Real-time elevation mapping with Bayesian ground filling and traversability analysis for UGV navigation

Han Xie, Xunyu Zhong, Bushi Chen, Pengfei Peng, Huosheng Hu, and Qiang Liu

***Abstract*— Unmanned ground vehicles (UGVs) require effective perception and analysis of their surrounding terrain for safe operation. This paper presents a novel approach to their local elevation mapping and traversability analysis using sparse data from a single LiDAR sensor, which can generate a dense local traversability map in real-time. By preserving ground height information, our method can differentiate between vertical obstacles, suspended objects and other terrains in the elevation map. The modified Bayesian generalized kernel elevation inference is utilized to predict and fill in sparse elevation maps to achieve local dense terrain traversability mapping. The system uses GPU parallel processing to accelerate calculations, ensuring real-time perception and dynamic processing. The proposed system was tested in both structured and unstructured environments, and achieved better performances in map filling and handling of suspended and vertical objects compared to other existing approaches.**

I. INTRODUCTION

As unmanned ground vehicles (UGVs) will operate on increasingly varied and complex terrain, effective terrain traversability mapping becomes a critical factor for their safe navigation. LiDAR sensors have been widely deployed due to their advantages of high sensing accuracy and low lighting tolerance over visual sensors. Several methods have been proposed to convert LiDAR data into terrain data for traversability analysis, including point cloud [1], 3D grid [2], and 2.5D elevation mapping [3] methods.

The point cloud mapping method requires a large amount of data and computation for neighborhood search processing. 3D grid map can reduce the amount of data but is limited in traversability analysis due to limited elevation accuracy [4] and sparse ground data. Moreover, both point cloud mapping and 3D grid map can be combined with sample-based planning methods to reduce computational costs [1] [5]. However, these methods may not be suitable for search-based planning methods that require a dense traversability map.

This work was supported in part by the Equipment Advance Research Foundation of China under Grant 61405180205.*(Corresponding author: Xunyu Zhong.)*

Han Xie, Xunyu Zhong, Bushi Chen, and Pengfei Peng are with the Department of Automation, School of Aerospace Engineering, Xiamen University, Xiamen, Fujian 361102, China(e-mail: xiehan@stu.xmu.edu.cn and zhongxunyu@xmu.edu.cn)

Huosheng Hu is with the School of Computer Science and Electronic Engineering, University of Essex, CO4 3SQ Colchester, U.K. (e-mail: hhu@essex.ac.uk)

Qiang Liu is with the School of Engineering Mathematics and Technology, University of Bristol, BS8 1TW, Bristol, U.K. He is also an honorary research scientist in the Department of Psychiatry, University of Oxford. (e-mail: qiang.liu@bristol.ac.uk)

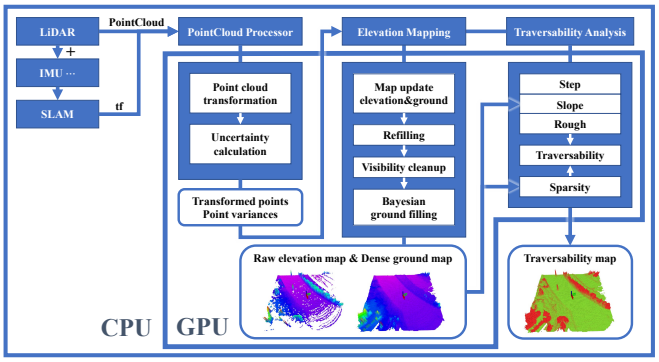

Fig. 1. Overview of traversability mapping processing. Most of data transmission is processed on CPU, and the processing of the point cloud and map data is carried out on GPU. The square boxes represent processing modules, and the rounded boxes represent the output of each module. The white arrows indicate the sequence of processes within a module, and the blue lines with arrows represent data transfer between modules. The overall process is divided into three main parts: point cloud processing, map processing, and traversability analysis.

Thanks to the continuity of elevation and the 2D grid-based distribution of map data, an elevation map not only reduces the amount of data while ensuring the accuracy of terrain elevation but also simplifies the operation of mapping sub-areas in traversability analysis. However, current elevation map methods have some limitations, such as the 2D distribution of data making it difficult to handle suspended objects [6], the vertical terrain information missing during traversability analysis [7], the map gaps occurring due to data sparsity and terrain occlusion, and the elevation map that is not directly applicable for navigation on UGVs.

In this work, we propose a real-time 2.5D elevation mapping method that utilizes Bayesian inference filling and traversability analysis for UGV navigation, which is shown in Fig. 1. Our work makes the several contributions. Firstly, the ground elevation attribute is introduced in an elevation map to process hanging obstacles and separate vertical obstacles from sloped terrain in the traversability analysis, resulting in more accurate and realistic terrain traversability representation. Secondly, the Bayesian elevation inference approach is deployed with line search to efficiently fill the gaps in the sparse map, including a re-inference method to address residual inference errors. Thirdly, a feature-based method is adopted for traversability analysis so that different terrain features can be calculated by elevation or ground elevation, resulting in traversability values that better match the terrain in terms of location. A GPU is used to realize the real-time performance of traversability mapping.

The rest of the paper is organized as follows. Section II reviews some related work in terms of elevation mapping, map filling, and terrain traversability analysis (TTA). The proposed approach is outline in Section III, including three parts: PointCloud processor, elevation mapping, and traversability analysis. Experimental results are given in Section IV to show the good performance of our proposed novel approach compared with other state-of-the-art methods. Finally, a brief conclusion and future work are given in Section V.

II. RELATED WORK

This section reviews some related work in elevation mapping, map filling, and terrain traversability analysis.

A. Elevation Mapping

Some elevation mapping methods have been developed from the GridMap library [8] that provides a well-established map data type. Methods in [3] and [9] form a baseline for elevation mapping on CPU devices. Elevation mapping usually uses feature-based traversability analysis [10], but the system suffers from low efficiency on CPU devices and fails to perform in real time. To solve this problem, [11] and [12] migrated sensory data and map data processing to GPU devices to accelerate calculations and reduce the time of local traversability mapping.

