



Article

A Comparison of FPGA and GPGPU Designs for Bayesian Occupancy Filters

Luis Medina ¹, Miguel Diez-Ochoa ², Raul Correal ², Sergio Cuenca-Asensi ¹, Alejandro Serrano ¹ , Jorge Godoy ³, Antonio Martínez-Álvarez ¹ and Jorge Villagra ^{3,*} 

¹ University Institute for Computing Research, University of Alicante, 03690 San Vicente del Raspeig, Spain; lmedina@dtic.ua.es (L.M.); sergio@dtic.ua.es (S.C.-A.); aserrano@dtic.ua.es (A.S.); amartinez@dtic.ua.es (A.M.-Á.)

² Ixion Industry & Aerospace SL, Julian Camarilo 21B, 28037 Madrid, Spain; mdiezochoa@ixion.es (M.D.-O.); rcorreal@ixion.es (R.C.)

³ Centre for Automation and Robotics (UPM-CSIC), 28500 Arganda del Rey, Spain; jorge.godoy@csic.es

* Correspondence: jorge.villagra@csic.es; Tel.: +34-918-711-900

Received: 15 September 2017 ; Accepted: 6 November 2017 ; Published: 11 November 2017

Abstract: Grid-based perception techniques in the automotive sector based on fusing information from different sensors and their robust perceptions of the environment are proliferating in the industry. However, one of the main drawbacks of these techniques is the traditionally prohibitive, high computing performance that is required for embedded automotive systems. In this work, the capabilities of new computing architectures that embed these algorithms are assessed in a real car. The paper compares two ad hoc optimized designs of the Bayesian Occupancy Filter; one for General Purpose Graphics Processing Unit (GPGPU) and the other for Field-Programmable Gate Array (FPGA). The resulting implementations are compared in terms of development effort, accuracy and performance, using datasets from a realistic simulator and from a real automated vehicle.

Keywords: Bayesian occupancy filter; FPGA; GPGPU; embedded system; ADAS

1. Introduction

Intelligent vehicle technology is advancing at a vertiginous pace. However, the complexity of some highly uncertain and dynamic urban driving scenarios still hampers the deployment of fully automated vehicles. One of the most important challenges in those scenarios is the accurate perception of static and moving objects, to properly understand the spatio-temporal relationship between the subject vehicle and the relevant entities.

In well structured driving environments, such as highways, the types of static and dynamic objects are easily modeled and tracked using geometrical models and their parameters. However, urban driving scenarios are so heterogeneous and unpredictable that they are extremely complex to manage under a feature-based perception paradigm. In addition, the associated tracking methodology raises the classic problem of object association and state estimation, which are highly coupled.

Occupancy grids [1] overcome these difficulties, and in particular the Bayesian Occupancy Filter (BOF) [2] by projecting objects onto a compact regularly subdivided probabilistic grid, and tracking them as a set of occupied or dynamic cells. Under this paradigm, there is no need for higher level object models, resulting in: (i) a much higher insensitivity to the extreme variability of objects; and, (ii) avoidance of the association problem. In addition, it produces a compact representation model, as empty space is also conveniently represented for proper situational awareness.

The main drawback of this approach is its computational cost, unaffordable for automotive embedded systems [3]. Currently, common Electronic Control Unit (ECU), based on single or multicore microcontrollers (MCUs) [4] are used for low to medium-speed sensor data processing. The overall

performance of these systems depends on the number of cores, their operating frequencies, memory bandwidth, and size. However, they are in general insufficient for processing high-demand algorithms in parallel with other data. As a result, MCU resource sharing (data acquisition, memory bus limitations, number of cores, frequency limitations etc.) to fuse data from high bandwidth sensors is prohibitive and the data are frequently fed into an occupancy grid that models the environment.

Likewise, current ECUs are dedicated embedded components, each one subject to a series of functional and safety requirements. As a result, state-of-the-art vehicles can carry up to 100 ECUs on board using 5 km cable lengths. This situation creates a significant series of disadvantages, such as material costs, communication bandwidth limitations, latency problems, higher power consumption, potential robustness issues and high development and maintenance costs.

The integration of high-performance computing units are needed to overcome current limitations, fed by multiple sensor technologies such as radar, computer vision systems, LiDARs (Light Detection and Ranging) or even ultrasounds, complementing and reinforcing each other, and communicating over a reliable high-bandwidth network. Optimal usage of a huge amount of heterogeneous information requires the use of novel computing platforms and powerful algorithms that exploit their architecture as best they can.

