Integration of ADAS algorithm in a Vehicle Prototype

Jerome Lussereau, Procópio Stein, Jean-Alix David, Lukas Rummelhard, Amaury Nègre, Christian Laugier, Nicolas Vignard, Gabriel Othmezouri

Integration of ADAS algorithm in a Vehicle Prototype

Jerome Lussereau[1], Procópio Stein[1], Jean-Alix David[1], Lukas Rummelhard[1], Amaury Nègre[2], Christian Laugier[1], Nicolas Vignard[3], Gabriel Othmezouri[3]

***Abstract*— For several years, INRIA and Toyota Europe have been working together in the development of algorithms directed to ADAS. This paper will describe the main results of this successful joint project, applied to a prototype vehicle equipped with several sensors.**

This work will detail the framework, steps taken and motivation behind the developed technologies, as well as address the requirements needed for the automobile industry.

I. Introduction

In the scope of a collaboration project of INRIA Rhône-Alpes and Toyota Europe, a prototype vehicle have been upgraded to serve as a test platform for the development and integration of ADAS technologies, that could be transferred to the industry and find application in real-life situations.

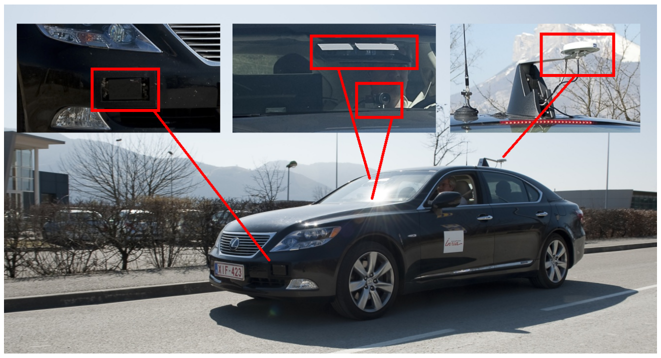

Fig. 1: The Lexus LS600h test plaftorm vehicle with its sensors highlighted

Figure 1 shows this vehicle, and its main sensors: two Ibeo Lux LIDARs installed in the front, a high-resolution camera and a stereocamera installed behind the windshield and a GPS receiver and antenna. Coupled with the GPS receiver, there is an IMU, which implements the integration of several sensors with the GPS output to provide precise measurements of the vehicle dynamics.

Besides the installed sensors, a connection to the CAN bus of the platform provides access to other measurements, as the steering wheel angle, pedals positions, turn lights, wiper activation and several other internal functions of the vehicle that are useful for understanding what are the driver's intentions in a given moment.

This ”intelligent” vehicle perceives its immediate environment using data from all onboard sensors. The purpose of the perception is to interpret the observed scene at different levels. The lowest level exploits sensor data to build a simplified environment representation, for example using a grid. At a higher level, it is possible to compute a decomposition of the scene as a set of objects, associated with information of position, speed, trajectory or class of object (car, pedestrian, bicycle, etc.). The major difficulty lies in the fact that observations of the environment are partial and imperfect within a moving scene. This ”dynamic” aspect brings strong constraints in terms of computing time.

Algorithms are used to understand the scene around the vehicle. As the algorithm interface is always changing, the design of the platform is really important. For example, it is preferable to decouple algorithms from each other to reduce dependency. A key point is to design reusable and testable algorithms so that research project could be accelerate and project collaboration could be improved.

The main objective of the developed algorithms is to assist the driver not only in everyday situations, but also in unexpected and difficult cases. In order to do so, the developed technologies must be able to provide very fast responses, in order to leave enough time for the driver to take action.

The high speed processing is accomplished by creating codes specifically for GPU accelerated computing. In such framework, it is possible to parallelize calculations, greatly improving the speed of an algorithm. The hardware used in the work the parallel computing platform, Nvidia GeForce Titan Black card with 2880 cores that can be used simultaneously.

