Probabilistic Graph-based Real-Time Ground Segmentation for Urban Robotics

Iván del Pino, Angel Santamaria-Navarro, Anaís Garrell Zulueta,
Fernando Torres and Juan Andrade-Cetto

***Abstract*—Terrain analysis is of paramount importance for the safe navigation of autonomous robots. In this study, we introduce GATA, a probabilistic real-time graph-based method for segmentation and traversability analysis of point clouds. In the method, we iteratively refine the parameters of a ground plane model and identify regions imaged by a LiDAR as traversable and non-traversable. The method excels in delivering rapid, high-precision obstacle detection, surpassing existing state-of-the-art methods.**

Furthermore, our method addresses the need to distinguish between surfaces with varying traversability, such as vegetation or unpaved roads, depending on the specific application. To achieve this, we integrate a shallow neural network, which operates on features extracted from the ground model. This enhancement not only boosts performance but also maintains real-time efficiency, without the need for GPUs.

The method is rigorously evaluated using the SemanticKitti dataset and its practicality is showcased through real-world experiments with an urban last-mile delivery autonomous robot.

The code is publicly available at https://gitlab.iri.upc.edu/idelpino/iri_ground_segmentation

***Index Terms*—Ground Segmentation; Terrain Analysis; Sequential Innovation; LiDAR**

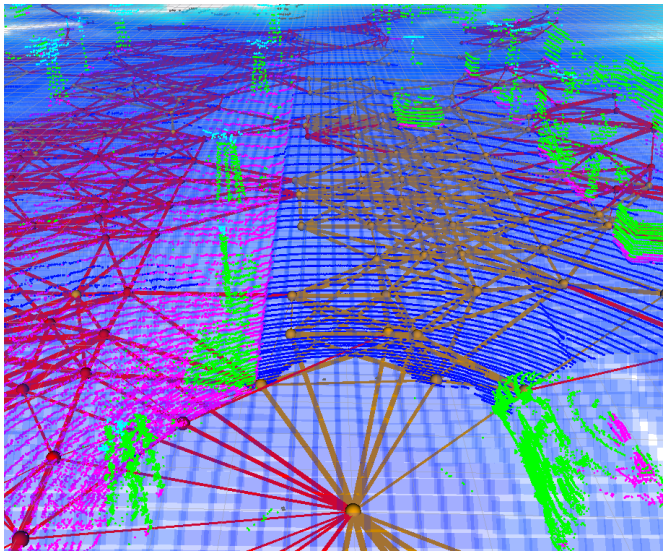

Fig. 1. Real-time terrain analysis using a probabilistic graph-based approach: The figure illustrates ground model predictions (depicted as blue squares), a traversability graph (where yellow edges represent traversable paths and red edges indicate non-traversable paths), and a segmented point cloud (with obstacles marked in green, traversable ground in blue, and non-traversable ground in magenta).

I. Introduction

Robust detection of obstacles and traversable space is a fundamental requirement for any autonomous robot to navigate safely. While this capability is easily achieved in indoor environments, it becomes considerably more challenging in outdoor settings due to unpredictable ground structures and uncontrolled illumination conditions. LiDAR sensors are a popular choice for outdoor scenarios because they provide precise geometric information over acceptable ranges and are relatively insensitive to external illumination sources. However, the sparsity and lack of color information in LiDAR data, combined with the complexities of outdoor environments, make LiDAR-based ground segmentation a highly demanding task, particularly when real-time performance is a practical necessity.

In recent research trends, Deep Learning models have gained prominence for the semantic segmentation of point clouds. These models have demonstrated remarkable results in benchmark tests such as SemanticKitti and NuScenes, accurately distinguishing among various classes [1], [2], [3]. However, Deep Learning methods have their drawbacks. They require an extensive amount of labeled data for supervised training, are computationally intensive both during training and inference, and raise concerns regarding their susceptibility to adversarial attacks when used in critical processes like obstacle detection [4].

In addition to Deep Learning methods, a wide range of alternative approaches exists. These include Elevation maps [5], Gaussian Processes [6], RANSAC with polynomial fitting [7], or Markov Random Fields [8]. Each approach has its advantages and drawbacks, but common challenges include achieving real-time performance, adapting to different sensors, and addressing traversability estimation.

To address these challenges, we propose GATA a probabilistic Graph-based real-time Approach for Terrain Analysis. The method generates a ground model organized as a graph structure encoding traversability information, facilitating fast

IDP, ASN, AGZ, and JAC are with the Institut de Robòtica i Informàtica Industrial, CSIC-UPC, Llorens i Artigas 4-6, 08028 Barcelona, Spain.

FT is with the Automation Robotics and Computer Vision Group (AUROVA), University of Alicante, San Vicente del Raspeig S/N, Alicante, Spain.

Corresponding author: idelpino@iri.upc.edu.

This work was partially supported by the EIT Urban Mobility project LOGISMILE (EIT-UM-2020-22140 and EIT-UM-2023-23374); the Spanish projects EBCON (PID2020-119244GB-I00, funded by MCIN/ AEI /10.13039/501100011033), AUDEL (TED2021-131759A-I00, funded by MCIN/ AEI /10.13039/501100011033 and by the "European Union NextGenerationEU/PRTR") and LENA (PID2022-142039NA-I00, funded by MCIN/ AEI /10.13039/501100011033 and by "ERDF A way of making Europe"); by BotNet (23S06128-00, funded by the "Ajuntament de Barcelona" and "Fundació la Caixa"); by the Consolidated Research Group RAIG (2021 SGR 00510) of the Departament de Recerca i Universitats de la Generalitat de Catalunya; and by a Margarita Salas Fellowship to IDP (MARSALAS21-08) funded by the Spanish Ministry of Universities, the European Union-Next Generation, and the University of Alicante.

and robust obstacle detection. Moreover, the ground model produces features that can be harnessed to refine traversability analysis. If ground-truth data is available, we train a shallow neural network to distinguish between traversable and non-traversable ground. While this shallow neural network cannot match the complexity of modern Deep Learning methods, it offers advantages, including the need for a modest amount of annotated data, fast training, and real-time segmentation on a single CPU core.

Key features of the method are:

- LiDAR-only: It can be utilized without complex hardware setups or calibration processes.
- Single-shot: It does not require odometry or localization for concatenating point clouds.
- Lightweight: It operates in real-time on a single CPU core.
- Easy to tune: It has a small number of meaningful parameters, such as Kalman noises, ROI sizes, and angular resolution.
- Flexible: It works with various 3D LiDAR sensors, regardless of their field-of-view or resolution.
- Easy to train: It requires minimal annotated data and training time [9].
- Complementary to Deep Learning methods: It can serve as an alternative to protect against adversarial attacks [10].
- Provides excellent point cloud segmentation and traversability analysis, outperforming existing methods in its class.

The remainder of this paper is organized as follows: in Section II, we provide a brief review of related works. In Section III, we offer a detailed description of our method, leading to an experimental evaluation in Section IV. Finally, in Section V, we present our conclusions and outline future research directions.

II. Related Work

Over the past few years, we have seen how Deep Learning-based algorithms have achieved excellent results in classic computer vision tasks, such as image classification or object detection, expanding technological possibilities and practical applications to unprecedented levels [11]. Beyond image processing, deep neural networks have recently been extended to point cloud processing, which is of great interest for autonomous driving since LiDAR sensors are a fundamental piece in most of these systems [12]. Initially, Deep Learning techniques were primarily applied to object detection in point clouds [13], however, with the advent of new tools, such as highly realistic simulators (e.g., CARLA [14]) and automated labeling algorithms (similar to those used in SemanticKITTI [1]), we now have access to point-wise-level annotated point cloud datasets. These datasets enable the development of Deep Learning-based semantic segmentation algorithms. Currently, the top-performing semantic segmentation methods in the SemanticKITTI benchmark rely on Deep Learning and employ various techniques. Among these, Knowledge Distillation [15] has gained prominence. In the study by Yan et al., [3], a Knowledge Distillation strategy is developed to leverage RGB camera information during network training but not during inference. This approach allows the system to operate even in the absence of cameras. Another notable example of Knowledge Distillation is found in the work presented by Hou et al., [16], where they introduce a method called "Point-to-Voxel Knowledge Distillation." By reducing the complexity of several large models, this technique enhances the handling of dispersion and variable density in point clouds generated by LiDAR sensors and currently ranks second in the SemanticKITTI semantic segmentation benchmark.

Despite their commendable performance, the practicality of employing deep-learning algorithms for ground segmentation is often limited. This limitation is primarily due to their substantial computational demands, rendering them unsuitable for real-time applications on robots lacking GPU capabilities [17]. Furthermore, deep neural networks possess qualities that are not yet fully understood and that generate counterintuitive and even "intriguing" effects, as shown in the work [18] where it is stated that unlike in classic computer vision algorithms, small or even imperceptible alterations in the input images of a deep network can cause serious classification errors in its output. These counterintuitive properties of deep networks have been studied within the Adversarial Machine Learning community [19] to develop methods able to degrade the performance of artificial Deep Learning-based systems by "adversarial attacks" and to create methods to defend them against such attacks [20].