Heterogeneous System on Chip (SoC) platforms can efficiently use different sensors for multiple functions and interconnect them with other systems—e.g., a camera is used for object detection and recognition, localization, and lane estimation. Additionally, parallel-processing resources offered by FPGA and GPGPU can be exploited to implement highly parallelizable computations. As a result, different functions, which, when available, are currently implemented in separate ECUs, can potentially be integrated within a single domain of a high-performance computing unit.

An in-depth study in this paper of the parallelization opportunities of new heterogeneous SoCs addresses the specific problem of object detection and tracking using the BOF paradigm. Although some previous works have optimized BOF designs for GPGPU platforms, in this paper, to the best of our knowledge, we are presenting the first implementation of the BOF using FPGA. In addition, a thorough comparison of BOF between heterogeneous SoCs using GPGPU (Nvidia Tegra K1) and FPGA (Xilinx Zynq) was carried out, using both synthetic sensor data from simulated and experimental datasets gathered by a real automated vehicle on open roads.

The remainder of this paper is as follows. Section 2 introduces and analyzes previous work on embedded systems for Advanced Driver Assistance Systems (ADAS), particularly those using multiple range sensors for object detection, world modeling and driving situation understanding. Then, a brief summary of the BOF is presented in Section 3, serving as the foundation for a description of the problem and the methodology that is followed, both of which are detailed in Section 4. The two chosen computing platforms are introduced in Sections 5 and 6, where specific requirements and constraints are detailed, followed by a complete description of the design solution that is adopted. The results of such implementations are compared and analyzed with simulated and experimental data in Section 7. Finally, the paper draws to a close with some concluding remarks and directions for future work in Section 8.

2. State of the Art in Embedded Systems for Multi-Sensor Perception

On-road object detection, classification, and tracking has been a topic of intense interest over recent years [5]. The complexity of such tasks where vehicles and Vulnerable Road Users (VRU) (often pedestrians, motorcyclists, and cyclists ...) coexist in uncertain and heterogeneous environments makes the use of multi-sensor architectures very necessary for automated driving systems. Indeed, a variety of sensing modalities have become available for on-road vehicle detection, including radar, LiDAR, ultrasounds and computer vision.

While LiDARs have high accuracy under optimal conditions, wide angle coverage, and precise target location, their performance is less accurate in inclement weather and/or when dirt collects on a sensor lens [6]. Most radars use some form of beam scanning to determine whether targets are in

the same or adjacent roadways, or in oncoming lanes. Microwave radars have a lengthy detection range and are able to operate in inclement weather, but they have a narrow field of view and are not robust enough for multi-target precise motion estimation [7], particularly around road curves. The images from a video camera can provide information on the presence of obstacles at short/medium distances [8]. These sensors have a wider field of vision and can recognize and categorize objects. However, they are not as reliable as radar when ascertaining depth-perception information. In addition, as humans vary significantly in size and shape, VRU detection is not robust enough, especially in crowded areas. Moreover, extreme lighting conditions (day/night) can dramatically reduce the effectiveness of the detection algorithms.

Modern ADAS applications use sensor fusion to take full advantage of the information that each sensor collects, so that the strengths of all these technologies can be intelligently combined. In some approaches, a radar or LiDAR is used to detect potential candidates, and then, during a second stage, Computer Vision is applied to analyze the objects that are detected. Other strategies claim the use of multiple-range sensors at a very low level to facilitate data fusion and ulterior object tracking. The Bayesian Occupancy Filter presents very good behavior under the latter paradigm. However, multiple-range sensors and other related grid-based approaches have typically been intractable for automotive embedded systems. Indeed, the commonly used MCU in the vehicle has insufficient processing power to process the various sensor inputs from multiple radars, cameras, laser scanners, and ultra-sonic sensors.

Future embedded solutions for perception will need to process high levels of throughput from heterogeneous sensors, providing real-time processing capabilities for computationally demanding algorithms while guaranteeing several non-functional properties such as reliability, real-time performance, low-cost, spatial constraints, low-power constraints, flexibility and short time-to-market [4].

