doerfler, and Nicholas J Wright. The performance and energy efficiency potential of fpgas in scientific computing. In *2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 8–19. IEEE, 2020. 32

[105] JAH Nieuwenhuijsen. Diffusion of automated vehicles: a quantitative method to model the diffusion of automated vehicles with system dynamics. 2015. 2

[106] Sangyoon Oh, Minsub Kim, Donghoon Kim, Minjoong Jeong, and Minsu Lee. Investigation on performance and energy efficiency of cnn-based object detection on embedded device. In *2017 4th International Conference on Computer Applications and Information Processing Technology (CAIPT)*, pages 1–4. IEEE, 2017. 24

[107] Shane M. O'Mara. Spatially selective firing properties of hippocampal formation neurons in rodents and primates. *Progress in Neurobiology*, 45(3):253–274, Feb. 1995.

[108] M. Owaida and G. Alonso. Application partitioning on fpga clusters: Inference over decision tree ensembles. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, pages 295–2955, 2018. 82

[109] Darsh Parekh, Nishi Poddar, Aakash Rajpurkar, Manisha Chahal, Neeraj Kumar, Gyanendra Prasad Joshi, and Woong Cho. A review on autonomous vehicles: Progress, methods and challenges. *Electronics*, 11(14):2162, 2022. 2

[110] Minwoo Park, Jiebo Luo, Robert T. Collins, and Yanxi Liu. Beyond gps: determining the camera viewing direction of a geotagged image. In *Proceedings of the international conference on Multimedia - MM '10*, page 631. ACM Press, 2010. 17, 18

[111] L. Pezzarossa, A. T. Kristensen, M. Schoeberl, and J. Spars. Can real-time systems benefit from dynamic partial reconfiguration? In *2017 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*, pages 1–6, 2017.

[112] Tony A Plate. Randomly connected sigma–pi neurons can form associator networks. *Network: Computation in Neural Systems*, 11(4):321–332, Jan 2000.

[113] Guillem Pratx and Lei Xing. Gpu computing in medical physics: A review. *Medical physics*, 38(5):2685–2697, 2011. 25

[114] Murad Qasaimeh, Kristof Denolf, Jack Lo, Kees Vissers, Joseph Zambreno, and Phillip H. Jones. Comparing energy efficiency of cpu, gpu and fpga implementations for vision kernels. In *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*, pages 1–8, 2019. 30, 37, 66

[115] Avadhesh Rathi. *Real-Time Adaptation of Visual Perception*. PhD thesis, University of California, Riverside, 2022. 65

[116] Ratheesh Ravindran, Michael J. Santora, and Mohsin M. Jamali. Multi-object detection and tracking, based on dnn, for autonomous vehicles: A review. *IEEE Sensors Journal*, 21(5):5668–5677, 2021. 17

[117] Tyler G.R. Reid, Sarah E. Houts, Robert Cammarata, Graham Mills, Siddharth Agarwal, Ankit Vora, and Gaurav Pandey. Localization requirements for autonomous vehicles. *SAE International Journal of Connected and Automated Vehicles*, 2(3), Sep 2019. xii, 6, 7

[118] David B Richardson. Electric vehicles and the electric grid: A review of modeling approaches, impacts, and renewable energy integration. *Renewable and Sustainable Energy Reviews*, 19:247–254, 2013. 7

[119] Francisca Rosique, Pedro J. Navarro, Carlos Fernández, and Antonio Padilla. A systematic review of perception system and simulators for autonomous vehicles research. *Sensors*, 19(3):648, Feb 2019. 18

[120] Enrico Rossi, Marvin Damschen, Lars Bauer, Giorgio Buttazzo, and Jörg Henkel. Preemption of the partial reconfiguration process to enable real-time computing with fpgas. *ACM Trans. Reconfigurable Technol. Syst.*, 11(2):10:1–10:24, July 2018. 33

[121] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, page 2564–2571. IEEE, Nov 2011.

[122] S. Saha, A. Sarkar, A. Chakrabarti, and R. Ghosh. Co-scheduling persistent periodic and dynamic aperiodic real-time tasks on reconfigurable platforms. *IEEE Transactions on Multi-Scale Computing Systems*, 4(1):41–54, Jan 2018. 33

[123] Aman B Saleem. Two stream hypothesis of visual processing for navigation in mouse. *Current Opinion in Neurobiology*, 64:70–78, Oct 2020.

[124] Gernot Sauter, Marcel Doring, and Rik Nuyttens. High performance pavement markings enhancing camera and LiDAR detection. *IOP Conference Series: Materials Science and Engineering*, 1202(1):012033, nov 2021. viii, 12, 13, 14

[125] Wilko Schwarting, Javier Alonso-Mora, and Daniela Rus. Planning and decision-making for autonomous vehicles. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(1):187–210, May 2018. 18

[126] Eric L. Schwartz. Computational anatomy and functional architecture of striate cortex: A spatial mapping approach to perceptual coding. *Vision Research*, 20(8):645–669, jan 1980.