This paper starts with a description of the deployment of ROS architecture as a middleware put in place for the integration of the different systems of our vehicle in section II. After that, three of the most important algorithms integrated with our vehicle are presented. The lane detection and departure system at section III; the visual-based localization at section IV, explaining how to correct GPS measurements based on road detection and map matching; and the fusion of different sensors into an occupancy grid for risk assessment at section V. Finally we will conclude with the perspectives and future developments of this integration project.

[1] Institut National de Recherche en Informatique et Automatique, INRIA Rhône-Alpes, France {jerome.lussereau, procopio.silveira-stein, jean-alix.david, lukas.rummelhard, christian.laugier}@inria.fr

[2] LIG/CNRS, France amaury.negre@imag.fr

[3] Toyota Motor Europe Technical Centre, Belgium {Gabriel.Othmezouri, nicolas.vignard}@toyota-europe.com

II. ADAS Algorithm Integration

An algorithm is a piece of software that takes inputs, processes them and provides outputs. In order to design a simple architecture of algorithms or modules, it is important to reduce dependencies.

The first level is the sensor part. Usually, it is the only level where modules (or drivers) call directly the Software Development Kit (SDK) or the Application Programming Interface (API) of sensor to a get data. In the other level, modules have to be independent, instead of calling directly the interface, it is better to exchange data. The exchanged data could be images, grids, bounding boxes, etc. Once modules only use data type as inputs, it is easy to use modules with online or offline data. In addition to the exchanged data, it is useful to have a unified time stamping of the data. This feature allows algorithms to compare the freshness and synchronize several types of data. Furthermore, the design should allow the interchangeability of modules so that modules could be compared or exchanged with partners.

In order to do this design, it is easier to use a middleware. In general, a middleware provides a way to connect modules, send/receive and time stamp data. The Fig. 2 show you typical demonstrator architecture by using a middleware.

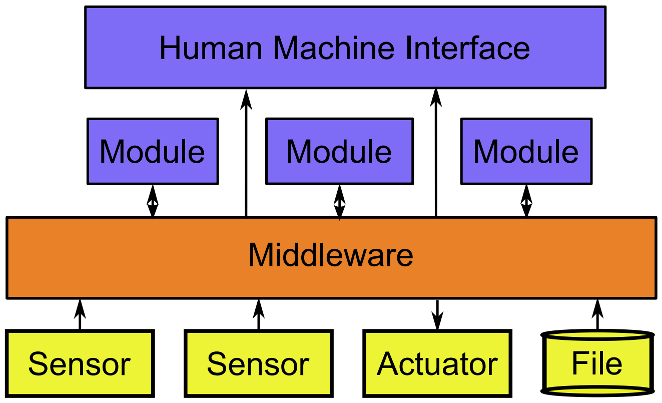

Fig. 2: Typical demonstrator architecture

At Inria, the team CHROMA uses the Robot Operating System (ROS) middleware in a demonstrator [1]. It is used to help researcher to reuse modules, standardize data types and exchange data between modules. By using ROS, the team can benefit from an active community which has already created several modules or drivers.

A demonstrator is an experimental vehicle which contains several sensor types (cameras, radars, LIDARs, etc.). The goal of the INRIA and TOYOTA demonstrator is to test/validate algorithms in real time, acquire special events while driving and demonstrate algorithms functionalities. So it is crucial to have the same algorithm processing online data or offline data without changing anything.

To facilitate the use and the maintenance of the demonstrator, it is necessary to set up a clear architecture, simple and well cut:

- to design each driver in a ROS package (one message type per sensor);
- to implement simple processing modules, performing a single task each;
- to create a ROS package by application to simplify.

By using individual ROS packages, it is possible to select which modules to use. For example, the Fig. 3 illustrates the output of a LIDAR, a camera, an Occupancy Grids (OG) and an algorithm which computes the Time to Collision (TTC) of the first obstacle.