The existing hardware (HW) technologies to cope with all the above requirements are as follows: (i) Application-Specific Integrated Circuits (ASIC), customized circuits for particular uses, (ii) FPGA, (iii) GPGPU, (iv) Digital Signal Processors (DSP), and (v) microprocessors (μ P). Table 1 summarizes the main advantages and drawbacks of each technology and provides some specific examples of ADAS applications where the perception of the environment plays a key role.

Table 1. Examples of latest-generation embedded ADAS using different HW technologies.

Tech.	Pros	Cons	Examples
ASIC	High performance	Expensive for prototyping	Disparity maps [9]
	Low-power consumption	Not reconfigurable	Object and lane detection [10]
FPGA	Low-power consumption	Poor for serial processing	Lane departure warning [11]
	Good at low-level computing	Complex to program	Pedestrian recognition [12]
GPGPU	Highly parallelizable	Power-hungry	Pedestrian detection [13]
	Programming flexibility	Complex to program	Road detection [14]
DSP	Well suited for image processing	Medium speed performance	Object detection [15,16]
	Good price to performance ratio	Complex to program	Lane departure warning [17]
μ P	Best for high-level processing	Poor parallelization	Lane departure warning [18]
	Easy to program	High power consumption	Vehicle detection [19]

Note that most of these recent works use (mono or stereo) computer vision. The use of multi-sensor architectures and high-level functionalities is still very limited in the state-of-the-art embedded systems for intelligent vehicles. A limitation that is mainly due to both a lack of computing resources and very specific, rigid designs, unable to combine different functional components on a single computing platform. These difficulties have forced a shift from homogeneous machines relying on a single kind of fast-processing element to heterogeneous architectures combining different kinds of processors

(such as MCUs, GPGPUs, DSPs, and FPGA), each specialized for certain tasks and programmed in a highly parallel fashion. Their weak points are poor optimization of available resources for performance and low energy consumption. Some examples of this new trend combine (i) multi-cores with FPGA for lane departure warning [20] and traffic sign recognition [21,22]; (ii) multi-cores with GPGPU [13]; and even, (iii) FPGA and GPGPU [23].

These new platforms allow us to consider traditionally prohibitive high-level processing for object detection and tracking using heterogeneous sensors. BOF real-time implementations are now realistic in this new context. One specific feature is that cells are independent, permitting dedicated hardware implementations leading to increased performance. Indeed, the cell independence hypothesis and sensor measurements tolerate loops in the Occupancy Grid algorithm that are implemented in a parallel manner. As a result, some prior works have explored the power of GPGPUs to implement different perception strategies based on occupancy grids, using multi-layer LiDARs [24] and fusing LiDAR and radar [25].

More recently, the integration of the occupancy grid multi-sensor fusion algorithm into low-power multi-core architectures has also been investigated [26,27]. These attempts use floating-point representation for probability estimation and fusion, but highly constrained embedded platforms will not often feature a floating-point unit. A fixed point design of an occupancy grid representation—BOF preprocessing step—was introduced by [28] to overcome that limitation. It shows good behavior, thereby opening the door to more HW-oriented implementations of the BOF (e.g., using FPGA). Indeed, in contrast to GPGPU architecture, FPGAs are designed for extreme customization. The current FPGA fine-grain architecture still takes advantage of irregular parallelism and non-standard data types. In addition, recent improvements are extending their applicability to different high-performance applications. This brings FPGAs closer in raw performance to state-of-the-art GPGPU-based SoC devices, as will be shown in Section 7.

3. Bayesian Occupancy Filter

Perception is one of the main processes that an autonomous vehicle has to perform. Within this process, data from the on-board sensors, typically multi-layer LiDARs and stereo cameras, are fed into the system for processing. The main purpose of the BOF framework when using those data feeds is to identify dynamic objects in a driving situation, estimating the position, trajectory, and velocity of each mobile object. These are then used for subsequent risk assessment and collision avoidance.

The BOF implementation is based on the work of INRIA (Institut National de Recherche en Informatique et en Automatique). A detailed description of its complexity and extension is unfortunately outside the scope of this paper. This section presents an overview of the framework; further details can be found in [29–32], also cited throughout the section.

The process follows a series of sequential steps, where the output of each one is fed as input into the following one; see Figure 1. The emphasis on acceleration and parallelizable opportunities has been focused on the system core, which comprises the heaviest computational processes. This core updates the prior state, according to the dynamics of the vehicle, by applying motion detection filtering [29], by computing the posterior state from new observations, and by updating and computing new velocity estimations of the cells, described below in further detail. Other processes, such as the creation of a representation of the environment [31]—the probability occupancy map, the clustering process, to move from cell level to object level [32] and the identification/classification of each object [30] are not included in the initial acceleration and parallelization efforts.