Previous methods updated the elevation map with the highest elevation in each cell. This approach is unable to describe vertical terrain within a cell, resulting in the inflation of impassable areas around vertical obstacles. In addition, sparse sensory data often generate gaps in the map, which are difficult to be used in traversability analysis. Our method aims to solve these problems.

B. Map Filling

To solve the filling problem of elevation maps, a mean filter or a median filter with edge-preserving capabilities was deployed in [13] to calculate the unknown value in the center cell by using the elevation information in the neighborhood. It can preserve steep terrain edges, and was simple to operate and efficient, However, it did not make full use of the prior information and cannot reflect the trend of the actual terrain. Moreover, after several iterations, the filled terrain may be too flat compared to the actual terrain, leading to the risk of terrain misjudgment.

As elevation maps can be treated as images, image processing methods have been applied to restore missing information in maps. The OpenCV library [14] offers two inpainting methods. [15] employed the Navier-Stokes fluid dynamics equation and [16] used the image gradient propagation smoothing estimator to predict the missing parts by diffusing image changes from the edges, which can effectively reconstruct map gaps caused by data sparsity.

However, these methods are only suitable for predicting small gaps when most of the map information is present. For terrain gaps caused by terrain occlusion, the filling quality decreases with insufficient prior information. In contrast, the work in [17] focused on the negative estimation of map gaps by using the lowest elevation of the gap edge, providing direct risk awareness of the concave terrain for traversability analysis. However, this approach can result in the over steep filling of some terrains.

Several learning-based methods have been proposed for exploring hidden terrain information to fill sparse maps. Shan et al. proposed [18] to use Bayesian sparse kernel [19] inference to predict elevation and traversability. However, the accumulated inference errors may lead to unreliable results in some extremely sparse areas. [20] used the potential occluded areas in dense areas to train a neural network to fill sparse areas and achieved good reconstructions on terrain occlusion. However, reconstructions of non-occlusion areas often resulted in over rough terrain. [21] used the Bayesian-GAN model to learn structured urban terrain samples, and achieved good reconstructions on occlusion, sparseness, and concave terrains in structured environment. However, this method required consistency between training and test environments, and was difficult to apply in unstructured environments.

We noticed that map gaps caused by terrain have a larger range than those caused by data sparsity in the sensing direction. Inspired by [17] and [18], we performed map filling using Bayesian sparse kernel inference with line search, and used new sensory data to update unreliable inference results, preventing the accumulation of inference errors.

C. Terrain Traversability Analysis

Terrain traversability analysis is an important part of environmental perception, providing an essential foundation for navigation. For 2.5D or 3D representations of an environment, the feature-based method is the most used one. This method calculates terrain features of map cells using the neighborhood terrain, and determines traversability based on these features. Three features, namely step, roughness, and slope, were used in [7] and [22] to describe terrain risk of map cells, in which thresholds and weights were used to fuse features and calculate traversability. [5] sampled points in the point cloud and searched for a fitting plane by calculating the slope, flatness, and sparsity of the plane with the map data in the neighborhood and fused these features to obtain traversability. [23] introduced risk features related to robot motion and established a multi-layer risk map using CVaR fusion to obtain a traversability map.

The feature-based method has strong flexibility, allowing for easy design of the feature composition according to the robot's structure. However, the distribution of traversability is affected by the radius of the neighborhood during feature calculation, which often results in over obstacle inflation. Moreover, since the method involves the operation of all map data, a large amount of calculation is required.

Deep learning methods were deployed in [24]–[26] to directly predict traversability from the elevation map. Supervised methods fix the quantification standard of traversability when training the neural network, making it difficult to adjust for different UGVs. Furthermore, the supervised labels are subjective and limited by annotation. On the other hand,

unsupervised methods require many interactive attempts, resulting in high learning costs and difficulty in application.

Our method uses feature-based traversability analysis that employs a dual-layer elevation map incorporating both raw and ground elevation data. By calculating slope, roughness, and step features corresponding to the respective elevation layers, we can subdivide the traversability of vertical and ground terrain. By adjusting the thresholds and weights for each feature, our approach can be applied on different robots. To reduce the calculation time, terrain feature extraction in traversability analysis is performed on GPU devices.

III. METHOD

Fig. 1 shows the framework of our local traversability mapping system. The system takes the point cloud data from the sensor and the robot pose estimation data given by the localization system as input, and outputs a local elevation map with traversability. In the specific design, the point cloud data is provided by LiDAR, and the position and pose data can be provided by various localization systems, such as LIO-SAM [27], FAST-LIO2 [28] and LIO-Fusion [29].

Firstly, the system uses the transformation information provided by the SLAM system to transform the point cloud data from the sensor frame to the map frame and calculate the uncertainty of each point using the noise model. Next, the transformed point cloud is mapped to a grid map that moves with the robot, and the elevation and ground elevation of each cell are updated. If any filled cells are found to be unreliable, the updated map is used to perform re-filling and updates these cells. Subsequently, ray tracing is used to visibility clean up dynamic objects in cells that have not been updated by sensory data, and Bayesian sparse kernel inference with line search is used to fill ground gaps in the sparse map. After processing the elevation map, the geometric traversability and sparsity of each cell are calculated, resulting in a dense local traversability map.

A. PointCloud Processor

1) Point cloud transformation: We obtain raw point cloud data in the sensor frame and use the localization system to obtain the transform matrix $\boldsymbol{T}$ that maps from the sensor frame to the map frame. For point $\boldsymbol{p}_s = (x_s, y_s, z_s)$, we transform it into $\boldsymbol{p}_m = (x_m, y_m, z_m)$ in the map frame by the formula:

$$\begin{bmatrix} \boldsymbol{p}_m \\ 1 \end{bmatrix} = \boldsymbol{T} \begin{bmatrix} \boldsymbol{p}_s \\ 1 \end{bmatrix} \tag{1}$$

During the map update process, the coordinate (x_m, y_m) of $\boldsymbol{p}_m$ is mapped to grid positions, and the value z_m of $\boldsymbol{p}_m$ is used to update the elevation and ground elevation of the corresponding cell.