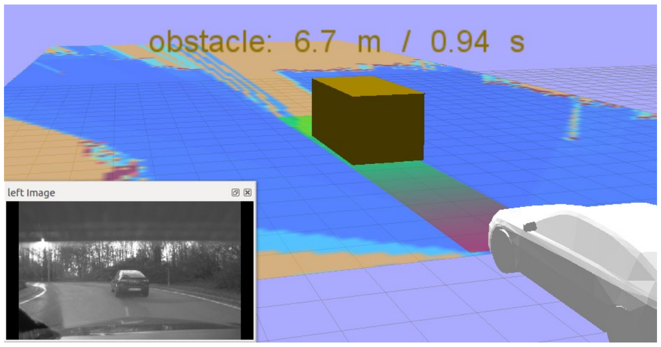

Fig. 3: Visualization of the time to collision and the first perceived object.

To develop our algorithms, it is important to be able to work on real data. ROS provides easy tools to register real time-stamped data, which can be used to work offline in the development step, allowing the evaluation of separate modules individually.

With modularity an agile method of development has been adopted. This approach allows the test of multiple solution and to compare results in order to select the best technique to be applied to a given problem.

Recorded data helps to categorize tests and classify what situation they represented. This step is critical because it allows to better choose what sensor to use according to the situation. For example, the forward facing camera's field of view is not wide enough to detect the highway boundaries. But merging data from the LIDAR helps to define where are the boundaries, so the position of the vehicle inside the highway can be better calculated.

For the sake of evaluation, it is also important to have a precise ground-truth dataset. Again, the possibility of easily working with registered data allows the creation of tools to manually annotate information, which can be used to evaluate the efficiency of the developed algorithms.

After a module have been tested with offline data and satisfactory results are obtained, it is integrated in the vehicle's system. This permits testing its capacity to properly work together with other modules in online situations and evaluate the additional resources required for a smooth operation (cpu load, network load, etc.).

III. Lane Detection and Departure

Lane detection is one of the challenging problem for ADAS. It give a model of the lane and the position of the vehicle inside the lane, allowing alerts to be generated in the case the vehicle inadvertently departure from a lane.

The aims of this lane tracker is to first detect two or more white lines on the road and then define the position of the vehicle inside the lane.

The white lines tracking is done with a particle filter implemented for parallelized computation. Then in the road plane, the distance between the center of the vehicle and the left white line is measured to give the information of position inside the lane.

a) Lane Tracker inputs: The lane tracker uses three inputs: the image of the camera, the camera intrinsic parameters and the transformation between the road and the camera frame.

b) Lane Tracker outputs: As a result of the detection, the tracker will give an output that is a model of the lane with five parameters: road width, road curvature, variation of road curvature, lateral position of the vehicle and orientation of the vehicle with respect to the lane. These parameters are illustrated in Figure 4.

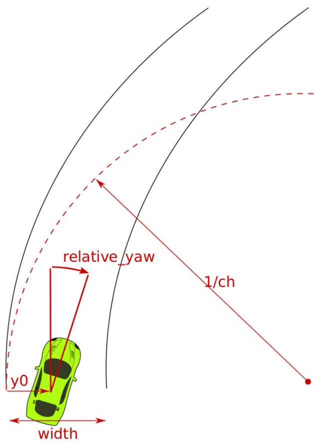

Fig. 4: Lane Tracker Model for a particle

c) Lane Tracker principles: The developed lane tracker is build onto two cores: an evaluator that return the chance of a pixel in the picture to be a white-line, and a particle filter that performs the tracking of the line, to estimate its future position. The aim of this filter is to improve the evaluation of a pixel pertinence to a white-line. It allows noisy measurements to be corrected, based on the expected position of each line in the following instants. This particle filter follows five steps, presented in Fig 5.