Various studies have illuminated the vulnerabilities of deep learning-based point cloud processing algorithms. Noteworthy examples include: a) Physical Camouflage: In the work by Tu et al., [21] meticulously crafted objects with unconventional shapes alter a car's appearance, rendering it invisible to deep networks; b) Semantic Segmentation Confusion: Zhu et al., [10] strategically place small elements within the environment to confound semantic segmentation systems. These elements cause misclassification of vehicles as ground points, leading to their disappearance; and c) GNSS-Based Backdoor Attacks: Li et al., [22] reveal a potential backdoor in point cloud rectification using GNSS information. This common preprocessing strategy, employed to mitigate point cloud deformations during high-speed sensor movement, can inadvertently serve as a gateway for adversarial attacks. Spoofing techniques contaminate the GNSS signal, effectively evading the perception system's detection of pedestrians, cars, and other critical obstacles. In light of these studies, we contend that diversifying obstacle detection methods beyond deep learning is critical for ensuring redundancy and robustness in safety-critical processes. This motivation drove the development of our method and remains a key focus for our future research.

Numerous methods have been proposed in the literature to address real-time pointwise ground segmentation without the need for Deep Learning techniques. Initial algorithms utilized elevation maps to detect obstacles in point clouds, assigning height information to each occupied cell, such as the mean height of all contained points or the difference between the maximum and minimum heights [23]. Some methods modeled the ground plane as a set of concentric lines, estimating their

slope and intercept using least squares [24], albeit with the inherent limitation of having a single equation for each angular zone. More contemporary works have employed probabilistic approaches for ground segmentation, as demonstrated in [25], where the ground is modeled as a Markov Random Field, and the maximum-belief ground height is sought using a polar grid representation. Reymann et al. [26] introduced a hierarchical classification method based on Random Forests. This method uses geometric features to differentiate flat surfaces, vegetation, and obstacles. Intensity features and multi-echo data are then used to further refine the classification into grass, small rocks, and asphalt. Inspired by this approach, we incorporate a second classification stage in our method, employing a shallow neural network to distinguish between traversable and non-traversable ground. A highly efficient ground segmentation method is presented in [27]. Rather than operating in 3D space, this method employs a projection in spherical coordinates to generate Range Images. The data in these images is processed column-wise, with each column encoding the elevation angles, thereby facilitating the analysis of the ground's slope. Zermas et al. [28] divided LiDAR scans into planes along the front-back (x) axis, using Singular Value Decomposition based on the lowest detected points. This method classifies points using a simple point-to-plane Euclidean distance threshold. While fast, this approach struggles in the presence of noise and outliers. A more robust variant is presented by Luo et al. [29], which divides the environment into inner, forward, and backward regions and estimates slope and intercept parameters for the best-fitting line using RANSAC [30] and Iteratively Reweighted Least Squares. Although effective in structured roads, it is less suitable for complex, unstructured scenarios. Another RANSAC-based method is presented in [31], where a two-step algorithm is employed. Initially, the more apparent non-ground points are discarded using the sensor's geometry and the distance between consecutive rings in the scan. Subsequently, a multi-region RANSAC plane fitting is used to finalize the refinement of the ground segmentation.

All these works faced the challenge of lacking point-wise annotated datasets for performance evaluation. This changed in 2019 when the SemanticKitti dataset was released, providing labels for over four billion points classified into twenty-eight classes. This dataset has become a standard benchmark for ground segmentation, and we use it in this paper to compare our results with the state of the art (see Section IV). In the latest literature on ground segmentation, Huang et al. [8] present a coarse-to-fine method that divides the segmentation process into three stages. The first two are purely geometric analyses, using a ring-based elevation map and spherical coordinates representation to produce a coarse segmentation, which serves as initialization for the third stage. This final stage employs a graph-based approach with a min-cut algorithm to separate ground and non-ground points. Qian et al. [7] describe a ground modeling algorithm that creates a polar grid containing only points with the minimum z-coordinate. It then applies the method described in [32] to remove obstacles and performs RANSAC-based second-order polynomial fitting. This process generates a ground model used for point cloud classification based on an Euclidean error threshold. Lim et al. [33] use a variable cell size polar grid, called the Concentric Zone Model (CZM), to accommodate data sparsity. Principal Component Analysis is used to find local planes in the CZM representation. These local planes are combined using three different features (Uprightness, Elevation, and Flatness) to perform the final segmentation. Notably, all these methods are sensor-specific and make assumptions about the point cloud's distribution or projection. Popular methods that also include graph structures for LiDAR segmentation include Zhu and Liu's method for the segmentation of traversable areas in rough terrain [34] that divides LiDAR line scans into line segments by least squares linear regression and uses graph-cut to classify line segments into ground and non-ground; and Oh et al. [35] that builds a graph from a constant-resolution triangular grid and employs PCA within each cell to find the normal vector to each surface. Cells are classified as terrain, obstacle, or unknown based on the number of points in the cell and the normal vector. Traversability analysis is performed by encoding central points of terrain cells as nodes within the graph. A search is conducted to analyze the edges between neighboring nodes, marking non-traversable nodes as unreachable. Point cloud segmentation is achieved by thresholding the Euclidean distance between points and model planes. Similar to our approach, Oh et al. is sensor-agnostic and also encodes the ground model in a graph, but has no means to estimate the model's uncertainty. In contrast, our approach allows for probability distribution prediction in the Z-axis and uses Mahalanobis distance checks for better treatment of noisy data. Additionally, we use a shallow neural network to distinguish between different types of ground, providing vital traversability information, as certain areas, like grass or unpaved terrain, may present similar traversability features.

III. Method

A. Problem Statement

The problem we aim to address involves a ground robot equipped with one or more LiDAR sensors, with known mounting points through calibration. Despite having limited processing power, this robot must navigate outdoor environments, necessitating the detection of obstacles at extended ranges in complex, non-flat terrains for tracking, localization, and mapping tasks. Additionally, it must differentiate between traversable and non-traversable surfaces to support planning. Specifically, we need to determine which points in a given point cloud belong to obstacles and which belong to the ground. Among the ground points, we must distinguish between traversable and non-traversable areas. Furthermore, this method should run in real-time without requiring GPUs.

For obstacle detection, we build a representation of the ground in which height is modeled with a Gaussian distribution for any desired coordinate in the XY plane. This ground model provides the foundation for obstacle detection, effectively treating it as an outlier rejection task. We use a Mahalanobis threshold to differentiate obstacles from the ground. For the traversability classification of ground points, we utilize a supervised learning approach. We train a shallow

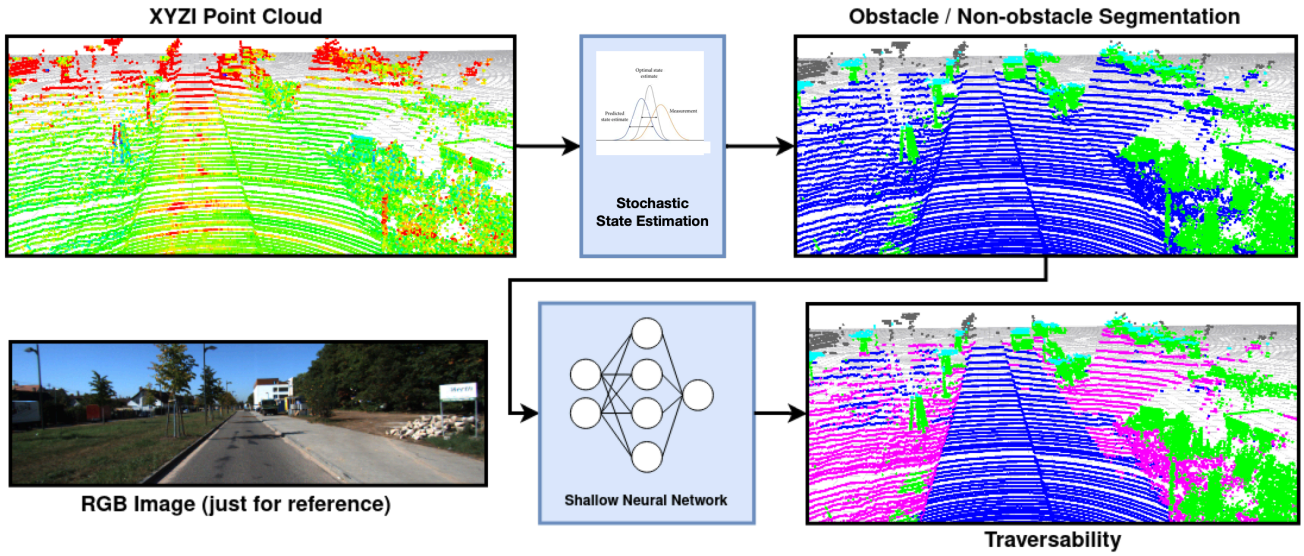

Fig. 2. Two-stage segmentation scheme. First, stochastic estimation is used to update parameters of the height and slope of the ground plane and to discard points belonging to obstacles. In the second step, a shallow neural network is used to classify traversable and non-traversable ground.

neural network (as discussed in Section III-J) to classify ground points as either traversable or non-traversable. This classification is based on a feature vector extracted from the point cloud and the ground model.

A schematic representation of the proposed segmentation pipeline is provided in Figure 2, with two sequential modules, one for ground segmentation, and the second one for traversability analysis. An algorithmic representation of the pipeline is given in Algorithm 1. The more relevant variables, referred to as Config Params in Algorithm 1, are thoroughly elucidated in Section IV-A.