2) Uncertainty calculation: Because of the measurement error between sensory data and actual terrains, the uncertainty of each point in the transformed point cloud needs to be considered. Our approach is like the methods used in [30] and [9], where the measured points are viewed as a Gaussian probability distribution model $p \sim N(p, \sigma_p^2)$, with p represents the elevation of the point in the map frame.

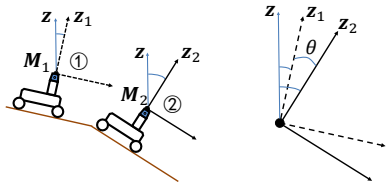

Fig. 2. Simplified representation of pose change angle calculation. Left shows the pose estimation of two adjacent frames and their z-axis vector $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ in the map frame, where $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ represent the corresponding rotation matrix. Right shows the pose change angle θ between $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$.

We simplify the noise model in [12] and calculate sensor noise $\sigma_s^2 = (c_{beam} + a_{beam} l)^2$, where c_{beam} represents the beam constant in the laser noise model described in [9], and a_{beam} is the beam angle parameter of the model. l donates the Euclidean distance between the point and the origin of the sensor frame.

In addition to the uncertainty of the sensor itself, the uncertainty from pose estimation also needs to be considered. Fig. 2 shows an example where two consecutive sensory data are obtained, and the corresponding rotation matrices $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ are obtained from the localization system. The z-axis direction vectors $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ of these two poses in the map frame can be calculated, and the angle θ between them can be determined. The uncertainty σ_t^2 from pose change can be obtained as follows:

$$\boldsymbol{z} = (0, 0, 1) \quad \boldsymbol{z}_1 = \boldsymbol{M}_1 \boldsymbol{z} \quad \boldsymbol{z}_2 = \boldsymbol{M}_2 \boldsymbol{z} \tag{2}$$

$$\theta = \arccos \frac{\boldsymbol{z}_1 \cdot \boldsymbol{z}_2}{|\boldsymbol{z}_1|\,|\boldsymbol{z}_2|} \tag{3}$$

$$\sigma_t^2 = (\lambda_t l \sin \frac{\theta}{2})^2 \tag{4}$$

where the non-negative hyperparameter λ_t is used for scaling, and the value l represents the distance between the point and the origin of the sensor frame.

By taking into account the uncertainties in sensor noise and pose changes, the uncertainty for each measurement point can be calculated as $\sigma_p^2 = \sigma_s^2 + \sigma_t^2$. The larger value between σ_s and σ_t prevails in σ_p, and the setting of λ_t can adjust the threshold of θ for the dominance shifting. We set $\lambda_t = 1$ for direct comparison between sensor noise and pose change in experiments.

B. Elevation Mapping

1) Map update: To update the elevation map, we use a method similar to [30] and [31]. We compute the Mahalanobis distance d between the sensory data elevation p_s and the map elevation $\hat{h}^-$, compare it to a threshold c, and use a Kalman filter to fuse nearby elevation measurements and variances. This allows us to obtain the new elevation $\hat{h}^+$ and variance σ_h^{2+} of the map cell. The updating rules are as follows:

$$d = \frac{|p_z - \hat{h}^-|}{\sigma_h^-} \tag{5}$$

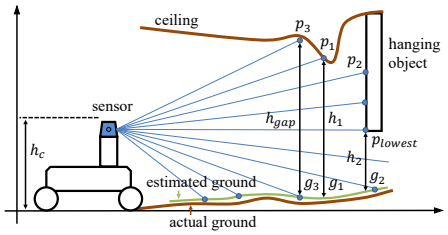

Fig. 3. Hanging objects handling. The brown curve represents the actual ground and ceiling in the environment, while the green curve indicates the filled ground elevation. The blue points represent the measure points from the sensor, and the black rectangle denotes a hanging obstacle.

$$\hat{h}^+ = \begin{cases} \dfrac{\sigma_p^2 \hat{h}^- + \sigma_h^{2-} p_z}{\sigma_h^{2-} + \sigma_p^2} & , d \le c \\ p_z & , d > c \wedge p_z > \hat{h}^- \end{cases} \tag{6}$$

$$\sigma_h^{2+} = \begin{cases} \dfrac{\sigma_h^{2-} \sigma_p^2}{\sigma_h^{2-} + \sigma_p^2} & , d \le c \\ \sigma_p^2 & , d > c \wedge \hat{h}^- \le p_z \end{cases} \tag{7}$$

where $\hat{h}^-$ and σ_h^{2-} represent the older elevation and variance of the map cell, and σ_p^2 is the variance of the point we calculate in point cloud processing.

Note that the lower elevation discarded above contains some terrain information, such as vertical terrain and ground elevation, which has a significant impact on hanging object processing and vertical terrain analysis. Therefore, we propose the ground elevation attribute of the elevation map to save the lowest elevation data.

Updating ground elevation is like the original method, but we discard higher elevations during distance comparison. Since the variance of the map cell is primarily influenced by the distance model and elevation updates are more frequent than ground elevation updates, we update the variance of the map cell with the latest updated variance.

During map updating, we handle hanging objects by comparing sensory data with existing ground elevation. As shown in Fig. 3, an elevation threshold h_c is set for robot passage. If the sensor point p_1 is the lowest point in the map cell, we calculate the height difference h_1 between p_1 and the ground elevation g_1 in this cell. If $h_1 > h_c$, p_1 is considered a passable hanging object, and the elevation of the cell is updated to g_1. When the lowest point p_{lowest} in the cell has a height difference $h_2 < h_c$ with the ground elevation g_2, the object is regarded as an obstacle and the elevation is updated by the highest point p_2. Additionally, when ground elevation g_3 is updated by current sensor points, we check the height difference $h_{gap} = p_3 - g_3$ between the two adjacent data points p_3 and g_3 in the same cell. If $h_{gap} > h_c$, the point p_3 is discarded and the elevation is updated by g_3.