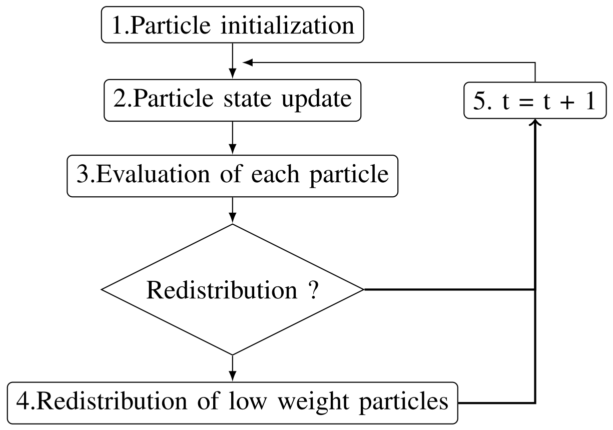
1.Particle initialization
2.Particle state update
3.Evaluation of each particle
Redistribution ?
4.Redistribution of low weight particles
5. t = t + 1

Fig. 5: Particle filter steps

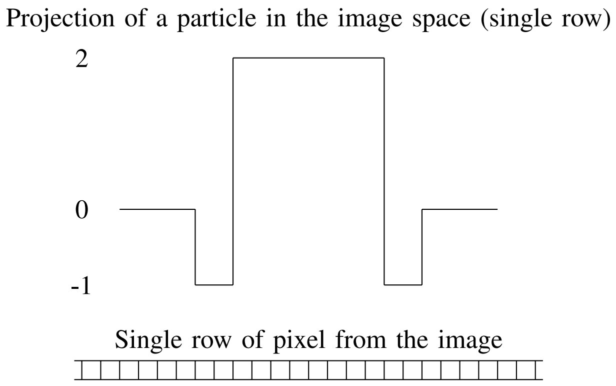

Fig. 6: Convolution input, showing a single row of a particle projection in the image space and a single row of pixel from the current image.

Each particle is a representation of a potential lane, so each particle is evaluated to assess how far it is from the actual lane in the image. To do this, a convolution of the current image (converted to grayscale) and the projection of the studied particle is performed. Fig. 6 illustrates the inputs for the convolution operation.

This convolution will result in a new image where matching white lines have a high value, and the sum of all pixel will be the score of the particle. At the same time, by correctly managing the scores of each particle, it is possible to detect not only plain white lines but also dashed white lines.

In order to check the performance of this module, 3 tools are used. One is designed to save the data (image and lane tracker data), another is used for manually marking picture with splines data [2], and the final one merges the saved data and the hand marked data to compared each line of the lane tracker data with each spline of the ground-truth.

The result is a matrix of mean pixel distances between ground-truth and computed lines. An Hungarian matching algorithm [3] to calculate the lowest cost matching solution. Finally, for a better evaluation of the pixel distance measurements, three zones are defined in the image, corresponding to a short, medium and long distances from the car.

Figure 7 illustrates this process. It is the evaluation of lane tracker output at one instant. It gives a visual and an accurate measurement of the mean pixel distances between green and red lines in each of the three zones (delimited with blue lines).

IV. Visual Map-Based Localisation

Given the lack of affordable GPS for precise localization, it is necessary to implement new ways to improve localization for vehicles with low cost sensors. Here a new localization method, which uses a geographic map and a camera, is presented. The main point of this method is to combine sensor readings and known data about the environment to

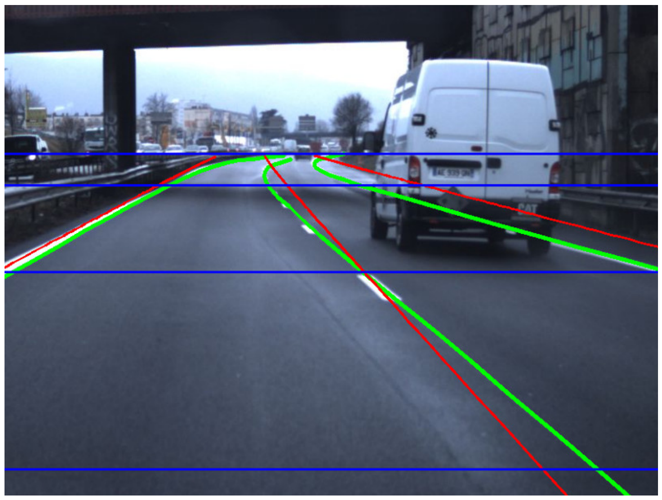
Fig. 7: Lane Tracker evaluation: red are lane tracker lines, green are manual marked lines, and blue are range evaluation zones.

