Environment Perception Architecture using Images and 3D Data

Horaţiu FLOREA, Robert VARGA, Sergiu NEDEVSCHI
Technical University of Cluj-Napoca, Computer Science Department
{horatiu.florea, robert.varga, sergiu.nedevschi}@cs.utcluj.ro

***Abstract*—This paper discusses the architecture of an environment perception system for autonomous vehicles. The modules of the system are described briefly and we focus on important changes in the architecture that enable: decoupling of data acquisition from data processing; synchronous data processing; parallel computation on GPU and multiple CPU cores; efficient data passing using pointers; adaptive architecture capable of working with different number of sensors. The experimental results compare execution times before and after the proposed optimizations. We achieve a 10 Hz frame rate for an object detection system working with 4 cameras and 4 LIDAR point clouds.**

I. Introduction

As the world prepares for the first stages of the public adoption of automated vehicles, stated to occur in the beginning of the 2020s [1], there is an expanding need for research towards comprehensive, redundant and accurate software solutions for environment perception. Their development is linked with the potential of bringing forward both significant improvements to the quality of life, as well as new challenges for the global economy [2].

In order to provide the best description of a vehicle's surrounding environment the system must acquire, process and summarize information covering the entire area around the car, captured with a multitude of sensing devices, offering measurements that complement each other, in terms of range, detected features, etc. Redundancy also stems from the use of a wide array of devices, which must be coupled with the ability of the system to provide stable results, even at a reduced fidelity, in the case of one or more partial or complete failures.

To this end, visible light cameras provide rich information about the surrounding environment which enables tasks such as pixel-level segmentation, where each pixel is assigned to an object class based on a variety of factors. Besides issues such as varying illumination conditions and occlusions, their principal downside is that without additional, often computationally expensive processing, the output data is bi-dimensional, which is insufficient for an autonomous-grade perception system.

To overcome this limitation, Light Detection And Ranging (LIDAR) technology has been popularized in recent years, partially due to significant reductions in the cost of the devices. LIDAR scanners operate by firing non-visible LASERs and measuring the time-of-flight of the reflected beam, thus creating a 3D scan of the environment. Some sensors have moving scan heads which capture a 360° view of its surroundings. The output of such devices is commonly referred to as a 3D Point Cloud or scan, and is comprised of a number of points, often expressed using Cartesian coordinates, with some additional information such as reflectance.

II. Related works

Before availability of cost-effective LIDAR technology became available, most solutions, such as the one presented in [3], relied on stereovision approaches for reconstructing 3D information from the scene captured with multiple cameras. This process requires significant computing resources or dedicated hardware in order to achieve accurate results.

One of the most successful stereo-based systems, detailed in [4], used a stixel-based representation for reconstructed 3D data, which, alongside monocular cameras, RADARs and a high accuracy digital map achieved performance levels that allowed the researchers to complete a 103km journey with complex scenarios.

In [5], the authors present an algorithm for grid-based environment particularly suited for urban areas. Here, they use a ground plane estimation technique to differentiate points scanned by a high density LIDAR device into ground and obstacle points. A 2.5D stixel representation enables further classification of curbs and scene objects.

The authors of [6] highlight the important role the design of a system's architecture plays in terms of the overall robustness when coping with world uncertainties. The article presents a summary and investigation of several existing architectures. In [7] and [8] a distributed system architecture for autonomous vehicles, based on the AUTOSAR standard is presented in detail, including the assignment and interaction mechanisms of the component sub-systems.

The remainder of the paper will go over the operation of each of the modules that comprise our proposed perception system. For more technical details about the system and the processing modules the reader should consult the original publication from [9]. In this paper we focus on improvements to the architecture and execution time optimizations. Section IV provides performance benchmarks of our system, before section V, which summarizes our work.