Algorithm 1: Probabilistic real-time ground segmentation and traversability analysis

input : point cloud (P), config params (c)
output: ground model ($\mathcal{G}$), segmented point cloud ($\hat{P}$)
$L \leftarrow generateCloudOfReferences(P, c)$;
$v_o \leftarrow setRootVertexPrior(c)$;
$\mathcal{G} \leftarrow v_0$;
$k \leftarrow 0$;
while $k < \mathcal{G}.numberOfVertices()$ **do**
  $v_k \leftarrow \mathcal{G}.getVertex(k)$;
  $ROI_k \leftarrow extractReferencesInROI(v_k, L, c)$;
  $ROI_k \leftarrow v_k.filterROIWithPrior(ROI_k, c)$;
  $v_k \leftarrow v_k.computePosterior(ROI_k, c)$;
  $\mathcal{G} \leftarrow vk.generateNewVertices(ROI_k, c)$;
  $k \leftarrow k + 1$;
end
$k \leftarrow 0$;
while $k < L.size()$ **do**
  $[\hat{P}, f] \leftarrow detectObstacles(P, L(k), \mathcal{G})$;
  if $c.use\_neural\_net$ **then**
    $\hat{P} \leftarrow predictTraversability(\hat{P}, f)$;
  end
  $k \leftarrow k + 1$;
end

B. Ground segmentation

In our specific problem, the objective is to compute the parameters, height, and slope, of a supporting plane of the ground local to a given robot position. As the robot traverses the environment, these parameters change smoothly and we update them using stochastic state estimation. The approach is computationally efficient, robust, and, unlike some other methods in the state of the art, considers uncertainty, which is valuable for the identification of obstacles as outliers to the plane.

To implement this approach, we locally explore the point cloud as if a virtual rover navigates through it while estimating the supporting plane. The exploration starts from the point cloud's origin, where a solid prior of the ground plane is available thanks to sensor extrinsic parameter calibration. The exploration is then propagated to nearby points, with a linear propagation of uncertainty of ground plane parameters as the exploration proceeds away from the point cloud origin, and using point cloud data to revise the ground parameter estimates. This process repeats until the entire point cloud is analyzed.

1) Inputs: The algorithm operates with a minimal set of inputs:

- A 3D Point Cloud: This can be a single shot or an accumulation of point clouds. It may originate from a single sensor or be generated by multiple sensors, as long as all the points within the cloud share a common reference frame.
- Ground Plane Prior: To initiate the algorithm, a prior estimation of the ground plane at the robot's position is required. This estimation is used to distinguish between ground and obstacles in the robot's immediate vicinity.

Typically, this information is obtained through extrinsic parameter calibration. It's important to note that this is a prior estimation and doesn't need to be perfect, as the method will refine the estimation based on sensory data. This flexibility makes it possible to use the algorithm with various robots without the need for a complex calibration process.

- Sensor Uncertainty: Two parameters are needed to model sensor uncertainty, one to propagate ground parameter estimates away from the sensor, and one to revise these estimates with measured data.
- Resolution Parameters: Three resolution parameters are required, including the grid cell size, the region of interest size, and the angular exploration resolution.
- Thresholds: Two thresholds are utilized in the algorithm. The first is the Mahalanobis distance, used to identify point outliers, and the second is a score threshold, which determines whether a point is classified as ground or an obstacle.

2) Outputs: The algorithm provides two primary outputs: a Segmented Point Cloud and a Ground Model.

- Segmented Point Cloud. The Segmented Point Cloud is derived from the original point cloud and is enhanced with labels and colors that convey the segmentation results. It includes five distinct classes, each represented by a specific color: Unlabeled (grey), Traversable Ground (blue), Non-Traversable Ground (magenta), Obstacle (green), and Above-Obstacles (cyan). The "Above-Obstacles" class includes points that are associated with obstacles but do not pose a direct collision risk, such as overhanging tree branches.
- Ground Model. The Ground Model encapsulates the results of terrain analysis and is used to generate the point cloud segmentation while providing crucial traversability information. The Ground Model can be described as a graph $\mathcal{G}$, comprising vertices V and edges E. Each vertex $\boldsymbol{v}_k \in V$ contains the following information:

$$\boldsymbol{v}_k = [x, y, z, a, b, \sigma_z, \sigma_a, \sigma_b] \tag{1}$$

In this definition, the x and y coordinates are considered deterministic, while the z coordinate and the slopes in the x and y directions are modeled as Gaussian random variables with respective means of z, a, and b, along with standard deviations σ_z, σ_a, and σ_b. These random variables are iteratively refined in our method. It's important to note that, for efficiency, the method assumes the slopes and the z coordinate are independent variables, simplifying the Mahalanobis distance computation for point segmentation. This simplification significantly enhances the method's speed while having minimal impact on performance.

The graph has its root node located at the reference frame's origin (at ground level) and can be used for tasks like path planning, representing a promising direction for future research (see Fig. 5).

C. Cloud of References

To achieve efficient real-time performance, we devised a reduction technique that we call "cloud of references." Instead of processing the entire point cloud, we divide it into smaller partial clouds by employing a grid with cells of size $s \times s$ (see Table I) in the XY plane. Each cell that contains at least one point forms a partial point cloud. For each of these partial point clouds, we generate a "reference" that includes the point with the lowest z coordinate in the cell. Additionally, a vector stores the indices of the remaining points in the same cell, which is later used to expedite the segmentation process. A flag is also employed to indicate whether the reference has already been utilized to create a vertex or not. The exploration process halts when no new vertices are generated. These references are collectively stored in a reference point cloud denoted as L (for Lowest), and the subsequent ground modeling is performed using this reduced dataset. This approach offers several advantages. Firstly, the regular grid structure ensures the algorithm's independence from the data acquisition method, eliminating assumptions like concentric circles or specific point distributions related to LiDAR sensors. Secondly, it mitigates the impact of varying LiDAR point density, which often occurs as the distance from the sensor increases. Lastly, it significantly reduces the volume of data that requires processing.

D. Root vertex prior

The initial step in generating the "Ground Model" involves creating the root vertex. This step necessitates selecting appropriate values for all the parameters specified in Equation 1, which together form the initial estimation of the ground plane at the origin of coordinates within the reference frame. While it may seem like a meticulous task, determining suitable values is actually quite straightforward. Typically, x and y are set to zero, representing the origin of the reference frame. The parameter z denotes its distance to the ground, while the slopes a and b are initially set to zero. This is because the Z axis is typically aligned vertically with the robot's axis, ensuring that the reference frame remains orthogonal to the ground plane.

The remaining parameters (σ_z, σ_a, and σ_b), which capture uncertainties and play a crucial role in outlier rejection, can be fine-tuned as follows. Begin with very small values and gradually increase them until no ground points are misclassified as obstacles. This iterative approach helps strike a balance between effectively identifying ground points and avoiding misclassifications.

E. Extracting references in the Region of Interest

In line with our core concept of local data analysis for real-time performance, we treat each vertex, denoted as $\boldsymbol{v}_k$, as a unique point of view from which we estimate a ground plane. This estimation solely relies on the data within a defined vicinity around the vertex, termed the "references" within a Region of Interest (ROI). The appropriate size for this area is contingent on the density of the point cloud and the intricacy of the environment under examination. To allow for

flexible adjustment, we introduce a parameter Δ that governs the dimensions of a square ROI, measuring $2\Delta \times 2\Delta$, with its center aligned at the vertex's position. The choice of Δ allows users to tailor the ROI size to suit the characteristics of the data, accommodating varying point cloud densities and environmental complexities.

F. Filtering the ROI using the prior

In our approach, even though we exclusively use the lowest point within each grid cell to establish references, it is common to find points within a Region of Interest (ROI) that pertain to obstacles. Such occurrences often result from occlusions and a lack of vertical resolution in the LiDAR data. Therefore, it is imperative to filter out these points before proceeding with ground plane estimation.

Leveraging the "prior" information stored in the vertex $\boldsymbol{v}_k$, along with a first-order error propagation operation, we can predict both the z value and its associated uncertainty for any given x and y coordinates:

$$\hat{z} = z_{v_k} + (x - x_{v_k})\, a_{v_k} + (y - y_{v_k})\, b_{v_k} \tag{2}$$

$$\hat{\sigma} z^2 = {\sigma_z}^2 v_k + (x - x_{v_k})^2 {\sigma_a}^2_{v_k} + (y - y_{v_k})^2 {\sigma_b}^2_{v_k} \tag{3}$$

Subsequently, for each point within ROI_k, we calculate its Mahalanobis distance by dividing the magnitude of the prediction error by the standard deviation of the prediction $d = |z - \hat{z}| \,/\widehat{\sigma}_z$ and, using a specified threshold value, denoted as τ (see Table I), we determine which of these points should be included as "observations" for computing the "posterior" estimation of the ground plane at the vertex's position.

From an implementation perspective, it's noteworthy that, since references can belong to multiple Regions of Interest, we store the ID of the vertex that yielded the most accurate prediction for each reference. This approach enables us to identify the vertex to use during the segmentation step without the need for time-consuming searches, consequently enhancing the algorithm's efficiency.