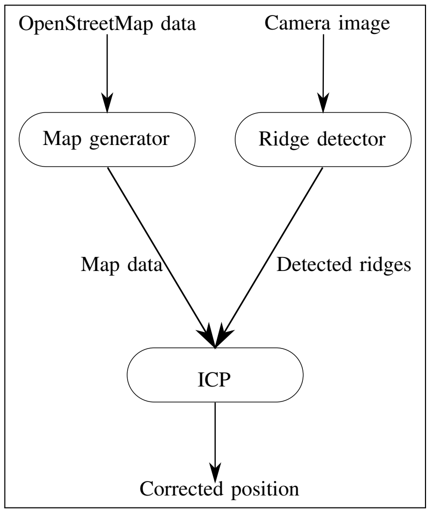

Fig. 8: The three parts of the approach: map generation, ridge detection and comparison with ICP algorithm are articuled as shown by this figure.

perform localization. Lane markings are detected using a camera, and then the extracted lines are compared to the ones stored in the map using ICP algorithm. Fig. 8 shows the organization of the approach.

A. *The map*

To construct a map in the vehicle environment, data from OpenStreetMap (OSM) [4] was used, specially because it is a free and opensource database. The OSM provides information on the roads and lanes, but there is no information about lane markings. Thus, to obtain that information, lines were generated semi-automaticaly, given roads and number and width of the lanes. This step is done offline and the new data are stored on the vehicle's onboard computer. Then requests can be send to this database to fetch local map data while driving. Fig. 9 shows the results of the data conversion

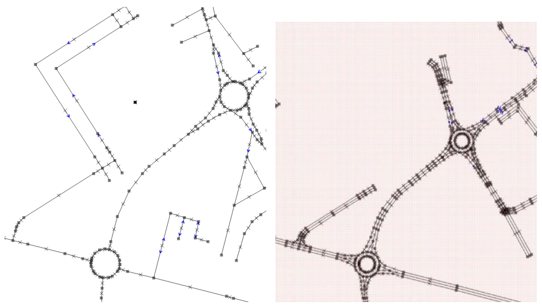
Fig. 9: Data conversion. (a) Raw OSM data, a line represents a road and it is not possible to see the lanes. (b) Modified OSM data, with lane markings.

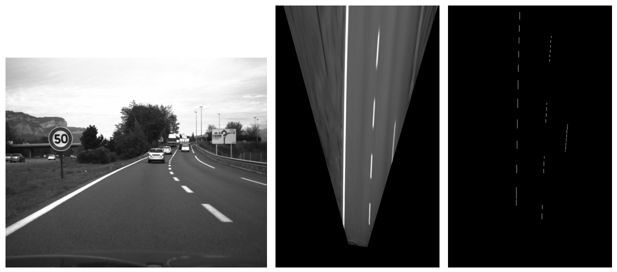
Fig. 10: Ridges detection. (a) Input image. (b) Projected image. (c) Detected ridges.

process.

B. *Line detection*

The line extraction is done using ridge detection on a top-down view of the camera image. Only one monocular camera is used, as it is an inexpensive sensor, and needs only to be calibrated once. The line detection is based on an algorithm using laplacian to extract ridges of the monochrome image. The algorithm is implemented for parallelized calculation using CUDA on a GPU, for an improved performance. Figure 10 shows the results of the ridge detector.

C. *Matching by ICP algorithm*

These data are compared using an ICP algorithm to obtain a better localization of the vehicle. The ICP is an iterative algorithm which minimizes the sum of the distances between the points detected as ridges and the segments of line extracted from the map. Each iteration is composed of two step:

1) the matching of the lines, each point is matched to the closest segment.
2) the computation of the corrected position using Levenberg-Marquardt algorithm to minimize the sum of errors.