III. Proposed architecture

Figure 1 illustrates the overall architecture of the proposed system. It can be observed that following the data synchronization and standardization module (*Flow Manager*, left) the

978-1-5386-8445-0/18/$31.00 ©2018 IEEE

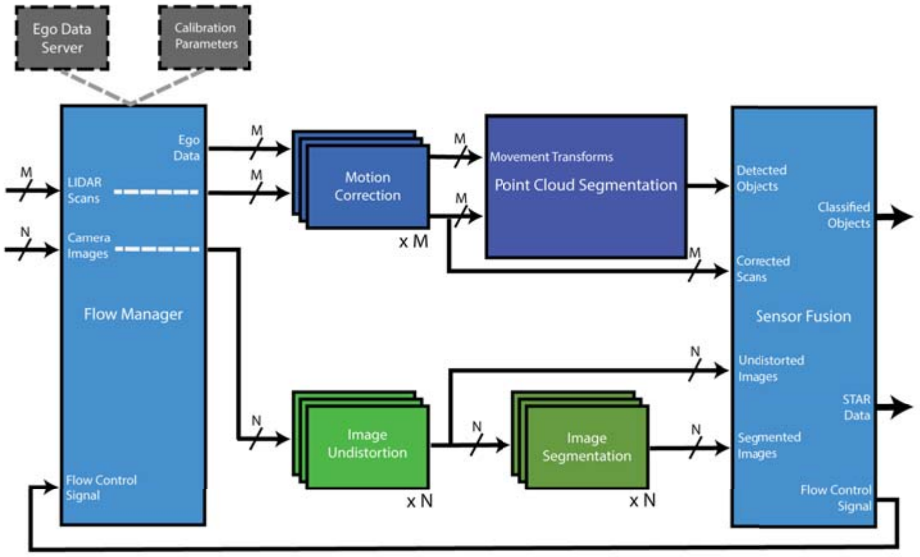

Fig. 1. Overview of proposed system

system can be differentiated into two separate processing sub-chains: one for processing 3D LIDAR measurements acquired by *M* sources (top, blue) and one for 2D image data, captured by *N* cameras (bottom, green). Most of the modules operate on measurements originating from a single sensor and need to be multiplied accordingly.

The intermediary results from both sequences is finally aggregated in the final *Sensor Fusion* module which outputs a list of classified objects along with a fused 3D cloud, composed of points enhanced with semantic and appearance information. Auxiliary information such as the movements of the ego-vehicle and the calibration parameters for the sensing devices is aggregated by the first module and linked to the data before initiating a processing cycle.

In keeping with the overarching view for the implemented perception system, which mandates that all sensory information is delivered synchronously for processing, there is an explicit need to ensure this at a software-based level, as the complex data collecting system cannot offer this guarantee. To meet this requirement, the Flow Manager aggregates data from all available sensors into so called **data batches**. Each sensor measurement can be aggregated into at most one data batch, or it can be dropped completely if the overall batch does not satisfy some imposed timing constraints.

One of the most important design decisions taken towards optimizing the performance of the entire processing chain as a whole was using only data pointers as means of inter-module communication. This approach comes with obvious benefits, as large data, such as 3D scans or images are not copied when being passed between modules, but also requires a globally consistent set of rules in terms of data allocation/de-allocation. To this end, we have opted for a strategy where the module introducing new data into the processing chain, be it raw measurements or intermediate results, is solely responsible for clearing the memory when appropriate.

As the computing hardware available for both offline development and online testing is fitted with a dedicated Graphics Processing Unit (GPU) beside the multi-core CPU, some of the modules along the image processing sub-chain, namely Image Undistortion and Image Segmentation, are designed to exploit this resource. Currently, the tasks carried out by them are optimized for nVidia GPUs using the CUDA API. As copying the input data from the main system memory to the internal GPU memory is a potential performance bottleneck, this is done only by the first module in the GPU sub-chain, which then passes along the pointers for GPU memory.

The rest of the processing is done on the CPU, which further contributes towards true parallelism, as the computations are done on separate devices. 3D Point cloud processing can be also migrated to the GPU, but since it would make that singular device a resource contention point while the CPU would be under-utilized, we have opted for the current approach. For increasing the utilization of the processor, some of the processing modules feature code sections paralleled using the OpenMP API.