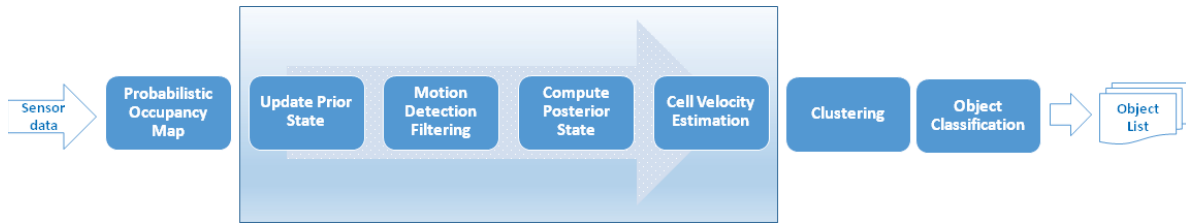


Figure 1. Bayesian Occupancy Filter process flow diagram and core system.

Initially, a representation of the environment must be created. A probabilistic occupancy map is built from sensor observations. The space in this grid-based map is discretized into $M \times N$ cells. Each cell contains the probability for the area that will be occupied. In the case of having multiple sensors, or a multi-layer LiDAR, an independent occupancy map is computed for each sensor/layer, see Figure 2. These multiple observations, and their corresponding occupancy maps, are then fused together into a unique representation of the environment, using a sophisticated weighted sum of observations, a mechanism known as Linear Opinion Pools [31]. The system uses that mechanism to compute an integrated probability value over the occupancy of each cell of the grid, given the opinion of a number of sensors. Also, a measure of the confidence for such estimations is also computed and taken into account to calculate the integrated occupancy probability for each cell. Factors such as confidence in each LiDAR measurement in frontal and rear areas of each potential hit and the inclination angle of each LiDAR plane layer—with respect to the road—influence the overall confidence level of each observation.

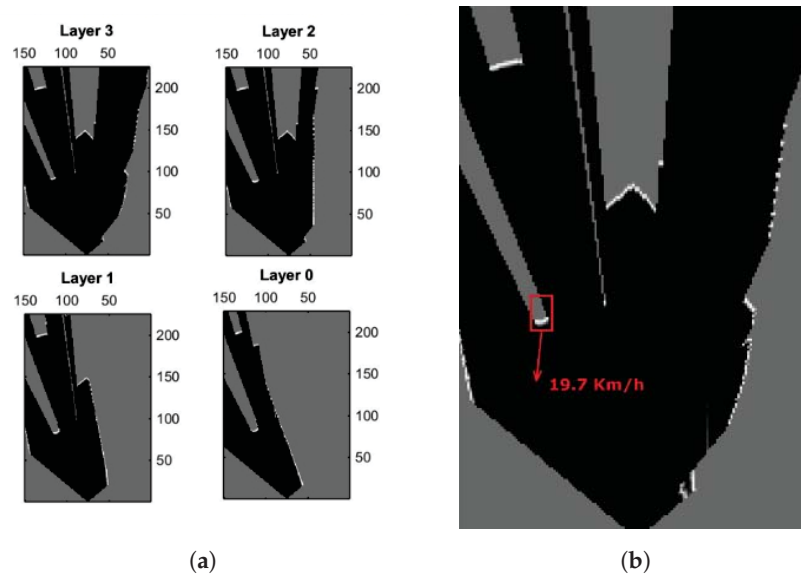


Figure 2. (a) Probabilistic occupancy map for each LiDAR layer, (b) integrated occupancy map. Dark areas represent empty space, white areas represent objects/occupied parts and unknown regions are in gray.

Figure 3 shows the BOF general dataflow. The algorithm calculates, for every time step (t), the state of the $M \times N$ cells in the occupancy grid (i.e., their Probability of occupancy P , and their distribution of velocity BOF4D), based on the previous state ($t - 1$), the observed occupancy probabilities (Z), and the linear and angular velocities of the vehicle (U). The computation cycle takes place in two steps: UpdatePrior step and UpdatePosterior step. The results are notated with the corresponding subscripts p_r and p_o .