G. Computing the posterior estimation

To compute the posterior estimation, we adopt a sequential innovation approach [36], which offers an efficient implementation, particularly because the matrix inversion required for calculating the posterior covariance simplifies to a scalar division, given that our measurements consist of scalar values z. Here, we detail the procedure:

We first organize the vertex prior estimation (which stems from either the spatial propagation of its parent or the initialization process for the root vertex v_0) into a state vector and covariance matrix. For notation simplicity, we will omit the subscript v_k, as all values pertain to the node under update:

$$\boldsymbol{x}^{\ominus} = [z, a, b]^T \tag{4}$$

$$\boldsymbol{P}^{\ominus} = \text{diag}\left({\sigma_z}^2, {\sigma_a}^2, {\sigma_b}^2\right) \tag{5}$$

Next, we express the measurement model from Equation 2 in vector form $z^{(i)} = \boldsymbol{H}^{(i)}\boldsymbol{x}$, with $\boldsymbol{H^{(i)}} = \left[1, \left(x^{(i)} - x\right), \left(y^{(i)} - y\right)\right]$. We initialize $\boldsymbol{x}^{\oplus,\mathbf{0}} = \boldsymbol{x}^{\ominus}$ and $\boldsymbol{P}^{\oplus,\mathbf{0}} = \boldsymbol{P}^{\ominus}$ to initiate the sequential innovation scheme

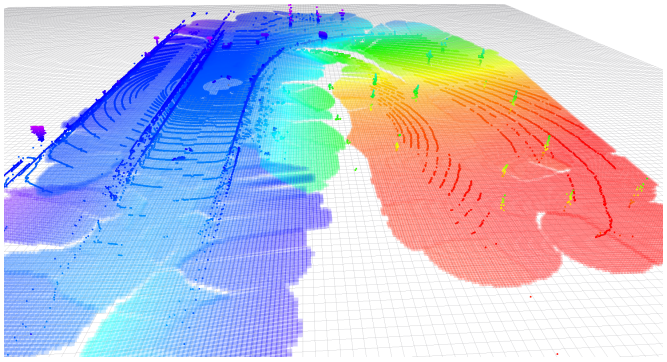

Fig. 3. Dense reconstruction of the ground using the predictions of the Ground Model. Predicted points are colored following their z value from red (low) to purple (high).

that runs for the n ground measurements within each ROI_k, denoted as $p^{(i)} = \left[x^{(i)}, y^{(i)}, z^{(i)}\right]$.

$$\boldsymbol{K^{(i)}} = \boldsymbol{P^{\oplus,i-1}}\boldsymbol{H^{(i)^T}}\left(\boldsymbol{H^{(i)}}\boldsymbol{P^{\oplus,i-1}}\boldsymbol{H^{(i)^T}} + r\right)^{-1} \tag{6}$$

$$\boldsymbol{x^{\oplus,i}} = \boldsymbol{x^{\oplus,i-1}} + \boldsymbol{K^{(i)}}\left(z^{(i)} - \boldsymbol{H^{(i)}}\boldsymbol{x^{\oplus,i-1}}\right) \tag{7}$$

$$\boldsymbol{P^{\oplus,i}} = \boldsymbol{P^{\oplus,i-1}} - \boldsymbol{K^{(i)}}\boldsymbol{H^{(i)}}\boldsymbol{P^{\oplus,i-1}} \tag{8}$$

In the equations above, the scalar r represents the variance of the additive measurement noise, which we consider to be constant and identical for every measurement (as indicated in Table I). Following these computations, we store in vertex $\boldsymbol{v_k}$ (as defined in Equation 1) the means and standard deviations of the last update, effectively representing the final posterior distribution.

Figure 3 shows height estimates for the point cloud of the highway shown in Figure 4. In addition to the estimation for each point in the point cloud, we showcase a dense reconstruction of the ground computed using the posterior estimation of the Ground Model. The color code represents z coordinate values, ranging from red (low) to purple (high). To create this reconstruction, we instruct the Ground Model to generate z predictions for pairs of x and y coordinates. All predictions displayed in the figure have a maximum standard deviation of one-third of a meter, thus indicating that the reconstructed area has a maximum uncertainty of one meter within the 3σ region. By examining the colors, one can observe how the Ground Model's predictions closely match the colors of the point cloud.

H. Generating new vertices

To create new vertices, we utilize the references within the filtered ROI_k that remain unexplored. These references are grouped into azimuthal regions, each spanning $\Delta\phi$ degrees (see Table I) around the current vertex. New vertices are then created at positions corresponding to the median azimuthal angles within each region. Once these new vertices are established, the used references are marked as *explored* to prevent redundant processing.

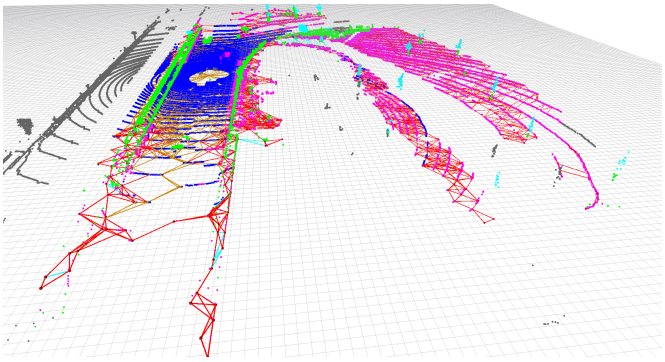

Fig. 4. Point cloud segmentation and Traversability graph in a highway with more than thirteen meters of altitude difference between the road and the lowest point in the cloud. Color code: Traversable ground (blue), non-traversable ground (magenta), navigation obstacles (green), overhanging obstacles (cyan), and not analyzed (gray).

To set the *prior* estimation for the newly created vertices, we propagate the parent vertex's state to the new positions using the following equations:

$$\boldsymbol{x}_{v_{k+1}} = \boldsymbol{F}_k \boldsymbol{x}_{v_k} \tag{9}$$

$$\boldsymbol{P}_{v_{k+1}} = \boldsymbol{F}_k \boldsymbol{P}_{v_k} \boldsymbol{F}_k{}^T + \boldsymbol{Q}_k \tag{10}$$

$$\boldsymbol{F}_k = \begin{bmatrix} 1 & x_{v_{k+1}} - x_{v_k} & y_{v_{k+1}} - y_{v_k} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{11}$$

$\boldsymbol{Q}_k$ denotes a zero-mean Gaussian additive noise scaled by the squared Euclidean distance between the parent and child vertices (d^2):

$$\boldsymbol{Q}_k = d^2 \begin{bmatrix} q_z^2 & 0 & 0 \\ 0 & q_a^2 & 0 \\ 0 & 0 & q_b^2 \end{bmatrix} \tag{12}$$

The parameter standard deviations (q_z, q_a, q_b) are fine-tuned through experimentation, the chosen values are given in Table I. Finally, we establish edges to connect the current vertex v_k with each newly created child vertex, adding them to the edges vector E of the graph $\mathcal{G}$.

I. Detecting obstacles using the ground model

Once all new vertices have been created, and no further vertices appear, the Ground Model is considered complete and ready for use in point cloud segmentation. To perform this segmentation efficiently, we use the cloud of references, denoted as L, which guides the process. Each reference within L provides us with the ID of the vertex that yielded the best prediction, along with a vector of indices pointing to the remaining points within its respective grid cell.

To segment the point cloud, we iterate over the vector of indices and, for each point, request the corresponding vertex in the Ground Model to generate a Gaussian probability distribution for z based on the x and y coordinates of the point. We then compute the Mahalanobis distance (d) and compute the score:

$$\eta = 1 - \frac{d}{\tau} \tag{13}$$

Points are considered as *ground* if their score exceeds a specific threshold, denoted as ν in Table I. Points that fail to reach this threshold are classified as non-ground. However, this classification does not necessarily imply that they are obstacles to navigation. Whether they pose a collision risk depends on three factors: the z coordinate of the ground (predicted or observed), the z coordinate of the point, and the robot's height (h as defined in Table I). Given these considerations, we categorize non-ground points into two classes: *obstacle* and *above-obstacle*.

Furthermore, points associated with references in L that remain labeled as *unexplored* after the ground modeling process is completed are classified as *unlabeled*. This typically occurs when the size of the Region of Interest is insufficient to analyze certain parts of the point cloud. This may be due to large occluded areas or the limited vertical resolution of the LiDAR sensor, resulting in undetected ground in those regions.

J. Predicting traversability

The algorithm, as described thus far, excels at efficiently detecting obstacles, with the ability to process a full Velodyne HDL-64 point cloud in just around eleven milliseconds on a standard laptop. This efficiency is crucial for tasks such as moving object tracking and localization. However, the information it provides alone may be insufficient for path planning, as whether the ground is considered traversable or non-traversable depends on various factors such as surface material and specific application requirements. For instance, a robot may or may not be allowed to navigate on grass.

To address this limitation without compromising real-time performance or necessitating GPU usage, we incorporate a Shallow Neural Network that operates in a supervised manner. This small neural network has dimensions of $13 \times 39 \times 2$, with thirteen features as input, thirty-nine neurons in the hidden layer, and two outputs, enabling binary classification (traversable / non-traversable) in real-time.

To create the input vector, we gather thirteen features for each point classified as *ground* by the Ground Model. These features encompass both point-level and cell-level characteristics, calculated by computing statistics on the points indexed by the references in L. Specifically, these features consist of:

- Point-level features: Squared ranges (point-to-sensor and point-to-reference), incidence angle (relative to the local plane estimated by the ground model), point intensity, prediction error (Euclidean distance between the predicted z value by the model and the point's z coordinate, which can be negative or positive), and score (as defined in 13).
- Cell-level features: The ratio between the number of ground and obstacle points in the cell, as well as the mean and variances of intensities, prediction errors, and scores.