Figure 2 presents the most frequently encountered sequence of processing steps for a data batch. After all inputs arrive at the Flow Manager, each LIDAR scan is motion corrected sequentially, while, in parallel on the GPU, each image undergoes Image Undistortion followed by Segmentation. Once all clouds are corrected, the Point Cloud segmentation process can start. Moreover, at this point data fusion can take place for the images for which the segmentation was completed. Finally, objects can be assigned classes after all images were included in the fusion process.

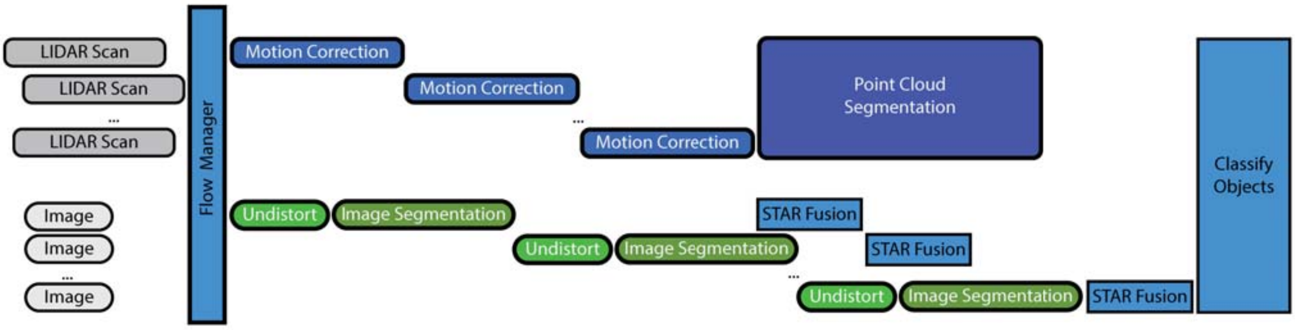

Fig. 2. Typical timeline of an execution cycle of the processing chain (time progresses left to right)

A. Data input constraints

Flexibility of our implementation was always a factor throughout the development process, yet achieving high computational performance often imposes some constraints. The inputs to our perception system are provided by a proprietary data acquisition system which meets our requirements. The processing chain does not feature hard coded sensor arrangements, and can be initialized at runtime with a varying number of measurement devices, for each of which calibration data is provided.

There must always be at least one camera and one 3D scanner available. The module that interfaces with the acquisition system collects and attaches the relevant intrinsic and extrinsic calibration information to the data batches. In terms of image data, the acquisition system guarantees that only valid, complete and in-order sets of images from all available detectors are sent for processing, and that frames within a set were captured virtually synchronously. LIDAR data provided is ensured to be timestamped at an individual point level, and that the clocks of the scanners are in-sync with the processing PCs. Moreover, the detectors' scanning motion is roughly in sync.

B. Flow Manager

The principal mechanism for controlling the operation of the proposed processing chain resides in the data flow management module, referred to as the Flow Manager (FM). It serves multiple purposes, with the main goal of aligning the incoming data's structure to the one employed by the optimized chain.

The main functionalities of Flow Manager are:

- Create data-batches - collect and match input from multiple sensors
- Data synchronization - call other modules in a synchronous manner, regardless of asynchronous data acquisition
- Decoupling of data acquisition and data processing - separate threads for listening to new data and data processing

Creating the aforementioned synchronized data batches, which contain measurements from all connected sensors, implicitly requires some form of data buffering, in order to store and aggregate incoming samples. This approach can introduce delays and computing overhead, however, we have determined that their magnitudes are acceptable in the context of our solution. LIDAR scans are stored in a per sensor buffer with a hard coded depth of one, in order refrain from processing too old data.