UpdatePrior: During the UpdatePrior step, a new prediction is created from the estimation of the previous state $(P_{po}, BOF4D_{po})^{t-1}$. This prediction is computed by transforming previous data according to the odometry, linear and angular velocities (U), over a given period of time. In a first process named *CellTransform*, every cell, represented by the coordinates of its central point, is transformed according to this dynamic data and projected onto a new position on the predicted map.

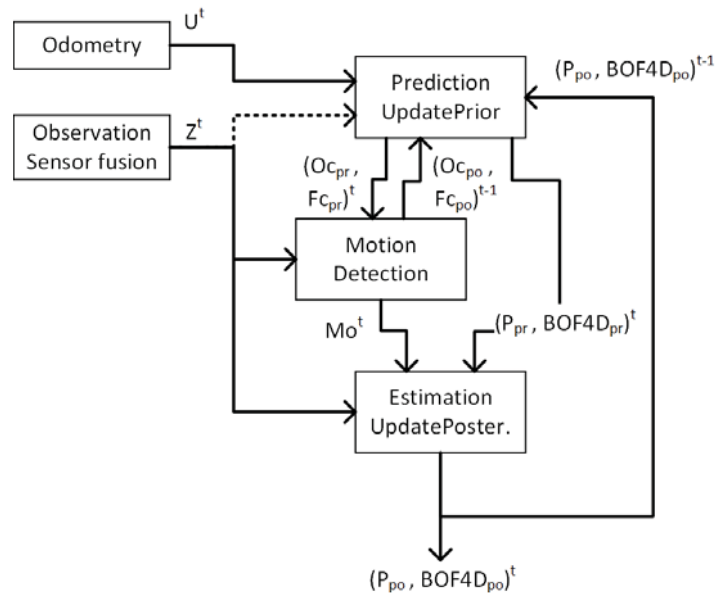


Figure 3. Bayesian Occupancy Framework general dataflow.

Localization errors may occur, due to uncertainty, sensor inaccuracy and external hazards such as skidding, where the dynamics of the vehicle might not be precisely represented through proprioceptive information. A second process, *OdomFix*, is applied to deal with that uncertainty problem and to update the Prior step data with greater accuracy. Instead of simply computing one transformation according to the vehicle dynamics based on sensor information, a series of potential transformations were calculated. Each transformation is then compared with the new observed occupancy grid (Z) and evaluated according to a score function. This function maintains a series of counters on the number of times each cell was observed as either empty (free) or occupied in the past (Fc, Oc) [29]. The candidate transformation with the highest score is then chosen as the prediction of the prior state information $(P_{pr}, BOF4D_{pr})^t$, and the free and occupied counters are updated according to this transformation $(Oc_{pr}, Fc_{pr})^t$.

UpdatePosterior: Once a prediction from the previous state has been computed, an estimation on the current state can be calculated, fusing that prediction with new observations. It is important to note this framework is designed to detect moving objects. However, most of the map or most of its cells turn out to contain information regarding static areas/objects. There is therefore no need to estimate the velocity for all those cells, which means that there is no need to update their velocity distribution, resulting in a huge saving of computation time. Therefore, a *Motion Detection* filtering step is all that is needed to make those savings through an heuristic function that classifies the cells as either static or dynamic, considering the values on either the free or the occupied counters. Therefore, if a given cell has been recorded as empty over some time in the past, in accordance with the free and occupied counter, and then appears as occupied in the most recent observation, it will be considered as a dynamic cell. A specific ratio has to be met between the number of times the cell was observed to be either empty or occupied, according to the counters, before it may be considered as dynamic. As a

result, only those cells, marked in the Mo^t grid, will be taken into account to compute the posterior state $(P_{po}, BOF4D_{po})^t$.

A series of hypotheses on the velocity of every dynamic cell are proposed, to estimate the velocity and trajectory of dynamic objects from occupancy probabilities. An initial 2D uniform probability distribution on the velocity hypotheses, vx and vy , is created for each cell Figure 4a (top). The distribution is discretized into $V \times V$ values and initialized to $1/\langle \text{number of cells} \rangle$ ($1/2500$ in the example). At each time step, besides updating the previous occupancy map to create a prediction, as previously described, a new probabilistic occupancy map is computed from new observations. They are then both fused together, obtaining an updated probabilistic occupancy map. The new observation data is also used to update the probability distribution on the velocity of each dynamic cell, propagating the probabilities using a *Bayesian* filtering mechanism [2]. In this propagation, each velocity hypothesis is updated according to the occupancy probability and probability distribution of the velocities of the *anterior*, which is the cell providing the prediction. The information on this cell is taken from the previous time step to establish whether a given velocity hypothesis is true for a given cell.