It is important to note that we intentionally excluded certain available information from this set of features. While this information could have potentially improved quantitative results, such as the $3D$ coordinates of the points, we made this decision to prevent the network from overly focusing on the

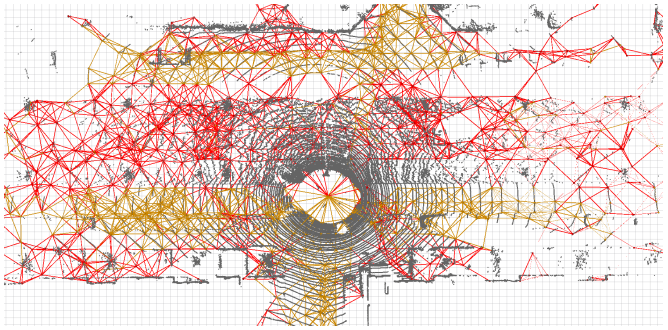

Fig. 5. Traversability graph in an urban environment. Each node contains a local plane estimation and vertices encode the traversability by connecting neighboring nodes whose plane estimations are geometrically compatible. In yellow we show the traversable edges, whereas in red we show connections discarded by the neural network.

structural details of the environment. For instance, sidewalks typically appear to the left or right of the vehicle, while the road is generally ahead or behind. While this information is valuable, relying too heavily on it could lead to undesirable outcomes, like failing to detect sidewalks in front of the robot, potentially hindering the integration of the method into an autonomous navigation system.

Using these feature vectors, the Shallow Neural Network is trained to classify points as traversable or not. Figure 5 shows the resulting traversability graph, with traversable edges shown in yellow and non-traversable ones in red.

IV. Experiments

The algorithm's performance is evaluated in two distinct settings: First, for quantitative comparison with other state-of-the-art methods, we employ the SemanticKITTI Dataset [1]. This dataset provides pointwise LiDAR ground truth and is widely regarded as one of the primary benchmarks for assessing the accuracy of ground segmentation algorithms. Secondly, we conduct a series of experiments with a last-mile delivery robot in the context of the Logismile project [1]. The experiments included a variety of scenarios, with tests in the Barcelona Robot Lab, a complex outdoor environment located in the Universitat Politècnica de Catalunya campus [37], and demonstrations in urban settings in Esplugues de Llobregat (Spain) and Debrecen (Hungary). We provide detailed descriptions of both evaluation setups in the following sections.

A. Configuration Parameters

For both evaluations, we maintained the same configuration parameters, as detailed in Table I. They play a crucial role in determining the algorithm's performance. Here's an overview of the key parameters and their influence on the algorithm:

- Cell Size:
  Description: The cell size represents the size of the cells in the regular grid used to generate the cloud of references.
  Effects: The density of the Cloud of References is primarily determined by the size of the cells. Smaller cells yield a higher number of nodes in the Ground Model, which is beneficial for complex environments. However, this increased complexity slows down the algorithm and provides less context for the neural network, as some input features are derived from cell-level statistics. Conversely, larger cells generate sparser reference clouds, simplifying the Ground Model. This reduction in complexity decreases computational costs and provides more context to the neural network. However, if the cell size is excessively large, errors due to cells containing simultaneously different surface materials (e.g., grass and asphalt) will affect a significant number of points.
  Given that the Cloud of References is used for model fitting and exploration, the cell size is a critical factor when adjusting the Region of Interest (ROI) size and the Exploration Resolution. This is essential as we require observations within the ROIs to maintain active exploration and to accurately correct the prior estimation.
  Typical Values: Cell sizes ranging from 0.5 to 3.0 meters are considered appropriate. The optimal selection depends on the complexity of the environment and the available computational resources.
- Root Node Prior:
  Description: These parameters control the creation of the root node in the graph, representing the initial estimation of the ground.
  Effects: Setting x_0 and y_0 can help position the root node at the center of the shadow area beneath the robot's

[1]LOGISMILE - www.eiturbanmobility.eu/projects/logismile

TABLE I
Parameter setup in KITTI experiments

parameter		value	units
cell size	$s \times s$	2.1 × 2.1	m^2
root node prior	x_0, y_0	0.0, 0.0	m, m
	z_0, σ_{z_0}	-1.73, 0.05	m, m
	a_0, σ_{a_0}	0.0, 1.5	deg, deg
	b_0, σ_{b_0}	0.0, 1.5	deg, deg
ROI sizes	Δ_{root}	7.0	m
	Δ	3.0	m
Mah. thres.	τ	3.0	-
Score thres. (no NN)	ν	0.475	-
Score thres. (with NN)	ν	0.0	-
standard deviations	r	0.3	m
	q_z	0.01	-
	q_a, q_b	0.4	deg / m
exploration resolution	Δ_ϕ	40.0	deg.
robot height	h	2.0	m

TABLE II
Shallow Neural Network Hyperparameters

hyperparameter	value
Input size	13
Hidden Layers	1
Neurons	39
Output size	2
Activation	Hyperbolic Tangent Sigmoid
Optimization	Bayesian Regularization
Loss Function	Sum of Squared Errors
Training sequences	0, 1, 2, 3, and 4
Training Samples	≈ 500K
Training Epochs	200
Training Time	≈ 5 min.

sensors. An appropriate value for z_0 should be based on the expected z coordinate of the ground from the reference frame. Standard deviations for x_0, y_0, z_0, a_0, and b_0 should be as small as possible without causing false negatives.
Recommendations: Tune these values based on the robot's shadow area and the ground's expected z coordinate.

- ROI Sizes:
  Description: Two different ROI sizes are defined: one for the root node (Δ_{root}) and another for the rest of the graph (Δ).
  Effects: ROI sizes determine the area around each vertex in the graph, influencing the exploration process. These sizes need to be large enough to ensure full coverage of the point cloud. When processing single-shot point clouds, a larger ROI may be necessary to accommodate the increased distance between ground points captured by different LiDAR layers. The ROI size should not be excessively large to maintain the local data analysis approach.
  Recommendations: Ensure the ROI sizes are appropriate for the environment and sensor characteristics.
- Mahalanobis Threshold:
  Description: The Mahalanobis threshold is used for outlier rejection during Ground Model creation and score computation during segmentation.
  Recommendations: The threshold value is typically kept fixed at 3 sigmas. Adjust the variances (prior, propagation, and observation) to achieve the desired behavior.
- Score Threshold:
  Description: The score threshold is used in the first segmentation stage to classify points as ground or obstacle.
  Effects: The threshold can be used to balance precision and recall in cases where the neural network is not used. A higher threshold reduces false positives but may lead to more false negatives.
  Recommendations: When the neural network is applied, a threshold of zero is recommended to let the network correct false positives from the initial segmentation.
- Standard deviations:
  Description: Parameter standard deviations (q_a, q_b, and q_z) are used to induce uncertainty into the prior estimation of new vertices. They are linearly scaled by the distance from the parent to the child node.
  Effects: Properly tuned standard deviations are essential for accurate results. Small values may lead to overconfidence and hence misclassification of ground points as obstacles, while large values may lead to a much permissive classification setting, misidentifying obstacles as ground.
- Exploration Resolution:
  Description: This value determines the resolution used for generating new vertices.
  Effects: Exploration resolution, in conjunction with the ROI size, controls the number of vertices created in the Ground Model. Optimal tuning ensures sufficient vertices to describe the environment without redundancy.
- Robot Height:
  Description: This parameter sets the threshold for declaring high obstacles as non-collidable, based on the height above the ground level predicted by the Ground Model.

Careful tuning of the presented parameters is essential to ensure that the algorithm performs well and provides accurate traversability analysis results in various environments and use cases. The specific values for these parameters may vary depending on the robot's characteristics and the characteristics of the environment in which it operates.

B. Shallow Neural Network training details

The process of training the Shallow Neural Network involves several key steps. Here's an overview of the training details:

- Dataset Preparation:
  A dataset was generated by running the GATA algorithm in ROS using SemanticKITTI data. The features extracted from GATA were saved into files for further processing.
- Choice of Supervised Learning Algorithm:
  Matlab's Classification Learner App was employed to explore and evaluate different supervised learning algorithms. Various algorithms, including Decision Trees, Random Forests, and Support Vector Machines, were experimented with. The goal was to find an algorithm that struck the right balance between classification accuracy and speed, as real-time processing was a requirement.
- Selection of Shallow Neural Network:
  After conducting experiments, a Shallow Neural Network showed promise, achieving similar results to Random Forest but with faster processing. Bayesian Regularization with the Sum of Squared Errors as a loss function was chosen to optimize the hyperparameters.
- Dataset Size and Optimization:
  The size of the training dataset was observed to have a limited impact on classification performance. There was a point at which adding more training examples did not result in further improvements. The dataset was decimated to strike a balance between training speed and classification accuracy.
- Validation and Training Subset:
  The training dataset was divided into a training subset and a validation subset. The first five sequences (representing 49 percent of the data) were used for training, while the remaining six sequences (51 percent of the data) were reserved for validation. The training subset was further reduced by randomly selecting one point (classified as "ground" by the Ground Model) from each reference in L and discarding nine out of ten of these points.
- Final Training Subset:
  The final training subset consisted of approximately 500,000 feature vectors, representing less than 0.02 percent of the entire dataset. Given the large volume of data, this decimated subset was deemed sufficient for training.
- Training of Shallow Neural Network:

The Shallow Neural Network was trained for 200 epochs. The training was completed in less than five minutes on a standard laptop.

- Uniform Performance:
  It was observed that there were no substantial differences in performance between the sequences used for training and the remaining sequences. Therefore, the results were not separated into training and validation sets.

The training and selection process for the Shallow Neural Network was made taking into account the need to optimize accuracy and speed while working with large LiDAR datasets. The final selected hyperparameters are given in Table II.