In actuality, for 3D data, the module keeps three buffers for each detector: an *accumulation buffer* where new data is always placed, a multi-detector *consistent buffer* which contains temporally coherent scans from all the sensors meaning the time difference between the first and last acquired scan is less than an imposed threshold, Th^{LIDAR}_{LIDAR} (eq. 1) and a final *in-use buffer* which holds the *consistent buffer* incorporated in the data batch currently being processed. Scans which are not placed in the consistent buffer are eventually discarded.

$$Th^{LIDAR}_{LIDAR} \geq max\left(\left|ts\left(scan_i\right) - ts\left(scan_j\right)\right|\right), \quad \forall i \leq M, j \leq M, i \neq j \tag{1}$$

Before storing incoming point clouds into the accumulation buffer, we also perform a pre-processing step by converting the received data structure, in order to keep only valid return measurements - 3D points for which the scanner's receiver actually registered the reflected laser beam. This step has less to do with memory constraints, but instead it simplifies and speeds-up further procedures.

Handling image data is done using a similar scheme, which takes into account the higher (3x) throughput. For this, the *consistent buffer* is replaced with a *image queue* structure, which holds a number of temporally coherent sets of images, added each time all the *accumulation buffers* associated to active cameras receive new frames. As previously mentioned, our current system works under the assumption the cameras are hardware synchronized, and, as such, a set containing a frame from each active camera is intrinsically assumed temporally coherent.

Preparing a new data batch for the processing chain consists of searching the *image queue* for the first set that was captured after the LIDAR reference point, within a given time limit, Th^{LIDAR}_{CAM} (eq. 2). Here, we do not use the absolute time difference, as the ordering of the events is relevant. As the queue is time ordered, this procedure is fast, with too old image sets being marked for removal. The reference point is selected as either the start or the end timestamps from one of the connected scanners, this being a user selectable property.

$$Th_{CAM}^{LIDAR} \geq ts\,(camera) - ts\,(scan_i) \geq 0, i \leq M, \quad (2)$$

The chosen image set's timestamp also functions as the batch's *master timestamp*, which will be used for tagging the final output. The Motion Correction modules use this information for temporally aligning 3D data to the images, for which they require ego-motion data, also aggregated by the FM upon batch creation.

1) Operation modes: The Flow Manager module supports three types of operations: single threaded, multi-threaded and signal-based.

- *Single threaded* use consists of sequentially sending the data batch's components to the appropriate processing modules, on a single thread, belonging to the data acquisition system. This means it is a blocking operation, which can be useful in the context of offline data exporting, but it can severely affect the online performance of the system.
- *Multi-threaded* operation is much better-suited for online operation, as only the buffering and data batch aggregation operations take place on the acquisition system's threads. Processing is split on two additional threads, one for 2D image processing and one of 3D data processing. This decoupling is one of the key elements contributing to the overall system's performance, including in terms of flexibility of running on lower-spec computing hardware, where potential data batches are inherently dropped until processing finishes.
- While in multi-threaded mode the generation of a new data batch can only be triggered by the formation of a new LIDAR *consistent buffer*, the *signal-based* augmentation can increase the overall throughput by triggering a new batch's creation when the previous one finished processing. This mechanism requires a signal to be passed back from the last processing module in the chain, in order to check whether a valid, non-processed *consistent buffer* is available. If not, default multi-threaded operation resumes.

C. Image processing modules

1) Image Undistortion: This module is responsible for undistorting the fisheye images from the cameras. We provide only an overview of the method used and show improvements to the implementation.

A camera model proposed by Geyer [10] and Cristopher Mei [11] is used. Cameras are calibrated using the Kalibr tool [12]. Extrinsic calibration is performed using markers and a special calibration room. Undistortion is performed by creating a virtual surface in front of the camera, sampling points from this surface with a given resolution, projecting the points onto the original image and interpolating the color values. Mainly two types of projection are possible: planar (projective) and cylindrical. We have found that the cylindrical projection offers a high horizontal field of view while introducing a limited amount of distortion.

The current implementation runs on the GPU. A lookup table is generated which specifies the position to which each of the surface points are projected. This table needs to be generated only once and during the online processing a single pass over the image is performed, the stored positions are retrieved and the color values are interpolated using bilinear interpolation. The resulting image is kept in the GPU memory and sent to the next module for segmentation.

2) Image Segmentation: The image segmentation operates on the undistorted fisheye image and provides semantic class information at pixel level. A CUDA implementation of the ERFNet [13] is used for this task. The classifier weights were tuned on our own dataset with labeled images.