Over time, the distribution probabilities converge and, in some cases, one of them is prominent, see Figure 4a (bottom). The most prominent probability is then taken as the estimated velocity for that cell, provided its value is over a given minimum threshold to be considered, obtaining its module and trajectory from vx and vy . It is important to note that only the probability distribution of the dynamic cells are updated and propagated at each time step. The velocity hypotheses of the cells that are considered static cells are set to a uniform distribution.

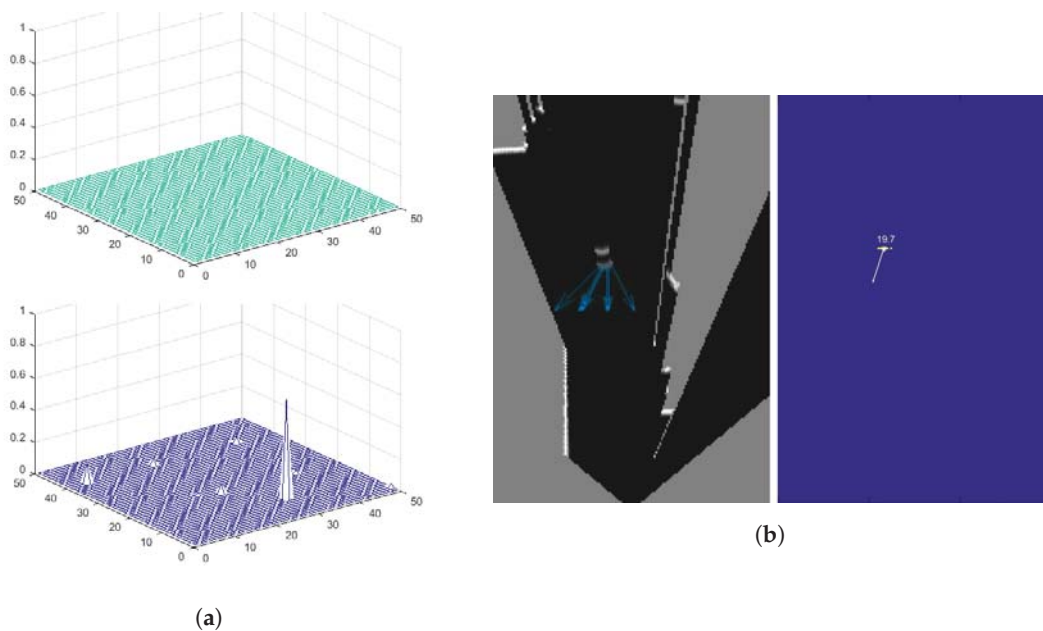


Figure 4. (a) Initial uniform probability distribution on the velocities (top) and convergence of probabilities (bottom), (b) clustering process grouping independent dynamic cells into objects.

So far the core of the BOF framework has been described. However, two processes still remain that are worth mentioning. The environment representation and tracking process have up until now been performed at the cell level. A process that is convenient, as it eliminates the data association and occlusion problems. However, the expected output of the system is a list of dynamic objects along with their associated position, trajectory, and velocity. Therefore, an object-level representation is necessary. To do so, a clustering process is applied to group independent cells into objects, see Figure 4b. A set of

cells that belong to the same object can only be considered if they meet two conditions: the condition of spatial constraint, meaning that the cells have to be close enough to be considered part of the same object, according to a configurable distance parameter; and, similarity of their trajectories and velocities, implying that all the cells follow a similar trajectory at a similar velocity. Such a similarity is computed according to some configurable parameter relating to the maximum permitted differences in their trajectory orientations and velocity modules. The combination of both constraints avoids grouping together cells that belong to different objects, despite their physical proximity, as in the case of observing two cars at an intersection that will cross the path of the host vehicle from two opposing perpendicular directions.