On Fig. 11 it is possible to see the results of the ICP in a highway scenario. It takes place on a two-lane road before it merges with another two-lane road. In the lower right corner it is possible to see the view of the camera, and thus that the vehicle is on the rightmost lane. The green lines correspond to the lines of the map. The red dots correspond to the detected lines. The red arrow corresponds to the position

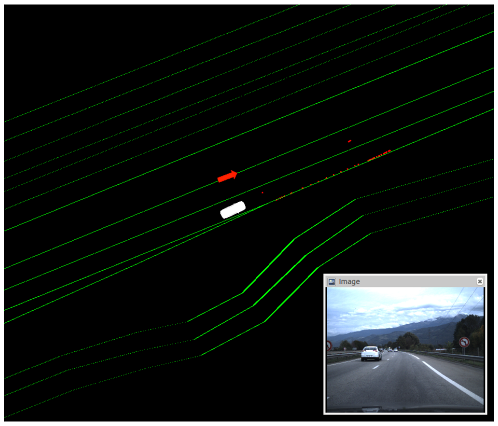

Fig. 11: ICP correction on highway

given by the GPS, which puts the car on the left of the leftmost lane. The white car corresponds to the corrected position, which puts the car in the middle of the rightmost lane, which is the correct position.

The results are very promising on highways, but the algorithm has a lower performance on other types of roads, mostly due to irregularities.

V. Sensor Fusion using an Occupancy Grid

One of the central development of this work is a low-level representation of the surrounding environment, which may serve as a building base for the development of additional high-level modules. Such representation is accomplished with the use of OG [5], which allows the implementation of a model of the environment that encompass both static and dynamic features surrounding the vehicle.

An advantage of the proposed approach is that it is sensor independent, being able to work with and fuse information from different sources, as LIDAR, stereocamera, infrared, RADAR, and so on [6], [7].

In the current platform, the data source used for the modeling of the vehicle surrounding comes from two LIDARs installed in the front of the vehicle. Each one of these LIDARs have four measurement layers, which are used to create individual OGs. The resulting eight OG are then fused into a single one, to be later filtered and enhanced using the Hybrid Sampling Bayesian Occupancy Filter (HSBOF) [8], [9].

Such filter provides an efficient manner of representing static and dynamic features of the environment around the experimental platform. The main strengths of the algorithm rest on :

- the distinction between static and dynamic grid cells;
- the use of classic grids in motionless areas, and particles to model the motion of dynamic areas.

These two features greatly reduce the requirement of computer resources and at the same time, improve the accuracy of the models. Fig. 12 illustrates this model.

Examples of the use of the HSBOF are shown in Fig. 13, with tests performed both in urban and highway environment.

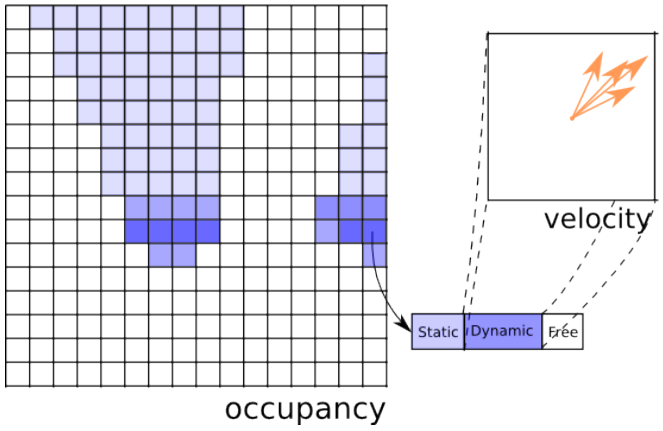

Fig. 12: Proposed representation : a 2 dimension grid, to each cell are assigned an occupancy value, a static coefficient and a set of particles

With a dynamic representation, it is possible to compute the probability of the **risk of collision** between the experimental platform and any static or dynamic element of the environment modeled by the HSBOF [10], [9]. To perform this computation, a future projection of the vehicle and of the occupancy grid is created. Such projection is performed for several small timesteps, where a list of all collisions between particles and the vehicle projection are stored in a list, as illustrated in Fig. 14.