Optimizations performed for this module include: reducing the image resolution to half along the rows only, or both along the rows and columns; receiving the input image directly in the GPU memory from Image Undistortion; paralellizing post-processing operations which transfer the resulting image from the GPU memory to the host memory and constructing the semantic label image; avoiding constructing the color label image when not necessary.

D. Point cloud processing modules

1) Motion Correction: The Motion Correction module addresses the intrinsic distortion introduced in LIDAR 3D scans when the platform to which the scanners are rigidly mounted is moving. Of course, without feedback from a detection and tracking solution regarding moving targets, only distortions due to the ego-movement can be handled. Our current approach is a slightly updated version of the one presented in [9].

For correcting a point cloud acquired over a time period to the master timestamp of the data batch, 6-Degrees of Freedom ego-motion data is required. Firstly, all the scanned points are individually transformed to a single time instance representation (that of the end of acquisition), based on their respective timestamps, after which a common transformation is applied to all points in order to bring them to the time instance of the image acquisition (the batch master timestamp). The movement undertaken by the ego-vehicle between the end of the scan and the master timestamp is also outputted to the Point cloud-based object detection module.

2) Point Cloud Segmentation: The corrected point clouds are fused and a Digital Elevation Map is created. Object cuboids are extracted based on a version of the approach presented in [14] adapted to LIDAR point clouds. The cuboids without object classes are sent to the fusion module where classification is performed based on the fusion between the point cloud and the semantic segmentation from the image views.

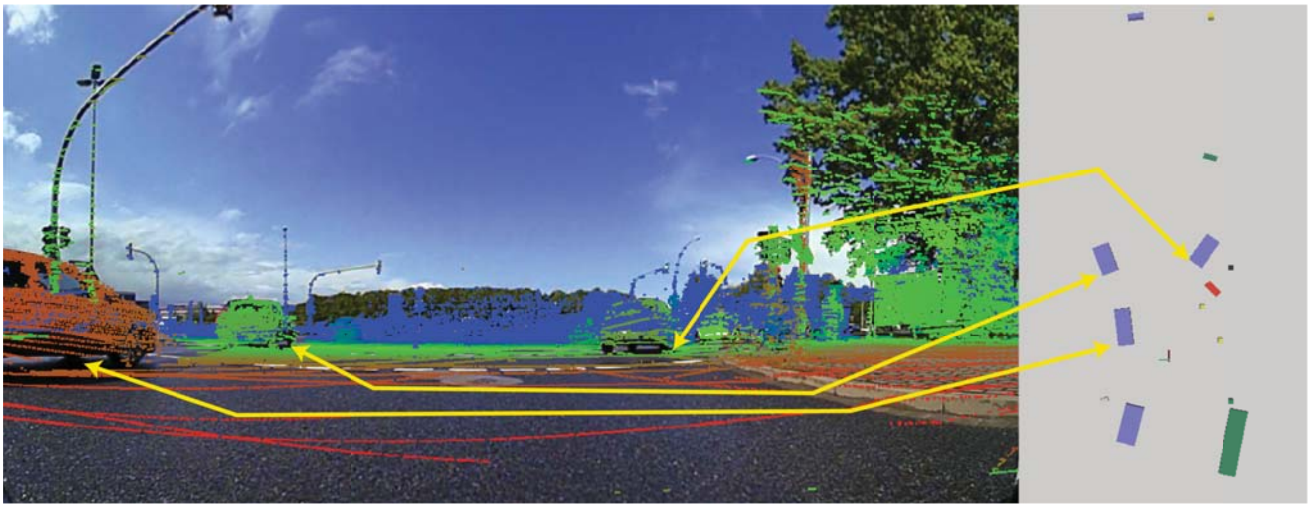

Fig. 3. Left: Motion corrected 3D points projected onto the undistorted image of the frontal view; Right: Detected and classified objects in the bird's eye view; blue rectangles represent cars; Arrows indicate correspondences between objects from the two views

E. Sensor Fusion

The sensor fusion module is responsible for generating the Spatio-Temporal Appearance-based Representation (STAR). The 3D point clouds originating from different sensors are fused together with the semantic segmentation from each camera view. Since the order in which these results are available is arbitrary, the fusion module must treat all cases in which input data can arrive.