C. Evaluation in the SemanticKITTI Dataset

The SemanticKITTI dataset, an extension of the KITTI dataset [38], is an ideal benchmark for assessing methods related to high-level scene understanding. It comprises a total of 28 classes, making it particularly suited for deep learning-based approaches. However, for the specific task of traversability analysis, only two classes are relevant: "traversable" (indicating safe ground surfaces for navigation) and "not traversable" (encompassing obstacles, overhanging objects, and ground surfaces unsuitable for navigation). The dataset's abundance of classes presents a challenge for evaluating traversability analysis methods due to several factors:

Practicality: SemanticKITTI does not provide tools to consolidate classes, making it impractical for traversability analysis evaluations, especially since the evaluation process is conducted on a remote server.

Non-uniform Evaluation: The dataset leaves the decision of which classes should be considered as traversable and non-traversable up to the researcher. This subjectivity can lead to non-uniform evaluations of methods, making direct comparisons challenging.

For experimental comparison with the state-of-the-art, we selected two contemporary papers that detail high-performance, real-time methodologies that operate independently of GPU acceleration: Hy-Seg [7], and TRAVEL [35]. Among these, only TRAVEL's implementation is publicly accessible via a repository. Consequently, we designed two distinct experiments: We first replicated the experiments delineated in the Hy-Seg paper, employing both GATA and TRAVEL. This allowed us to perform a quantitative comparison against all the methods enumerated in the Hy-Seg paper. Then, we conduct an experiment to compare GATA and TRAVEL in a context similar to the Logismile project where the robot is allowed to navigate by sidewalks, parking areas, and roads but not through vegetation or unpaved terrain. The comprehensive descriptions of these experiments are presented in Sections IV-C1 and IV-C2 respectively.

1) First Evaluation, only Road is traversable: For this evaluation, we conducted experiments and employed metrics outlined in the Hy-Seg paper by Qian et al. [7]. Specifically, we measured Intersection over Union (IoU) and Recall using the following equations:

$$IoU = \frac{TP}{TP + FP + FN} \tag{14}$$

$$Recall = \frac{TP}{TP + FN} \tag{15}$$

These metrics were calculated for each driving sequence within the SemanticKITTI training dataset, and we reported the mean values in Tables III and IV. Our evaluation considered three different versions of our method:

- GATA Without Neural Network (Score Threshold of 0.475):
  This version of the method does not incorporate a neural network. It utilizes a minimum score threshold of 0.475 (see Eq. 13) to classify points as traversable. The results demonstrated similarity to state-of-the-art methods, with slightly better performance in the Intersection over Union (IoU) metric and slightly lower scores in Recall. This was attributed to fewer false positives due to the score threshold, albeit at the cost of an increase in false negatives.
- GATA with Neural Network (VEGETATION and TERRAIN Non-Traversable, Score Threshold of 0.0):
  This version integrates a neural network trained to classify VEGETATION and TERRAIN as non-traversable classes. It employs a score threshold of 0.0. This configuration resulted in a significant increase of approximately ten points in the IoU metric, with no adverse impact on Recall. The neural network effectively reduced false positives and slightly decreased false negatives, indicating improved classification of ROAD points without confusion with VEGETATION or TERRAIN.
- GATA with Neural Network (Only ROAD Traversable):
  In this version, the neural network is trained exclusively to classify ROAD points as traversable. It achieved the highest IoU results, with an increase of approximately twenty-five points compared to state-of-the-art methods and other configurations. However, this came at the expense of a reduction of approximately ten points in Recall. The reduction in False Positives –which is the key factor in the IoU improvement– was due to the inclusion of SIDEWALK and PARKING in the non-traversable class (the GATA VEG+TER classifies them as traversable), but this increased False Negatives causing the degradation in the Recall metric as some ROAD points were misclassified as non-traversable.

Notably, the inclusion of SIDEWALK and PARKING classes in the non-traversable category raised challenges, as they exhibited similarities within the feature space with the ROAD class, making it more difficult for the neural network to distinguish between them.

2) Vegetation and Terrain are not traversable: For this evaluation, we have prioritized the use case that holds particular significance for our research: the integration of the algorithm into a mid-sized autonomous delivery robot designed for operation in urban environments, encompassing pedestrian areas and roads with vehicular traffic. In this context, we define traversable classes as ROAD, SIDEWALK, PARKING, and LANE MARKING, while categorizing TERRAIN and VEGETATION as non-traversable. Given the availability of only the TRAVEL method in a public repository, we compare three methods: TRAVEL, GATA without a neural network

TABLE III
IOU - ONLY *ROAD* IS TRAVERSABLE

Method	Mean	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
Douillard [23]	44.12	40.20	63.76	41.06	40.77	60.27	40.68	33.62	43.82	37.17	42.09	41.86
Bogoslavsky [27]	44.98	41.51	64.85	41.47	39.63	61.48	40.13	34.05	44.86	39.20	44.04	43.59
Himmelsbach [24]	44.90	41.24	63.71	40.39	40.74	61.73	39.83	33.71	44.38	40.38	43.84	43.97
Zhang [25]	41.18	36.92	62.40	38.43	37.50	60.39	33.83	32.09	39.84	34.84	40.05	36.66
Huang [8]	47.15	43.71	68.84	43.23	43.09	63.40	39.52	34.18	47.50	41.63	46.19	47.33
Hy-Seg [7]	47.40	44.09	67.94	44.82	43.77	61.07	41.40	35.49	46.76	42.99	46.84	46.19
TRAVEL [35]	44.56	42.39	65.04	41.15	39.40	61.81	37.69	35.10	44.92	38.62	42.67	41.40
GATA w/o NN	47.58	44.16	68.05	44.14	45.00	66.00	39.92	36.42	46.17	41.58	46.13	45.77
GATA VEG + TER	57.12	50.20	79.00	49.42	56.97	75.38	46.17	55.84	52.97	53.88	54.09	54.44
GATA ROAD	**72.53**	**70.72**	**84.05**	**72.68**	**68.80**	**76.09**	**68.81**	**74.41**	**73.45**	**73.08**	**71.31**	**64.45**

TABLE IV
RECALL - ONLY *ROAD* IS TRAVERSABLE

Method	Mean	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
Douillard [23]	96.25	96.70	94.82	97.44	95.49	94.92	96.25	96.87	95.86	97.18	96.51	96.67
Bogoslavsky [27]	97.00	96.87	95.44	97.90	96.77	98.67	97.78	95.91	96.47	96.61	97.89	96.67
Himmelsbach [24]	96.91	95.76	96.30	98.72	95.79	96.70	97.80	97.18	95.72	97.30	96.93	97.83
Zhang [25]	98.11	97.90	**98.93**	98.10	98.12	99.06	97.76	97.69	97.94	97.84	98.31	97.54
Huang [8]	98.93	99.42	97.05	99.29	**99.54**	99.47	99.53	98.80	98.93	**99.33**	98.72	98.18
Hy-Seg [7]	98.89	**99.65**	98.55	98.76	98.47	99.24	98.79	99.10	**98.94**	98.74	98.63	**98.94**
TRAVEL [35]	**99.12**	99.08	98.49	**99.52**	98.93	**99.92**	**99.65**	**99.47**	98.05	99.24	**99.31**	98.65
GATA w/o NN	98.26	97.82	95.23	99.21	98.68	99.76	98.82	98.64	96.88	98.36	99.30	98.17
GATA VEG + TER	98.38	98.77	98.08	98.82	98.39	97.63	99.28	97.09	98.81	98.60	98.18	98.50
GATA ROAD	87.49	91.90	90.64	88.25	84.59	87.25	88.47	87.09	90.39	88.47	86.61	78.75

(with a score threshold of 0.475), and GATA with a neural network trained to exclude VEGETATION and TERRAIN from the traversable class set (score threshold set to 0.0).

The results of these experiments are presented in tables V (mean values) and VI (standard deviations). In addition to the Intersection over Union (IoU) and Recall metrics defined earlier, we introduce the following indicators:

- Precision (P): Precision is calculated as $P = \frac{TP}{TP+FP}$, providing insight into the accuracy of positive predictions.
- F1 Score ($F1$): The F1 score, defined as $F1 = \frac{2*P*R}{P+R}$, represents the balance between precision and recall and offers a single metric to gauge performance.
- Accuracy (Acc): Accuracy is computed as $Acc = \frac{TP+TN}{TP+TN+FP+FN}$, reflecting the overall correct classification rate.

We also include two additional metrics:

- Execution Time (T): Measured in milliseconds, this metric signifies the time required to run the algorithms.
- Key Obstacle Recall (KOR): As defined in [8], this metric calculates Recall based on points belonging to potential dynamic obstacles like pedestrians, cars, bicycles, and more.

Upon analysis of the tabulated results, a pattern emerges that aligns with our previous evaluation. GATA outperforms TRAVEL in Precision, F1, Accuracy, Intersection over Union, and Key Obstacles Recall, while TRAVEL shows superior Recall. As a numerical example, GATA exhibits an approximate two-point advantage in Intersection over Union (IoU) without the use of a neural network, and a significant fourteen-point enhancement when employing the network. Turning our attention to execution time, GATA emerges as the fastest method when it operates without the neural network. However, its performance experiences a slowdown when the network is utilized. Notably, all experiments were conducted within a Virtual Box on a laptop. Despite this constraint, the version of GATA that incorporates the neural network still meets real-time requirements as it consumes a mean of about 70 milliseconds to process a point cloud while the Velodyne sensor produces them at a rate of $10Hz$. It is of great interest, in order to fairly assess the significance of this experiment, to consider that in its original paper, TRAVEL was compared to RANSAC[30], Zermas et al.[28], Narksri et al.[31], and Lim et al. [33], achieving the best results in terms of F1 score and Accuracy, while also being the fastest method among those evaluated.