2) Visibility cleanup: The ray tracing method is used to clear dynamic objects for the map cells that have not been updated by the latest sensory data. We use the Bresenham algorithm [32] to search for map cells in the ray direction of the sensor. As shown in Fig. 4, for the map cell p_{old} with the elevation h_{old} has not been updated, we search for the first lowest sensor point p_{lowest}, and calculate the elevation change between p_{lowest} and the sensor height h_{sensor} to obtain the elevation threshold h_t. If $h_{old} > h_t$, we update h_{old} to ground elevation g_p, and the cell p_{old} to p_{new}.

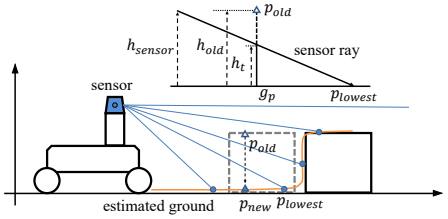

Fig. 4. Ray tracing visibility cleanup. The dashed rectangle represents the previous location of the object, while the solid rectangle represents its current location. The blue point represents the current sensory data point, and the orange curve represents the filled ground elevation. The simplified diagram above shows the calculation of the threshold h_t.

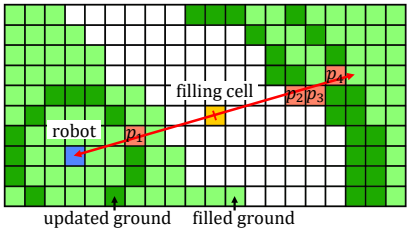

Fig. 5. Bidirectional line search method. The blue grid shows the location of robot's sensor, and the yellow grid indicates the filling cell. Dark green grids represent the ground height grid updated by sensory data, and light green grids indicate the previously filled grid. The red lines with an arrow show the search direction starting from the filled grid, and the red grids represent the samples obtained.

While performing visibility cleanup, our method uses ground elevation to correct the elevation, which reduces map gaps in [9] and [11].

3) Bayesian ground filling: For each map cell without ground elevation, we also apply the line search method to sample data for filling. Starting from the filling cell, we search towards the direction of the sensor for a certain distance to obtain the first sample set $\mathcal{D}_1 = \{(x_i, y_i)_{i=1:N_1}\}$, and search in the opposite direction to obtain another sample set $\mathcal{D}_2 = \{(x_i, y_i)_{i=1:N_2}\}$. For each sample $p_i = (x_i, y_i)$, y_i represents the ground elevation updated by sensory data, and x_i represents the distance between the filling cell and the sampled cell. Fig. 5 shows the search method with 6 grids. Different from [18], our method uses line search instead of neighborhood search to acquire a larger range of sampling. Moreover, each filling cell uses the valid ground elevation to reduce the error accumulation caused by inference iteration.

Then we apply Bayesian sparse kernel inference [33] to the sample set to obtain height predictions. The expected elevation μ^* in each direction is calculated with the corresponding sample set $\mathcal{D}$ as follows:

$$E\left[\mu^* | \lambda, \mathcal{D}, x^*\right] = \frac{\lambda \mu_0 + \sum_{i=1}^{N} k(x_i, x^*) y_i}{\lambda + \sum_{i=1}^{N} k(x_i, x^*)} \tag{8}$$

where $k(x_i, x^*)$ is the kernel function used in Bayesian

inference, (x_i, y_i) represents the data in the sample set $\mathcal{D}$, μ_0 is the prior elevation, and λ is the non-negative confidence parameter of μ_0, which servers as the weight source in calculation. The larger λ is, the closer μ^* is to μ_0.

Since map gaps are typically caused by unobserved lower terrain, we take the minimum elevation in $\mathcal{D}$ as μ_0. Based on the effectiveness of the sparse kernel in elevation inference demonstrated in [18], the sparse kernel [19] is used as the kernel function. Then we set $\lambda = 1$, which is the maximum value of the sparse kernel $k(x_i, x^*)$, and achieved balanced and good performances in our structured and unstructured environments tests.

For cells that obtained no sample set, we do not process them. For cells that only obtained one sample set, the predicted elevation μ^* is directly used as the ground elevation of the cell but is considered unreliable. For cells that have obtained two sample sets $\mathcal{D}_1$ and $\mathcal{D}_2$, with the predicted elevation μ_1^* and μ_2^*, we calculate the final predicted elevation μ^* as follows:

$$\mu^* = \begin{cases} \dfrac{(n_1+n_2)\mu_1^* + n_2\mu_2^*}{n_1+2n_2} & ,\mu_1^* \le \mu_2^* \\ \dfrac{n_1\mu_1^* + (n_1+n_2)\mu_2^*}{2n_1+n_2} & ,\mu_1^* > \mu_2^* \end{cases} \tag{9}$$

where n_1 and n_2 represent the number of gap cells before obtaining the sample sets $\mathcal{D}_1$ and $\mathcal{D}_2$. And the filled cell with is considered reliable.

4) Refilling: We designed a filled cell update step after the map update by sensory data and before the new filling operation. This step updates only the filled cells considered unreliable to correct possible residual errors in the predicted elevation.

For cells with only one sample set obtained or no sample set obtained, their inference results are considered unreliable. Therefore, we perform sample search and Bayesian elevation inference on these cells again, and update their predicted elevation using the new sensory data. For cells that have two sample sets obtained by bidirectional search in the previous filling operation and cells updated by sensory data, we consider their elevation to be highly reliable, and thus do not update them in the re-filling update step.