Data is received and logged from each sensor. Data from each sensor should arrive once during a processing cycle. This is assured by the Flow Manager module. Fusion happens when all connected LIDAR point clouds have arrived and a segmentation is ready. The points from all the LIDARS are projected onto the segmented image and the points which fall inside the image will get a semantic class.

Wrong associations between points and semantic classes happen when an object visible by the LIDAR is occluded in the image view. These cases are detected and the corresponding semantic labels are marked as unreliable (occlusion handling). Points with multiple labels from different image views get the label from the view in which the projection is closer to the image center.

If all sensor input from connected modules has arrived, we perform operations which require the complete set. A signal is sent to the Flow Manager indicating the end of a processing cycle, enabling it to send the next data batch. We associate semantic classes to objects detected by the Point Cloud-based Segmentation module.

Optimizations performed for this module include: avoiding further processing during fusion when a 3D point clearly falls outside the image; caching projection coordinates; using points already transformed into common car coordinate system instead of transforming them during processing; reordering of operations to make use of previous calculations; multi-threaded implementation of the fusion operation.

IV. Experimental results

Our perception system was developed in the C++ language and integrated into an industry-standard proprietary, time-triggered framework with both data playback and live operation (on the test vehicle). As such, each of the constituent parts of the processing chain is an independent software module which is loaded into the runtime environment where they can be interconnected via the interface.

Offline testing of the perception system was carried out using recorded data sequences that accurately reproduce the flow and timing of sensory information. The hardware specification of the computer running the processing chain was: Intel Core i9-7900X CPU, 128GB Physical System Memory and a NVIDIA GeForce GTX 1080 Ti GPU.

The common vehicle setup included 4 cameras and 4 LIDAR scanners (of different capabilities). The inter-LIDAR scan timing constraint, Th_{LIDAR}^{LIDAR}, has a set value of $40ms$, while the LIDAR-camera constraint, Th_{CAM}^{LIDAR} is set to $110ms$.

Figure 3 illustrates the 3D measurements from all scanners projected onto the front camera, with coloring based on the relative distance to the vehicle (left) and the associated detected objects, colored according to their class assigned based on the semantic information (i.e. Blue - vehicles, Red - Vulnerable Users, Green - Vegetation).

Table I shows the measured execution times of relevant modules before and after the described optimizations. The time shown is an average over 500 measurements. The time required for the whole processing chain is measured in Flow Manager from the moment the data-batch is sent to the moment when the ready signal is received from the Sensor Fusion module.

TABLE I
AVERAGE EXECUTION TIMES FOR EACH MODULE

Module	before	after
Image Undistort	12.15 ms	0.97 ms
Image Segmentation	38.18 ms	19.00 ms
Motion Correction	1.52 ms	0.49 ms
Point Cloud Segmentation	-	45.00 ms
Sensor Fusion	5.64 ms	1.55 ms
whole processing chain	304 ms	100 ms

V. CONCLUSION

The architecture of the perception system has been reworked and heavily optimized. The most important changes include:

- Introduction of the Flow Manager module, which assures data synchronization, creates valid data-batches and decouples listening for new input from processing.
- Separating processing modules to CPU-based (mainly point cloud processing and fusion modules) and GPU-based (mainly image processing modules) which allows for their parallel execution.
- Working with data pointers in each module. This avoids unnecessary data copy operations which add up quickly. Also forces the owner of the data to manage its lifetime and encourages data reusage.
- Parallelization of operations performed on the CPU.
- Providing a GPU implementation for image undistortion, which allows direct access of the resulting image in the device memory.
- Lowering the image resolution for image segmentation.
- Performing point cloud filtering and transforming the point cloud to the common car reference frame as soon as possible. Later operations use this representation.