Finally, accurate classification of a moving object in urban traffic scenarios is a key element for safe decision-making in intelligent vehicles. An object identification process, using a classifier combining grid-based footprints and speed estimations, aims to recognize each detected object at the scene and to classify them into different categories, such as vehicles or pedestrians [30]. As a result, the BOF framework delivers a list of dynamic objects that have been identified at the scene, along with their position and trajectory in relation to our vehicle, velocity and type of object.

All this information is then fed into a risk assessment module, part of the control subsystem, in charge of decision processes on directions and the computation of vehicle maneuvering.

4. Problem Description

The work described in the following 3 sections aims to compare different performance and functional metrics of the BOF design and implementation using on the one hand a Multi-processor System on Chip (MPSoC) with a GPGPU and on the other hand a MPSoC with an FPGA. This comparison will be conducted using synthetic data from a realistic simulation environment and experimental data from an automated vehicle driving on open roads.

The inputs to both designs are a set of measurements from the CAN (Controller Area Network) bus of the vehicle, including the linear velocity and heading of the vehicle in the given period of time (so-called odometry in Section 3), and those transmitted by the multi-layer LiDAR sensing the environment. The precision and the frequency at which these sensors update are specified in the following two sub-sections, where simulation and experimentation settings are described. In addition, the algorithmic output is expected to be the same in both approximations, namely a matrix of 240×160 cells including occupancy probability and 2D velocity estimation—from a discretized distribution of 15×15 values).

Simulation: SCANer studio is a complete software tool meeting all the challenges of driving simulation. It includes models of sensors, vehicles and traffic interactions. With regard to the sensor models, the framework can simulate GPS, Camera, Radar, LiDAR and ultrasonic sensors. In particular for the LiDAR sensor, the simulation tool uses the ray tracing method to compute the results. Each ray is drawn in the graphic display and the rays are regularly drawn inside the detection arc (defined by the sensor position and a depth measurement).

Figure 5 shows a simulated driving situation and how a vehicle, equipped with an emulated multi-layer IBEO Lux LiDAR, perceives the environment and other vehicles. This device consists of 4 different layers vertically separated at intervals of 0.8° from each other, pointing from 1.2° above the horizontal plane (top left image in Figure 5) to -1.2° below (bottom right image in Figure 5). The data generated by this multi-layer LiDAR model is used to generate an integrated occupancy grid, as illustrated in Figure 2, and to extract the relevant footprints of the scene (Figure 6).

The dataset generated with this simulation framework is recorded from a pre-defined route of 650 m, which is completed at a variable speed after 120 s. The ego-vehicle always follows traffic regulations and it encounters other vehicles (7141 samples) or pedestrians (1861 samples). Note that all of the 1554 observations in the dataset may contain one or several of these samples. The LiDAR frequency is 12.5 Hz, while the longitudinal and angular speed of the vehicle is measured at 25 Hz.

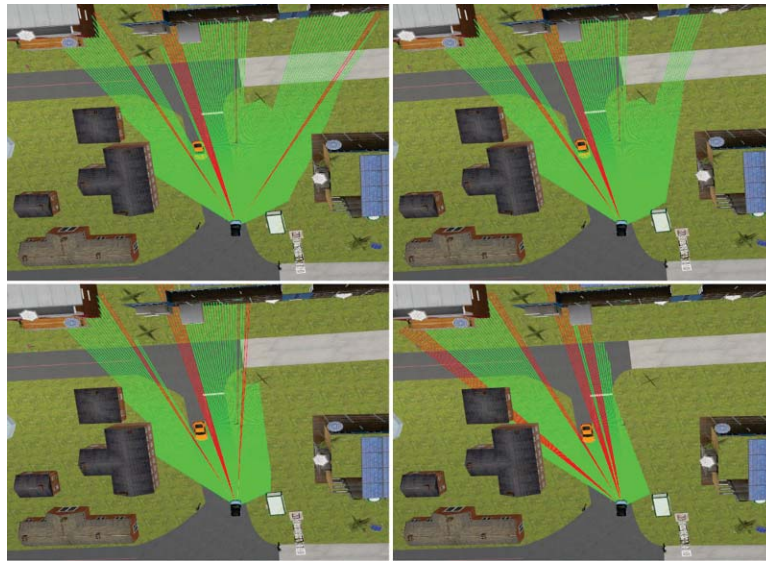
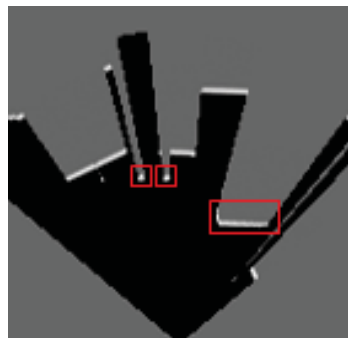