C. Traversability Analysis

We use the feature-based terrain traversability analysis, and select step, slope and roughness as the terrain features to calculate the traversability of the cell p. First, we search the neighborhood centered on cell p, and record all searched elevations as $\mathcal{H} = \{h_1, h_2, \ldots, h_i\}_{i=1:N_h}$, where N_h is the number of elevations. And we record all searched ground elevations as $\mathcal{G} = \{g_1, g_2, \ldots, g_i\}_{i=1:H_g}$, where H_g is the number of ground elevations.

1) Slope: To obtain the slope feature value C_{slope} , we use the ground elevation set $\mathcal{G}$ and the corresponding position set $\mathcal{P} = \{(x_i, y_i)_{i=1:H_g}\}$ to fit a plane, and calculate the normal vector of the plane using Singular Value Decomposition (SVD) [34]. Then, we use the angle θ_s between the plane normal vector and the unit z-axis vector in the map frame to calculate C_{slope} with a radian threshold θ_c as follows:

$$C_{slope} = \max(1 - \frac{\theta_s}{\theta_c}, 0) \tag{10}$$

2) Step: For the step feature, we use the mean value $\bar{g}$ and the minimum value g_{min} of $\mathcal{G}$ to calculate C_{step} with the elevation h_p of cell p:

$$C_{step} = \max(1 - \frac{2h_p - g_{\min} - \bar{g}}{2h_c}, 0) \tag{11}$$

where h_c is the step height threshold. If there is no elevation in cell p, we use ground elevation g_p instead of h_p.

3) Roughness: We use a similar calculation to [11] and use the mean value $\bar{h}$ of $\mathcal{H}$, and $\mathcal{G}$ for filled ground, to calculate C_{rough} with the roughness threshold r_c as follows:

$$C_{rough} = \max(1 - \frac{h_p - \bar{h}}{r_c}, 0) \tag{12}$$

4) Traversability: C_{step} mainly reflects changes in the vertical terrain, C_{slope} focuses on the changing trend of the ground, and C_{rough} shows the difference between the cell and the surrounding terrain. Then we fuse the three features to calculate the traversability value C_{traver} as follows:

$$C_{traver} = w_{step}C_{step} + w_{slope}C_{slope} + w_{rough}C_{rough} \tag{13}$$

where w_{step}, w_{slope} and w_{rough} represent the weights of features, and $w_{step} + w_{slope} + w_{rough} = 1$. If any feature value is 0, set C_{traver} to 0.

A traversability value of $C_{traver} = 1$ indicates free passage, while $C_{traver} = 0$ indicates impassable terrain. By utilizing two elevation layers, the traversability value corresponds more accurately to the cell where the terrain changes.

5) Sparsity: For the filled cells with dense sensory data around but still unreliable and near the robot, we consider the possibility of concave obstacles or occluded terrain that cannot be perceived by sensors and are difficult to predict. In such scenarios, we calculate the feature $C_{sparsity}$ instead of geometric traversability, as follows:

$$C_{sparsity} = \begin{cases} \max(1 - \dfrac{h_s d_c}{h_c d_s}, 0) & ,h_s \ge h_c \\ \max(1 - \dfrac{h_s}{h_c}, 0) & ,h_s < h_c \end{cases} \tag{14}$$

where h_s represents the elevation difference between the valid ground elevations searched from bidirectional directions, and d_s is the distance between the filled cell and the robot, h_c is the step threshold, and d_c is the distance threshold. If only one or no elevation is obtained from line search, the traversability of this cell is considered unknown.

IV. EXPERIMENTAL RESULTS

We tested our method on several datasets and a physical robot. Our dataset recording platform consisted of a Robosense RS-LiDAR-16 laser scanner, an Xsens MTi 30-series IMU, and a Scout-mini UGV (a four-wheel differential mobile robot chassis) for capturing data in real environments.

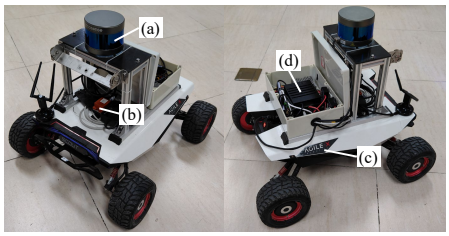

Fig. 6. Recording and testing platform. (a) Robosense RS-LiDAR-16, (b) Xsens MTi- 30-2A8G4 IMU, (c) Scout-mini UGV, (d) Nvidia AGX Xavier.

TABLE I
AVERAGE TIME CONSUMPTIONS

Step (ms)	CPU	CuCpp	Ours(PC)	Ours(AGX)
Points Process (to GPU)	0.3	1.5	3.3	6.1
Map Update (to GPU)	0.6	1.1	4.0	18.2
Cleanup (& Filling)	14.8	0.3	0.5	0.6
Traversability Analysis	123.1	1.1	1.4	2.4
Publish (to CPU)	0.1	4.7	4.4	6.2

We evaluated our method on a laptop PC (AMD Ryzen 7 5800H, Nvidia RTX 3070 Laptop) using the datasets.

We built the testing platform as Fig. 6 with the recording platform and an Nvidia AGX Xavier, and tested our method in real-world navigation tasks.

To obtain pose estimation and localization information, we used FAST-LIO2 [28], which can publish TF information in Robot Operating System (ROS).

A. Dataset Test

The datasets were collected from indoor and outdoor environments shown in Fig. 7, which included complex structured and unstructured terrains. The dataset published point cloud data at 20Hz and IMU data at 300Hz. We ran FAST-LIO2 synchronously to obtain real-time localization and pose estimation information during testing. The size of the local elevation map was uniformly set to 8m $\times$ 8m, with a resolution of 0.05m. We down sampled the point cloud data with a voxel size of 0.05m and only used points within the height range of $[-1.0\mathrm{m}, +1.5\mathrm{m}]$ in the sensor frame.