The current system can be augmented in the future in several aspects, such as adding a live operation monitor agent that can detect device failures and reconfigure the system on-the-fly, providing built-in support for the next generation of Solid State LIDAR sensors, with the ability of targeted measurements, aggregating data from other types of sensors such as short/long range RADAR. However, further performance optimization for in-car use will remain our priority.

ACKNOWLEDGMENT

This work was partially supported by the by the UP-Drive project (Automated Urban Parking and Driving), Horizon 2020 EU funded, Grant Agreement Number 688652. This work was also partially supported by a grant of Ministry of Research and Innovation, CNCS - UEFISCDI, project number PN-III-P4-ID-PCE-2016-0727, within PNCDI III.

REFERENCES

[1] T. Litman, *Autonomous vehicle implementation predictions*. Victoria Transport Policy Institute, 2017.

[2] B. Clark, G. Parkhurst, and M. Ricci, “Understanding the socioeconomic adoption scenarios for autonomous vehicles: A literature review,” 2016.

[3] S. Bota, S. Nedevschi, and M. Konig, “A framework for object detection, tracking and classification in urban traffic scenarios using stereovision,” in *Intelligent Computer Communication and Processing, 2009. ICCP 2009. IEEE 5th International Conference on*. IEEE, 2009, pp. 153–156.

[4] J. Ziegler, P. Bender, M. Schreiber, H. Lategahn, T. Strauss, C. Stiller, T. Dang, U. Franke, N. Appenrodt, C. G. Keller *et al.*, “Making bertha drivean autonomous journey on a historic route,” *IEEE Intelligent Transportation Systems Magazine*, vol. 6, no. 2, pp. 8–20, 2014.

[5] J. Rieken, R. Matthaei, and M. Maurer, “Benefits of using explicit ground-plane information for grid-based urban environment modeling,” in *Information Fusion (Fusion), 2015 18th International Conference on*. IEEE, 2015, pp. 2049–2056.

[6] Ö. Ş. Taş, F. Kuhnt, J. M. Zöllner, and C. Stiller, “Functional system architectures towards fully automated driving,” in *Intelligent Vehicles Symposium (IV), 2016 IEEE*. IEEE, 2016, pp. 304–309.

[7] K. Jo, J. Kim, D. Kim, C. Jang, and M. Sunwoo, “Development of autonomous carpart i: Distributed system architecture and development process,” *IEEE Transactions on Industrial Electronics*, vol. 61, no. 12, pp. 7131–7140, 2014.

[8] ——, “Development of autonomous carpart ii: A case study on the implementation of an autonomous driving system based on distributed architecture,” *IEEE Transactions on Industrial Electronics*, vol. 62, no. 8, pp. 5119–5132, 2015.

[9] R. Varga, A. Costea, H. Florea, I. Giosan, and S. Nedevschi, “Super-sensor for 360-degree environment perception: Point cloud segmentation using image features,” in *Intelligent Transportation Systems (ITSC), 2017 IEEE 20th International Conference on*. IEEE, 2017, pp. 1–8.

[10] C. Geyer and K. Daniilidis, “A unifying theory for central panoramic systems and practical implications,” in *European conference on computer vision*. Springer, 2000, pp. 445–461.

[11] C. Mei and P. Rives, “Single view point omnidirectional camera calibration from planar grids,” in *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, 2007, pp. 3945–3950.

[12] J. Rehder, J. Nikolic, T. Schneider, T. Hinzmann, and R. Siegwart, “Extending kalibr: Calibrating the extrinsics of multiple imus and of individual axes,” in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 4304–4311.

[13] E. Romera, J. M. Alvarez, L. M. Bergasa, and R. Arroyo, “Efficient convnet for real-time semantic segmentation,” in *Intelligent Vehicles Symposium (IV), 2017 IEEE*. IEEE, 2017, pp. 1789–1794.

[14] F. Oniga and S. Nedevschi, “Processing dense stereo data using elevation maps: Road surface, traffic isle, and obstacle detection,” *IEEE Transactions on Vehicular Technology*, vol. 59, no. 3, pp. 1172–1182, 2010.