With this list, it is possible to query the algorithm for the probability of collision during different time windows, which permit distinct alerts to be given to the driver, according to the TTC. For example, in a longer time horizon the algorithm can generate an alert and in the case of an imminent collision, the algorithm can activate the brake and pre-crash systems.

Another important advantage of this approach is that not only it is possible to predict when a collision will happen, but also where it will take place in the map, and from where in the environment the risk is coming. This greatly increases the type of alerts to be generated and the treatment possible with such information.

Pedestrian collisions have been tested with a soft human-sized crash test dummy, which allowed to validate the accuracy of the risk detection, the TTC and the probability associated with it. The HSBOF presented substantial results, whether it be for grid occupancy construction, motion estimation or collision risk assessment. Figure 15 shows an example of a real test of the risk assessment system, already integrated into the vehicle's ecosystem.

VI. CONCLUSIONS

During the last five years, Toyota and INRIA have been working together on the project presented in this work. This partnership have produce several important results, among articles, thesis subjects and also a patent on predictive technologies. The HSBOF algorithm, that represents the base layer of development, is already integrated in the vehicle and providing very good results, specially regarding the risk assessment.

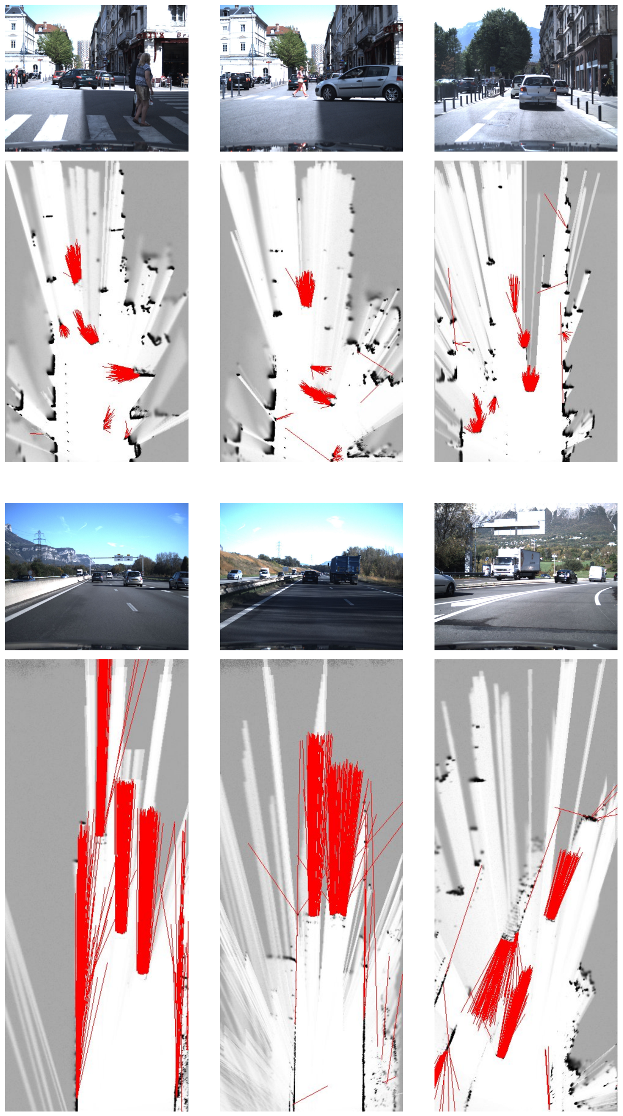

Fig. 13: Resulting occupancy grid and velocity field on different urban and highway situations. Black cells represent the occupied space and red lines represent the average velocity vector for cell with a high dynamic probability.