In our dataset tests, we compared the local traversability mapping performance with the CPU baseline method provided in [9], as well as the GPU baseline methods provided in [11] and [12], which we respectively referred to as CuCpp and CuPy. We also compared the map filling performance with the original method proposed in [18] and the filling method used in [12]. We set the weights for features as $w_{slope} = 0.4$, $w_{step} = 0.3$ and $w_{rough} = 0.3$. The CPU baseline method used the same features and weights to calculate the traversability using filters. The CuCpp method used two features, slope and roughness, with $w_{slope} = 0.5$ and $w_{rough} = 0.5$. The CuCy method used a pre-trained neural network to predicate traversability. The threshold values for individual features were set as $\theta_c = 0.6\mathrm{rad}$ for slope, $h_c = 0.2\mathrm{m}$ for step, and $r_c = 0.2\mathrm{m}$ for roughness.

Fig. 8 presents a comparison between our approach and other methods for generating traversability elevation maps of complex structured and unstructured environments.

Fig. 7. The actual environments of datasets. The environments comprise indoor and outdoor settings, with a variety of terrains, as well as suspended, vertical, and dynamic objects. The overview comes from Google Earth.

1) Suspended Objects Handling: As shown in Fig. 8 (a), (b) and (g), Our method could effectively remove passable unstructured hanging branches and structured suspended table in front of the robot with ground elevation and robot height, while other methods failed to handle these objects.

2) Terrain Traversability Analysis: Fig. 8 (c) and (d) shows narrow passable terrains consisting of vertical railings and vegetation. Compared to other methods, our method separated vertical objects from ground features in analysis, resulting in a more accurate and clear representation of traversable areas on the map. As for the unstructured and non-vertical objects, such as rocks in wild showed in 8 (e), our method considered them as terrains with risky slope and calculated traversability with neighborhood features. Such traversability analysis is more suitable for UGV to accurately determine the terrains with different risks.

3) Dynamic Object Handling: As shown in Fig. 8 (f) and (g), the CPU baseline method resulted in significant delays because of the long processing time on map, manifested as the absence and residue of dynamic obstacles on the map. Different from CuCpp, our method checked the visibility of all cells and reconstructed the cleared ground to reduce map gaps. Our updated terrain is closer to the latest point cloud, demonstrating better data synchronization than CuPy.

The time consumptions results are shown in TABLE I, and CuCy is not included due to its different program structure. Although the GPU methods incur additional time when copying memories between devices, our processing speed on map is much faster than the CPU baseline, and slightly slower than CuCpp due to additional processes.

4) Map Filling Comparison: A comparison of our method with the original Bayesian sparse kernel inference method (BGK) provided in [18], and the min-filter and inpaint methods used in [12] is shown in Fig. 9. Although our method cannot completely fill the map in some occluded environments, it was dense enough for traversability analysis and reduced over-filling errors. The BGK and inpaint methods were prone to misjudgment in sparse areas due to maintaining the trend of terrain changes, shown in Fig. 9 (a), (b) and (c), and the min filter tended to make overly conservative terrain predictions, leading to erroneous concave terrain shown in Fig. 9 (d). Benefiting from the line search inference, the propagation direction of our inference remained consistent

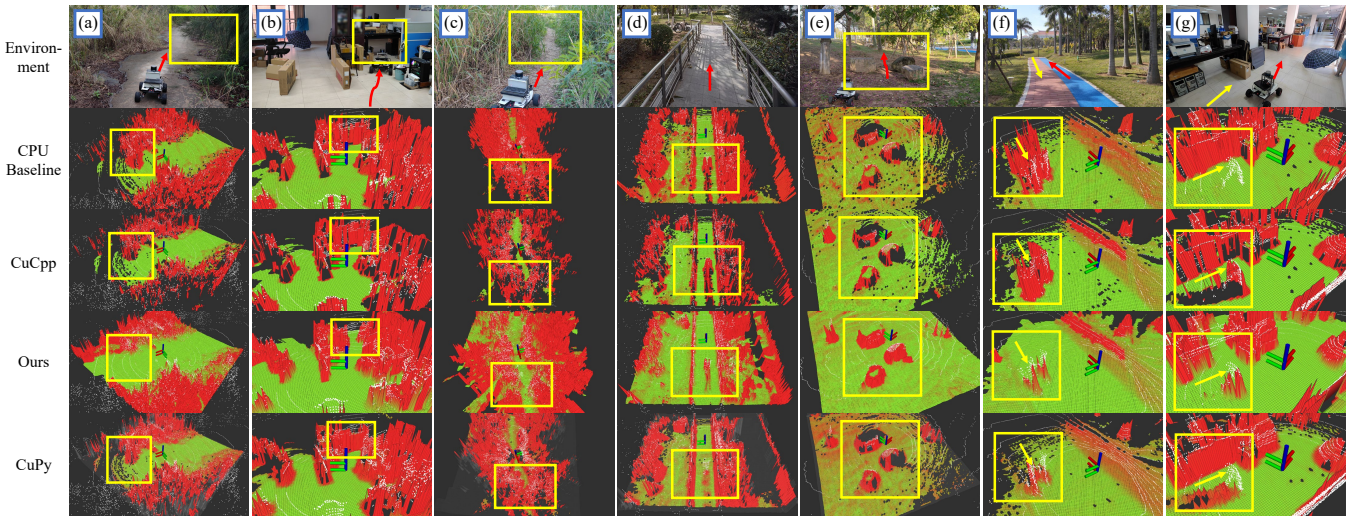

Fig. 8. Comparison of traversability mapping performances. The leftmost column shows the real environments of the dataset, other columns show the traversability mapping results. The small axis indicates the pose of the robot, and the white points are point cloud data. The yellow boxes indicate the special terrains and objects mentioned in the text. The arrows represent the motion directions of the robot (red) and dynamic objects (yellow). In the traversability maps, red represents impassable areas, green represents passable areas, and the color change from green to red indicates a decrease in traversability.