3) Quantitative Results Analysis: Analyzing globally the quantitative experimental results, we can observe that when our system operates without the Neural Network, it delivers results that are competitive with top-performing state-of-the-art methods. Our approach, however, offers superior speed, efficiency, and adaptability, primarily due to the probabilistic score threshold that balances Precision and Recall. As demonstrated in Table V, thanks to the score threshold (experimentally adjusted to 0.475 to maximize the Intersection over Union results) our system (GATA w/o NN) outperforms TRAVEL in every metric (Precision, F1, Accuracy, IoU, Key Obstacles Recall, and Execution Time) except Recall. Importantly, if Recall is of particular importance, it can be enhanced by simply lowering the score required to classify points as traversable.

When our method makes use of the shallow neural network, it significantly outperforms the state-of-the-art. This improvement becomes more pronounced as the number of primarily flat surfaces included in the non-traversable class increases, as competing methods struggle to distinguish between different textures. Our experimental results demonstrate that our shallow

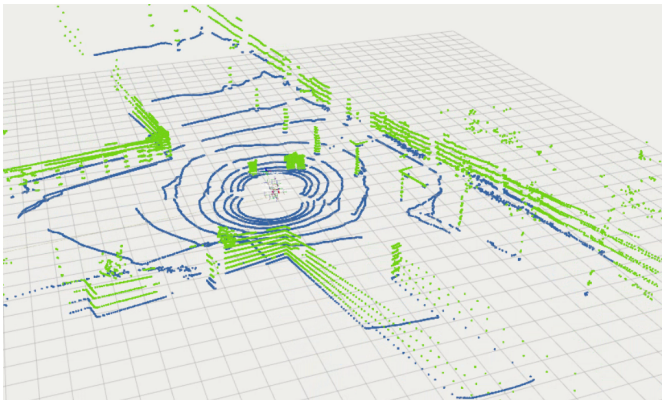

Fig. 6. Point cloud captured with the ONA robot in the Barcelona Robot Lab. The method adequately classifies traversable and non-traversable regions even for this very different sensor configuration.

Fig. 7. ONA robot during the Logismile demo in Esplugues de Llobregat

Fig. 8. ONA robot navigating in Debrecen.

neural network is proficient at road detection. However, this comes with a slight trade-off in the recall metric, attributable to the texture similarity in the feature space among structured surfaces. Despite this, our neural network excels in detecting natural textures, such as those found in vegetation or unpaved terrain. This capability is crucial for urban robot navigation, underscoring the practical relevance and potential of our method.

D. Real tests with the ONA robot

The ONA robot, developed as part of the Logismile project, is a mid-sized delivery robot engineered for autonomous operation in urban settings. This includes traversing pedestrian areas such as sidewalks and squares, as well as navigating on urban roads alongside other vehicles like cars and motorcycles. Notably, our initial tests with ONA were conducted before our algorithm incorporated the Neural Network. These early tests played a pivotal role in motivating us to develop a method to distinguish between different ground surfaces.

In ONA, the LiDAR placement was engineered to minimize shadow areas by lowering its mounting height. Unlike the KITTI setup, which employs a single 360° sensor atop the vehicle, we equipped ONA with two 360° sensors placed at opposing corners near the ground (see Figure 8). Note also that the four small cylinders atop the robot are not LiDARs but rather light signals used for indicating maneuvers and conveying information about the robot's operational state.

ONA's LiDAR sensors are Robosense RS-Lidar-16 models, featuring 16 layers, a horizontal resolution of 0.2° at 10 Hz, and a vertical resolution of 2.0°. While this setup and specifications significantly differ from KITTI, our algorithm's general approach allows for seamless integration without requiring any special processing. A sample point cloud captured with the ONA robot is presented in Fig. 6. This image illustrates how the shadow area around the robot takes on a non-circular shape due to the dual sensor placement, along with a significantly lower vertical resolution.

The specific environment for conducting experiments was the Barcelona Robot Lab, chosen for its challenging terrain featuring strong slopes and multi-level areas. This environment provided valuable insights and served as a testing ground preceding two Logismile demonstrations: one in Esplugues de Llobregat near Barcelona (Fig. 7) and the other in the Hungarian city of Debrecen (Fig. 8).

Throughout these experiments, our method for terrain classification consistently demonstrated robust performance, successfully navigating complex urban landscapes and covering several kilometers without encountering issues in obstacle detection. However, we observed a significant challenge stemming from the sensor's low vertical resolution and extensive range. In situations with negative slopes, the ground points reflected in the LiDAR data became progressively distant, making it challenging for the algorithm to continue data exploration. This occurred because the Region of Interest (ROI) size was insufficient to reach these distant points, and expanding the ROI indefinitely was not a feasible solution as it would undermine the local data analysis approach. Consequently, some remote obstacles remained unlabeled and were subsequently discarded, which adversely affected localization algorithms.

To address this issue and given our limited sensor resolution, we adopted an approach that classified the lowest point in each unanalyzed cell as the ground and categorized the remaining points in the cell based on a simple height threshold relative

TABLE V
MEAN VALUES - *TERRAIN* AND *VEGETATION* ARE NOT TRAVERSABLE

Method	P μ	R μ	F1 μ	Acc μ	IoU μ	KOR μ	T μ
TRAVEL [35]	73.64	**98.15**	83.92	85.60	72.63	97.37	16.08
GATA w/o NN	77.70	94.33	85.03	87.33	74.26	97.78	**11.65**
GATA VEG + TER	**91.92**	94.21	**92.99**	**94.65**	**86.98**	**98.66**	71.39

TABLE VI
STANDARD DEVIATIONS - *TERRAIN* AND *VEGETATION* ARE NOT TRAVERSABLE

Method	P σ	R σ	F1 σ	Acc σ	IoU σ	KOR σ	T σ
TRAVEL [35]	07.59	**00.68**	05.15	05.94	07.43	00.48	**00.91**
GATA w/o NN	07.50	01.79	04.84	05.29	07.04	00.30	02.21
GATA VEG + TER	**03.17**	02.72	**02.21**	**01.96**	**03.67**	**00.28**	10.85

to this lowest point. This modification enabled us to utilize the sensor's full range without contaminating the data used for localization with ground points and without sacrificing too much information about distant obstacles.

Another issue we encountered in urban environments involved surfaces that, while not classified as obstacles, were unsuitable for traversal in the context of our autonomous delivery application. These included areas like grass or unpaved terrain. This observation spurred the development of traversability analysis using the Shallow Neural Network that is detailed in this paper.

V. CONCLUSIONS AND FUTURE WORK

In this study, we presented GATA, a probabilistic, graph-based algorithm for real-time terrain analysis. GATA processes 3D point cloud data to generate a probabilistic Ground Model, enabling rapid and reliable obstacle detection. For applications requiring a more detailed representation of ground surfaces, we integrated a shallow neural network to classify ground points based on traversability, using features derived from the Ground Model.

Our quantitative evaluation on the SemanticKitti dataset demonstrated the superiority of our method over existing approaches. In its simplest implementation, our method outperformed the competition in all evaluated metrics (Precision, F1, Accuracy, IoU, Key Obstacles Recall, and execution time), except for Recall. When the shallow neural network was employed, the system kept meeting the real-time requirements and the improvements were even more significant, reaching an IoU increase of 14 points when TERRAIN and VEGETATION classes were deemed non-traversable, and 25 IoU points when only ROAD class was considered traversable.

Furthermore, we seamlessly integrated GATA into a real-world last-mile delivery robot and conducted a series of experiments and demonstrations. Regardless of the sensor resolution or configuration, GATA consistently demonstrated robust performance across diverse environments. This highlights GATA's value as a user-friendly, real-time terrain analysis solution, particularly for robots with limited processing capabilities.

Looking ahead, we plan to explore the integration of our method with Deep Learning techniques to develop semantically rich and adversarial-resistant point cloud segmentation systems. We also aim to harness the Ground Model produced by GATA, which inherently encodes traversability relationships between nodes, for trajectory planning in dynamic environments. These research directions open up exciting opportunities for further advancements in the field of Intelligent Vehicles.

REFERENCES

[1] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall, "SemanticKITTI: A dataset for semantic scene understanding of LiDAR sequences," in *Proc. IEEE Int. Conf. Comput. Vis.*, Seoul, 2019, pp. 9297–9307.

[2] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuScenes: A multimodal dataset for autonomous driving," in *Proc IEEE Conf. Comput. Vis. Pattern Recognit.*, Virtual, 2020, pp. 11 621–11 631.

[3] X. Yan, J. Gao, C. Zheng, C. Zheng, R. Zhang, S. Cui, and Z. Li, "2DPASS: 2D priors assisted semantic segmentation on LiDAR point clouds," in *Proc. Eur. Conf. Comput. Vis*, Tel Aviv, 2022, pp. 677–695.

[4] A. Bar, J. Lohdefink, N. Kapoor, S. J. Varghese, F. Huger, P. Schlicht, and T. Fingscheidt, "The vulnerability of semantic segmentation networks to adversarial attacks in autonomous driving: Enhancing extensive environment sensing," *IEEE Signal Processing Mag.*, vol. 38, no. 1, pp. 42–52, 2020.