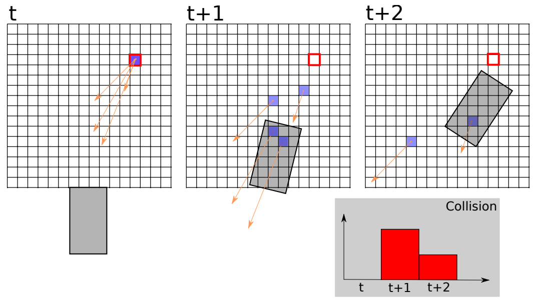

Fig. 14: Collision risk estimation over time for a specific cell. The cell position is predicted according to its velocity, along with the mobile robot. This risk profile is computed for every cell, and then used to integrate over time the global collision risk.

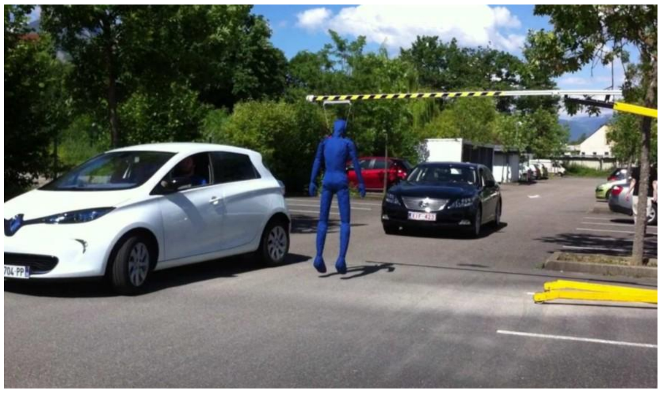

Fig. 15: Real test example of collision detection. The white car blocks the view of a pedestrian (blue) until the last instant when it swerves to the right. The collision detection algorithm installed in the Lexus platform generated alerts about an imminent collision for the driver.

Recently new performance tests have been conducted using the HSBOF on an NVIDIA Jetson Pro card, which performs parallel calculation. The advantage of using this hardware is its reduced size and power consumption. These are important requirements for a future development as an off-the-shelve solution for automotive perception.

A future objective of our team is to focus on the incorporation of the object concept at the HSBOF, in order to reach higher level models of the environment. The classification of an object as a bicycle, a car or a pedestrian, for example, allows a better prediction and behavior assessment. This improves the knowledge of the vehicle surrounding and enhances the assistance provided to the driver.

References

[1] Morgan Quigley, Brian Gerkeyy, Ken Conleyy, Josh Fausty, Tully Footey, Jeremy Leibsz, Eric Bergery, Rob Wheelery, and Andrew Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[2] M. Aly, "Caltech Lanes Dataset," April 2015. [Online]. Available: http://vision.caltech.edu/malaa/datasets/caltech-lanes/

[3] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[4] "OpenStreetMap," April 2015. [Online]. Available: https://www.openstreetmap.org

[5] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[6] J.-D. Yoder, M. Perrollaz, I. Paromtchik, Y. Mao, and C. Laugier, "Experiments in Vision-Laser Fusion using the Bayesian Occupancy Filter," in *International Symposium on Experimental Robotics*, Delhi, India, Dec. 2010.

[7] Q. Baig, M. Perrollaz, J. Botelho, and C. Laugier, "Fast classification of static and dynamic environment for Bayesian Occupancy Filter (BOF)," in *IROS12 4th Workshop on Planning, Perception and Navigation for Intelligent Vehicles*, Villamoura, Portugal, June 2012.

[8] A. Nègre, L. Rummelhard, and C. Laugier, "Hybrid Sampling Bayesian Occupancy Filter," in *IEEE Intelligent Vehicles Symposium (IV)*, Dearborn, United States, June 2014.

[9] L. Rummelhard, A. Nègre, M. Perrollaz, and C. Laugier, "Probabilistic Grid-based Collision Risk Prediction for Driving Application," in *ISER*, Marrakech/Essaouira, Morocco, June 2014.

[10] Q. Baig, M. Perrollaz, and C. Laugier, "Advances in the Bayesian Occupancy Filter framework using robust motion detection technique for dynamic environment monitoring," *IEEE Robotics and Automation Magazine*, Mar. 2014.