[127] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv:1312.6229 [cs]*, Feb 2014. arXiv: 1312.6229. 17, 18

[128] Weijing Shi, Mohamed Baker Alawieh, Xin Li, and Huafeng Yu. Algorithm and hardware implementation for visual perception system in autonomous vehicle: A survey. *Integration*, 59:148–156, 2017. 65

[129] Sayem Mohammad Siam and Hong Zhang. Fast-seqslam: A fast appearance based place recognition algorithm. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, page 5702–5708. IEEE, May 2017. 17

[130] Lenin Singaravelu, Calton Pu, Hermann Härtig, and Christian Helmuth. Reducing tcb complexity for security-sensitive applications. volume 40, pages 161–174, 10 2006. 34

[131] Santokh Singh. Critical reasons for crashes investigated in the national motor vehicle crash causation survey, 2015. 6

[132] Robert Spangenberg, Daniel Goehring, and Raúl Rojas. Pole-based localization for autonomous vehicles in urban scenarios. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2161–2166, 2016. 8

[133] Asim Suhail, Manoj Jayabalan, and Vinesh Thiruchelvam. Convolutional neural network based object detection: A review. *Journal of critical reviews*, 7(11):786–792, 2020. 24

[134] Rolls Edmund T. Spatial view cells and the representation of place in the primate hippocampus. *Hippocampus*, 9(4):467–480, 1999.

[135] Qi Tang, Zhe Wang, Biao Guo, Li-Hua Zhu, and Ji-Bo Wei. Partitioning and scheduling with module merging on dynamic partial reconfigurable fpgas. *ACM Trans. Reconfigurable Technol. Syst.*, 13(3), Aug. 2020.

[136] H. Topcuoglu, S. Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, March 2002. 33

[137] Akihiko Torii, Josef Sivic, Tomas Pajdla, and Masatoshi Okutomi. Visual place recognition with repetitive structures. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013. 17

[138] Stefan Treue. Visual attention: the where, what, how and why of saliency. *Current Opinion in Neurobiology*, 13(4):428–432, Aug. 2003.

[139] Konstantinos A Tsintotas, Loukas Bampis, and Antonios Gasteratos. Visual place recognition for simultaneous localization and mapping. *Autonomous Vehicles Volume 2: Smart Vehicles*, pages 47–79, 2022. 65

[140] Yaman Umuroglu, Yash Akhauri, Nicholas James Fraser, and Michaela Blott. Logicnets: Co-designed neural networks and circuits for extreme-throughput applications. In *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, 2020. 30, 65

[141] Anuj Vaishnav, Khoa Dang Pham, Joseph Powell, and Dirk Koch. Fos: A modular fpga operating system for dynamic workloads. *ACM Transactions on Reconfigurable Technology and Systems Volume*, (28):13Issue 4December 2020 Article No.: 20pp 1–28, 2020. ix, 35, 36, 89

[142] Jessica Van Brummelen, Marie O'Brien, Dominique Gruyer, and Homayoun Najjaran. Autonomous vehicle perception: The technology of today and tomorrow. *Transportation Research Part C: Emerging Technologies*, 89:384–406, Apr 2018. 18

[143] Liang Wang, Yihuan Zhang, and Jun Wang. Map-based localization method for autonomous vehicles using 3d-lidar **this work is supported in part by the national natural science foundation of china under grant no. 61473209. *IFAC-PapersOnLine*, 50(1):276–281, 2017. 20th IFAC World Congress. viii, 9

[144] Xiaofei Wang, Yiwen Han, Victor C. M. Leung, Dusit Niyato, Xueqiang Yan, and Xu Chen. Convergence of edge computing and deep learning: A comprehensive survey. *IEEE Communications Surveys Tutorials*, 22(2):869–904, 2020. 8, 65

[145] Ryan W. Wolcott and Ryan M. Eustice. Visual localization within lidar maps for automated urban driving. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 176–183, 2014. viii, 9, 10

[146] Tian Xia, Ye Tian, Jean-Christophe Prévotet, and Fabienne Nouvel. Ker-one: A new hypervisor managing fpga reconfigurable accelerators. *Journal of Systems Architecture*, 98:453–467, 2019. ix, 34, 35, 38, 81, 89

[147] Yanming Yang, Xin Xia, David Lo, and John Grundy. A survey on deep learning for software engineering. *ACM Comput. Surv.*, dec 2021. Just Accepted. 15

[148] Ryan M. Yoder, Benjamin J. Clark, and Jeffrey S. Taube. Origins of landmark encoding in the brain. *Trends in Neurosciences*, 34(11):561–571, Nov 2011.

[149] Skrynnik A. Krishtopik A. et al Yudin, D.A. Object detection with deep neural networks for reinforcement learning in the task of autonomous vehicles path planning at the intersection. *Opt. Mem. Neural Networks*, (28):283–295, 2019. 17

[150] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A survey of autonomous driving: Common practices and emerging technologies. *CoRR*, abs/1906.05113, 2019. 14, 17

[151] Mubariz Zaffar, Shoaib Ehsan, Michael Milford, David Flynn, and Klaus McDonald-Maier. Vpr-bench: An open-source visual place recognition evaluation framework with quantifiable viewpoint and appearance change. *arXiv:2005.08135 [cs]*, May 2020. arXiv: 2005.08135. 14, 17, 18, 19

[152] S.W. Zhang, K. Bartsch, and M.V. Srinivasan. Maze learning by honeybees. *Neurobiology of Learning and Memory*, 66(3):267–282, Nov 1996. 17

[153] W. Zhang, J. Zhang, M. Shen, G. Luo, and N. Xiao. An efficient mapping approach to large-scale dnns on multi-fpga architectures. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1241–1244, 2019. 82

[154] Xiwu Zhang, Lei Wang, and Yan Su. Visual place recognition: A survey from deep learning perspective. *Pattern Recognition*, 113:107760, 2021. 17

[155] Jiangpeng Zhu, Guofeng Yang, Xuping Feng, Xiyao Li, Hui Fang, Jinnuo Zhang, Xiulin Bai, Mingzhu Tao, and Yong He. Detecting wheat heads from uav low-altitude remote sensing images using deep learning based on transformer. *Remote Sensing*, 14(20):5141, 2022. 65