[5] P. Pfaff, R. Triebel, and W. Burgard, "An efficient extension to elevation maps for outdoor terrain mapping and loop closing," *Int. J. Robotics Res.*, vol. 26, no. 2, pp. 217–230, 2007.

[6] À. Santamaria-Navarro, E. H. Teniente, M. Morta, and J. Andrade-Cetto, "Terrain classification in complex three-dimensional outdoor environments," *J. Field Robotics*, vol. 32, no. 1, pp. 42–60, 2015.

[7] Y. Qian, X. Wang, Z. Chen, C. Wang, and M. Yang, "Hy-Seg: A hybrid method for ground segmentation using point clouds," *IEEE Trans. Intell. Veh.*, vol. 8, no. 2, pp. 1597–1606, 2023.

[8] W. Huang, H. Liang, L. Lin, Z. Wang, S. Wang, B. Yu, and R. Niu, "A fast point cloud ground segmentation approach based on coarse-to-fine Markov random field," *IEEE Trans. Intell. Transport. Syst.*, vol. 23, no. 7, pp. 7841–7854, 2022.

[9] J. Xu, R. Zhang, J. Dou, Y. Zhu, J. Sun, and S. Pu, "RPVNet: A deep and efficient range-point-voxel fusion network for LiDAR point cloud segmentation," in *Proc. IEEE Int. Conf. Comput. Vis.*, Montreal, 2021, pp. 16 024–16 033.

[10] Y. Zhu, C. Miao, F. Hajiaghajani, M. Huai, L. Su, and C. Qiao, "Adversarial attacks against LiDAR semantic segmentation in autonomous driving," in *Proc. ACM Conf. Embed. Networked Sens. Syst.*, Coimbra, 2021, pp. 329–342.

[11] N. O'Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. V. Hernandez, L. Krpalkova, D. Riordan, and J. Walsh, "Deep learning vs. traditional computer vision," in *Proc. 2019 Comp. Vis. Conf. (CVC), Volume 1 1*. Springer, 2020, pp. 128–144.

[12] Y. Li and J. Ibanez-Guzman, "Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems," *IEEE Signal Processing Mag.*, vol. 37, no. 4, pp. 50–61, 2020.

[13] V. Vaquero, I. del Pino, F. Moreno-Noguer, J. Solà, A. Sanfeliu, and J. Andrade-Cetto, "Dual-branch CNNs for vehicle detection and tracking on LiDAR data," *IEEE Trans. Intell. Transport. Syst.*, vol. 22, no. 11, pp. 6942–6953, 2021.

[14] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Conf. Robot Learning*. PMLR, 2017, pp. 1–16.

[15] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[16] Y. Hou, X. Zhu, Y. Ma, C. C. Loy, and Y. Li, "Point-to-voxel knowledge distillation for lidar semantic segmentation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit*, 2022, pp. 8479–8488.

[17] X. Xie, L. Bai, and X. Huang, "Real-time LiDAR point cloud semantic segmentation for autonomous driving," *Electronics*, vol. 11, no. 1, p. 11, 2022.

[18] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[19] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. D. Tygar, "Adversarial machine learning," in *Proc. 4th ACM workshop Security Artificial Intelligence*, 2011, pp. 43–58.

[20] D. J. Miller, Z. Xiang, and G. Kesidis, "Adversarial learning targeting deep neural network classification: A comprehensive review of defenses against attacks," *Proc. of the IEEE*, vol. 108, no. 3, pp. 402–433, 2020.

[21] J. Tu, M. Ren, S. Manivasagam, M. Liang, B. Yang, R. Du, F. Cheng, and R. Urtasun, "Physically realizable adversarial examples for lidar object detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit*, 2020, pp. 13 716–13 725.

[22] Y. Li, C. Wen, F. Juefei-Xu, and C. Feng, "Fooling LiDAR perception via adversarial trajectory perturbation," in *Proc. IEEE Int. Conf. Comput. Vis.*, Montreal, 2021, pp. 7898–7907.

[23] B. Douillard, J. Underwood, N. Melkumyan, S. Singh, S. Vasudevan, C. Brunner, and A. Quadros, "Hybrid elevation maps: 3d surface models for segmentation," in *2010 IEEE/RSJ Int. Conf. Intell. Robots and Systems*. IEEE, 2010, pp. 1532–1538.

[24] M. Himmelsbach, F. V. Hundelshausen, and H.-J. Wuensche, "Fast segmentation of 3d point clouds for ground vehicles," in *2010 IEEE Intell. Vehicles Symp.* IEEE, 2010, pp. 560–565.

[25] M. Zhang, D. D. Morris, and R. Fu, "Ground segmentation based on loopy belief propagation for sparse 3d point clouds," in *2015 Int. Conf. 3D Vis.* IEEE, 2015, pp. 615–622.

[26] C. Reymann and S. Lacroix, "Improving LiDAR point cloud classification using intensities and multiple echoes," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Hamburg, 2015, pp. 5122–5128.

[27] I. Bogoslavskyi and C. Stachniss, "Efficient online segmentation for sparse 3d laser scans," *PFG–J. Photogrammetry, Remote Sensing, and Geoinformation Science*, vol. 85, pp. 41–52, 2017.

[28] D. Zermas, I. Izzat, and N. Papanikolopoulos, "Fast segmentation of 3D point clouds: A paradigm on LiDAR data for autonomous vehicle applications," in *Proc. IEEE Int. Conf. Robotics Autom.*, Singapore, 2017, pp. 5067–5073.

[29] Z. Luo, M. V. Mohrenschildt, and S. Habibi, "A probability occupancy grid based approach for real-time LiDAR ground segmentation," *IEEE Trans. Intell. Transport. Syst.*, vol. 21, no. 3, pp. 998–1010, 2019.

[30] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[31] P. Narksri, E. Takeuchi, Y. Ninomiya, Y. Morales, N. Akai, and N. Kawaguchi, "A slope-robust cascaded ground segmentation in 3d point cloud for autonomous vehicles," in *2018 21st Int. Conf. Intell. Transport. Syst. (ITSC)*. IEEE, 2018, pp. 497–504.

[32] A. Petrovskaya and S. Thrun, "Model based vehicle detection and tracking for autonomous urban driving," *Auton. Robots*, vol. 26, no. 2, pp. 123–139, 2009.

[33] H. Lim, M. Oh, and H. Myung, "Patchwork: Concentric zone-based region-wise ground segmentation with ground likelihood estimation using a 3D LiDAR sensor," *IEEE Robotics Autom. Lett.*, vol. 6, no. 4, pp. 6458–6465, 2021.

[34] Z. Zhu and J. Liu, "Graph-based ground segmentation of 3d lidar in rough area," in *Intl. Conf. Tech. Practical Robot Applications (TePRA)*, 2014, pp. 1–5.

[35] M. Oh, E. Jung, H. Lim, W. Song, S. Hu, E. M. Lee, J. Park, J. Kim, J. Lee, and H. Myung, "TRAVEL: Traversable ground and above-ground object segmentation using graph representation of 3D LiDAR scans," *IEEE Robotics Autom. Lett.*, vol. 7, no. 3, pp. 7255–7262, 2022.

[36] J. Andrade-Cetto, "The Kalman filter," Institut de Robòtica i Informàtica Industrial, Universitat Politècnica de Catalunya, Tech. Rep. IRI-DT-02-01, March 2002.

[37] A. Garrell, M. Villamizar, F. Moreno-Noguer, and A. Sanfeliu, "Proactive behavior of an autonomous mobile robot for human-assisted learning," in *Proc. Int. Conf. Robot Hum. Interact. Commun.*, Gyeongju, 2013, pp. 107–113.

[38] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the KITTI vision benchmark suite," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Providence, RI, 2012, pp. 3354–3361.

Iván del Pino is a Margarita Salas postdoctoral Fellow at IRI. He holds a BS, MS, and PhD degrees in Engineering from Univ Alicante. His research efforts are on mobile autonomous robotic systems.

Àngel Santamaria-Navarro is Lecturer at UPC's Automatic Control Department. Àngel holds MS Eng and PhD degrees from UPC. He was a finalist for the Georges Giralt Best PhD Award. Prior to joining UPC as a lecturer, he held postdoc and Robotics Research Technologist appointments at NASA's JPL. His current interests are on perception and control of mobile robotic systems.

Anaís Garrell Zulueta is Lecturer at UPC's Automatic Control Department. She holds a BS in Math from UB and a PhD from UPC (FPU Scholar, runner-up Best Spanish Robotics PhD Thesis). Her current research is focused on human acceptance of robotics technologies and human-robot interaction.

Fernando Torres is a Full Professor at the Systems Eng Dept. and director of the AUROVA Research Group at the Univ Alicante. He holds BS and PhD degrees in Industrial Eng from UPM. His group carries out research on robotic manipulation, visual servo, robotic perception, neurorobotics, field robotics, computer vision, and e-learning.

Juan Andrade-Cetto is a Research Scientist and former Director of the Institut de Robòtica i Informàtica Industrial, CSIC-UPC. He is the PI of UPC's Robotics and Artificial Intelligence (RAIG) Research Group. Juan holds a BSEE from CETYS Univ, an MSEE from Purdue Univ. (Fulbright Scholar), and a PhD from UPC (EURON Georges Giralt Best PhD Award). His research work is in the areas of perception and state estimation for robotics.