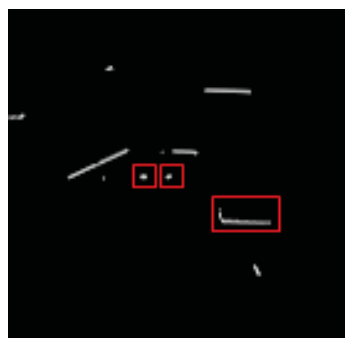
Figure 5. Fields of view of each LiDAR's layer in each subfigure. Red rays are those that do not find any target, whose length is the LiDAR maximum depth (200 m for the IBEO Lux); green rays refer to those that collide in the 3D space, with a length that is the distance between the sensor and the target.



(a)



(b)



(c)

Figure 6. (a) Snapshot of a scene taken from the dataset where pedestrians and vehicles are detected (red boxes); (b) Image from camera, occupancy grid combining the 4 layers of the LiDAR; (c) the resulting set of footprints at the scene.

Experimentation: A dataset has been recorded using an automated vehicle, to complement the results obtained using the simulation environment. The picture on the left-hand-side of Figure 7 shows the architecture of the vehicle, where the most relevant equipments are zoomed and numbered (note the IBEO Lux LiDAR in number 2). In addition to these visible parts, the SW architecture integrates the data from internal sensors (steering wheel, yaw rate, wheel speed, lateral and longitudinal accelerometers) that circulate through the CAN bus. The sampling frequency of these data is identical to those described in the simulation framework, parameterized to be consistent with the real on-board sensors.



Figure 7. Automated vehicle architecture.

The experimental recording consisted of a 3350.3 m urban and inter-urban route (Figure 8a), during 356 s, where different types of vehicles and pedestrians appear in the vehicle driving scene throughout the 4447 cycles/observations. As can be appreciated in Figure 8b, the vehicle speed never exceeds 70.8 km/h, but there are several stretches where there is a meaningful steering angle variation (a maximum of $284^\circ/s$), leading to demanding angular speed for the BOF computation.

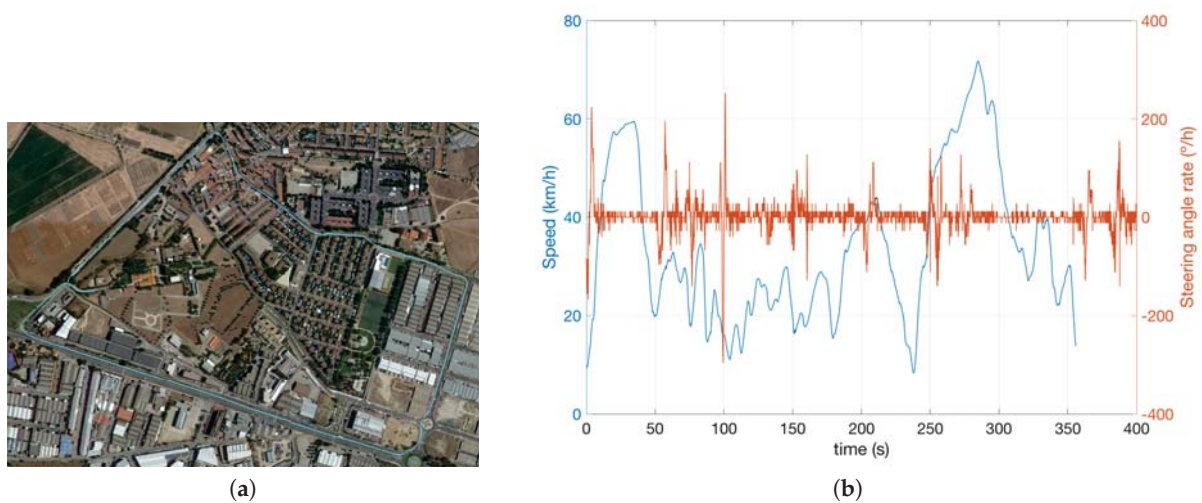


Figure 8. (a) Path followed by the vehicle (b) Speed and steering angle of the vehicle.