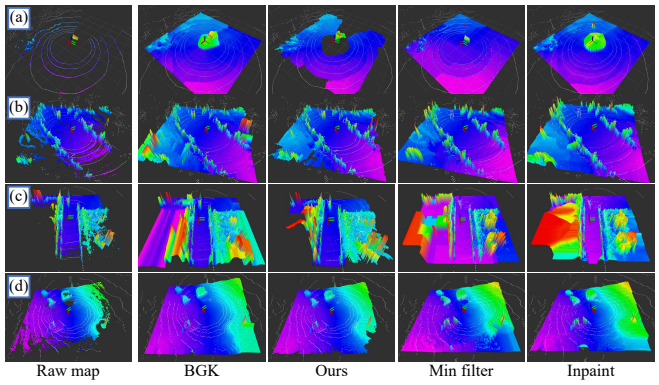

Fig. 9. Comparison of map filling performances. The first column on the left shows the sparse elevation map without filling. The color in iridescence order represrents the elevation, and the minimum elevation color is purple.

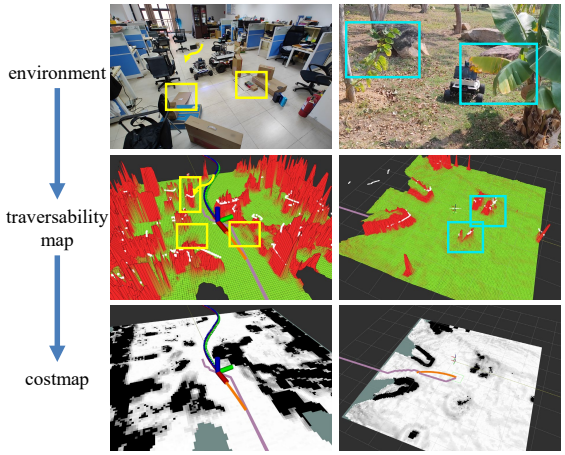

Fig. 10. Navigation in structured and unstructured environments. Left is indoor structured environment with obstacles, moving object (the yellow box with moving direction arrow) and risk terrains (yesllow boxes with a small 15° slope and 6.5cm boxs). Right Shows an unstructured environment with rough grass, trees, rocks and passable hanging object (blue boxes). The white points are the 2D point cloud at the plane of LiDAR. The small axis indicates the pose of the robot. The purple line is the global path and the orange line is the local path.

with the robot's motion, which makes our method suitable for different environments. Additionally, our method's filling results in sparse areas were more accurate than other methods due to the constraints on inference and the refilling method.

B. Navigation Test

We tested the real-time traversability mapping performance of our approach in structured and unstructured environments using the testing platform shown in Fig. 6. The point cloud filtering and map parameters settings remained as the dataset testing, while the traversability parameters were modified according to the platform's mobility performance, with slope threshold $\theta_c = 0.6\text{rad}$, step threshold $h_c = 0.15\text{m}$, and roughness threshold $r_c = 0.2\text{m}$.

We tested the navigation performance of our testing platform in unknown environments using the local traversability map. We converted the local traversability map into a 2D cost-map and used the ROS navigation stack for autonomous navigation. The path planning was performed using the A* algorithm in gloabl and the DWA method in local.

The real-time generated traversability maps and cost-maps are shown in Fig. 10. In the structured indoor environment, our method could accurately determine impassable obstacles with passable small slopes and boxes, update and clear dynamic object with a very small delay. In the unstructured environment, our method could accurately identify tree and rock obstacles in rough and sloped forest environments, and calculate traversability for passable rough terrain such as tree pits, and hanging branches above the robot. The 2D planning system can navigate through complex environments with the cost-maps generated from our traversability maps.

Our system generated traversability maps and cost-maps at

the same frequency (20Hz) of inputs on the onboard Nvidia AGX Xavier. The time consumptions result is shown in TABLE I, which meets the real-time performance requirement of low-speed dynamic terrain perception.

V. CONCLUSIONS

In this paper, we proposed a novel method to generate dense local traversability maps in real time. The ground elevation attribute is introduced to elevation maps, and line-search Bayesian elevation inference is used on ground elevation to make the traversability map denser and subdivided on terrains. Experimental results have shown that our method can find clearer travelable areas from different environments.

Our method has achieved good performance in complex environments. However, it faces challenges when dealing with soft objects such as traversable grass, and special weather conditions that affect sensors. Nonetheless, due to the compatibility of our method with point cloud inputs from multiple sensors and its extensibility in grid map layers, we have the potential to address these challenges by integrating different sensor data and non-geometric features in future work. For further improvement of terrain traversability analysis and navigation, we will focus on introducing proprioception and visual features into our system.

REFERENCES

[1] P. Krüsi, P. Furgale, M. Bosse and R. Siegwart, "Driving on point clouds: Motion planning trajectory optimization and terrain assessment in generic nonplanar environments," *Journal of Field Robotics*, vol. 34, no. 5, pp. 940-984, 2017.

[2] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous robots*, vol. 34, pp. 189-206, 2013.

[3] P. Fankhauser, M. Bloesch, C. Gehring, M. Hutter and R. Siegwart, "Robot-centric elevation mapping with uncertainty estimates," *Mobile Service Robot*, pp. 433-440, 2014.

[4] Y. Pan, X. Xu, X. Ding, S. Huang, Y. Wang and R. Xiong, "GEM: Online Globally Consistent Dense Elevation Mapping for Unstructured Terrain," in *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1-13, 2021.

[5] Z. Jian et al., "PUTN: A Plane-fitting based Uneven Terrain Navigation Framework," *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Kyoto, Japan, 2022, pp. 7160-7166.

[6] F. Ruetz et al., "OVPC Mesh: 3D Free-space Representation for Local Ground Vehicle Navigation," *2019 International Conference on Robotics and Automation (ICRA)*, Montreal, QC, Canada, 2019, pp. 8648-8654.

[7] A. Chilian and H. Hirschmüller, "Stereo camera based navigation of mobile robots on rough terrain," *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, St. Louis, MO, USA, 2009, pp. 4571-4576.

[8] P. Fankhauser and M. Hutter, "A universal grid map library: Implementation and use case for rough terrain navigation," in *Robot Operating System (ROS)*. Springer, 2016, pp. 99-120.

[9] P. Fankhauser, M. Bloesch and M. Hutter, "Probabilistic Terrain Mapping for Mobile Robots With Uncertain Localization," in *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3019-3026, 2018.

[10] C. Sevastopoulos and S. Konstantopoulos, "A Survey of Traversability Estimation for Mobile Robots," in *IEEE Access*, vol. 10, pp. 96331-96347, 2022.

[11] Y. Pan, X. Xu, Y. Wang, X. Ding and R. Xiong, "GPU accelerated real-time traversability mapping," *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Dali, China, 2019, pp. 734-740.

[12] T. Miki, L. Wellhausen, R. Grandia, F. Jenelten, T. Homberger and M. Hutter, "Elevation Mapping for Locomotion and Navigation using GPU," *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Kyoto, Japan, 2022, pp. 2273-2280.

[13] N. Gallagher and G. Wise, "A theoretical analysis of the properties of median filters," in *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 29, no. 6, pp. 1136-1141, 1981.

[14] G. Bradski. "The OpenCV Library," *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, vol. 25, pp. 120-123, 2000.

[15] M. Bertalmio, A. L. Bertozzi and G. Sapiro, "Navier-stokes, fluid dynamics, and image and video inpainting," *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, Kauai, HI, USA, 2001, pp. I-I.

[16] A. Telea. "An image inpainting technique based on the fast marching method," *Journal of graphics tools*, vol. 9, no. 1, pp. 23-34, 2004.

[17] F. Jenelten, R. Grandia, F. Farshidian and M. Hutter, "TAMOLS: Terrain-Aware Motion Optimization for Legged Systems," in *IEEE Transactions on Robotics*, vol. 38, no. 6, pp. 3395-3413, 2022.

[18] T. Shan, J. Wang, B. Englot and K. Doherty, "Bayesian Generalized Kernel Inference for Terrain Traversability Mapping," *Proceedings of the 2nd Annual Conference on Robot Learning*, pp. 829-838, 2018.

[19] A. Melkumyan, F. T. Ramos, "A sparse covariance function for exact Gaussian process inference in large datasets," *International Joint Conference on Artificial Intelligence*, pp. 1936-1942, 2009.

[20] M. Stölzle, T. Miki, L. Gerdes, M. Azkarate and M. Hutter, "Reconstructing Occluded Elevation Information in Terrain Maps With Self-Supervised Learning," in *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 1697-1704, April 2022.

[21] B. Yang, Q. Zhang, R. Geng, L. Wang and M. Liu, "Real-Time Neural Dense Elevation Mapping for Urban Terrain With Uncertainty Estimations," in *IEEE Robotics and Automation Letters*, vol. 8, no. 2, pp. 696-703, 2023.

[22] M. Wermelinger, P. Fankhauser, R. Diethelm, P. Krüsi, R. Siegwart and M. Hutter, "Navigation planning for legged robots in challenging terrain," *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Daejeon, Korea (South), 2016, pp. 1184-1189.

[23] D. D. Fan, K. Otsu, Y. Kubo, A. Dixit, J. Burdick, and A.-A. AghaMohammadi, "Step: Stochastic traversability evaluation and planning for safe off-road navigation," *arXiv preprint arXiv:2103.02828*, 2021.

[24] R. O. Chavez-Garcia, J. Guzzi, L. M. Gambardella and A. Giusti, "Learning Ground Traversability From Simulations," in *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1695-1702, 2018.

[25] L. Wellhausen and M. Hutter, "Rough Terrain Navigation for Legged Robots using Reachability Planning and Template Learning," *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Prague, Czech Republic, 2021, pp. 6914-6921.

[26] L. Wellhausen, A. Dosovitskiy, R. Ranftl, K. Walas, C. Cadena and M. Hutter, "Where Should I Walk? Predicting Terrain Properties From Images Via Self-Supervised Learning," in *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1509-1516, 2019.

[27] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti and D. Rus, "LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping," *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Las Vegas, USA, 2020, pp. 5135-5142.

[28] W. Xu, Y. Cai, D. He, J. Lin and F. Zhang, "FAST-LIO2: Fast Direct LiDAR-Inertial Odometry," in *IEEE Transactions on Robotics*, vol. 38, no. 4, pp. 2053-2073, Aug. 2022.

[29] W. Wu, X. Zhong, D. Wu, B. Chen, X. Zhong and Q. Liu, "LIO-Fusion: Reinforced LiDAR Inertial Odometry by Effective Fusion With GNSS/Relocalization and Wheel Odometry," in *IEEE Robotics and Automation Letters*, vol. 8, no. 3, pp. 1571-1578, 2023.

[30] A. Kleiner, C. Dornhege, "Real-time localization and elevation mapping within urban search and rescue scenarios," *Journal of Field Robotics*, vol. 24, no.8-9, pp. 723-745, 2007.

[31] P. Pfaff, R. Triebel, W. Burgard, "An efficient extension to elevation maps for outdoor terrain mapping and loop closing," *The International Journal of Robotics Research*, vol. 26, no. 2, pp. 217-230, 2007.

[32] J. E. Bresenham, "Algorithm for computer control of a digital plotter," in *IBM Systems Journal*, vol. 4, no. 1, pp. 25-30, 1965.

[33] W. R. Vega-Brown, M. Doniec, N. G. Roy, "Nonparametric Bayesian inference on multivariate exponential families," *Advances in Neural Information Processing Systems*, 2014.

[34] K. Klasing, D. Althoff, D. Wollherr and M. Buss, "Comparison of surface normal estimation methods for range sensing applications," *2009 IEEE International Conference on Robotics and Automation*, Kobe, Japan, 2009, pp. 3206-3211.