Proceeding of the IEEE
International Conference on Robotics and Biomimetics
Dali, China, December 2019

GPU accelerated real-time traversability mapping

Yiyuan Pan, Xuecheng Xu, Yue Wang, Xiaqing Ding, and Rong Xiong
State Key Laboratory of Industrial Control and Technology
Zhejiang University
Hangzhou, 310027, China
wangyue@iipc.zju.edu.cn

***Abstract*— The navigation of autonomous mobile robots requires effective localization and mapping modules. Dense map representation of the robot surroundings, which contains detailed information of the drivable region can be easily used for motion planning. To build a dense map on mobile robots, the main challenge is that the system has to be efficient due to the limited computational resources. In this paper, we propose a novel approach to generate a dense map with drivable information. First, the dense map with elevation information is generated by the proprioceptive localization results acquired from kinematic and inertial measurement, as well as the accumulated raw data from the range sensor. Then, we calculate slope and roughness of each grid on the map to assess whether this area is accessible. Combining the data in these two steps, we can form the dense map with drivable information. The entire system accelerated by GPU performs well in handling dynamic obstacles. For implementations, we demonstrate the effectiveness of our approach with mobile robot in a complex outdoor environment and have a detailed comparison with other methods.**

***Index Terms*— dense mapping, ray tracing, traversability**

I. Introduction

Many challenges need to be solved before wheeled robots, or footed robots can navigate safely and efficiently through an environment. One crucial part is to search for the drivable area to plan a motion over and around the obstacles. In this respect, we need knowledge of the surrounding which is not blocked by obstacles as the drivable area. Equipped with onboard range sensors (such as laser range, time-of-flight, and stereo camera sensors), mobile robots can collect raw distance measurements at each instant and calculate optimal height values of each grid based on multiple repeated observations. Using simultaneous localization and mapping (SLAM) algorithms [1], we can obtain a dense elevation map, which can represent the highest height of all terrains. However, the terrain is very likely to be uneven, and we can not calculate the height information of each grid to estimate the traversability. So we combine the adjacent grid elevation information to calculate its geometry features such as slope and roughness to evaluate its traversability. The higher the score of traversability, the greater the chance of safe navigation. When we have traversable information of the dense map, we can process the obstacle area by ray tracing to confirm whether it is a dynamic obstacle. Robotics systems usually require real-time processing capability which means that the input data have to be processed at a speed that is faster than or equal to the speed of the input data. However, the real-time processing requirement of robotics applications is hard to satisfy because of limited computational resources (such as CPUs, GPUs, and memory) and power on mobile robots. In this study, we obtain the localization results by an Extended Kalman Filter [2] which is updated by the results of laser-based ICP algorithm and then construct a real-time dense map using range sensors.

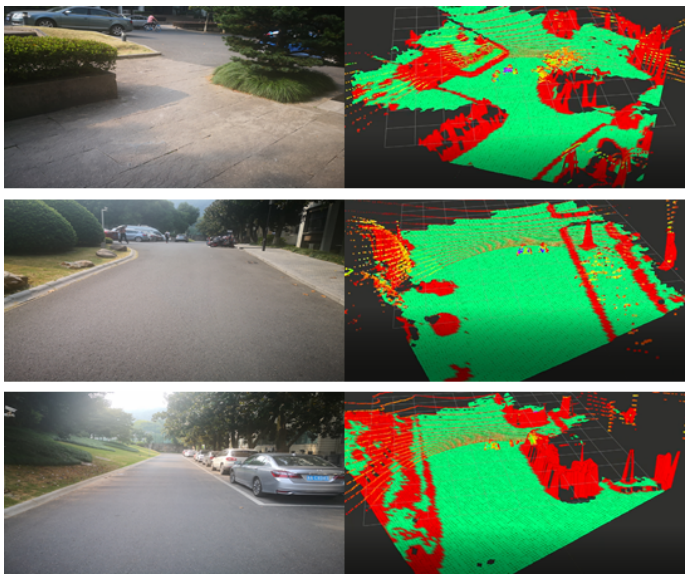


Fig. 1. Drivable area results of dense map in different scenarios. The green area represents the drivable area which has high traversability score, and the red represents the impassable area which has low traversability score. The method can identify the car, the step, and other obstacles as impassable area, and identify the flat and the slope with low gradient as drivable area.

We test our system on datasets collected by an unmanned ground vehicle equipped with two lidars and one Inertial Measurement Unit (IMU). It shows that we can build a dense map faster than *elevation map* [16], which can commonly be used for motion planning. We also analyze the results of constructing dense mapping as shown in Fig.1.

The contributions of this work are as follows:

- We construct the dense map using GPU and multi-threaded parallelism, which makes our entire system in

978-1-7281-6321-5/19//$31.00 ©2019 IEEE

real-time.

- We introduce a dense map representation with drivable information which is easy for navigation.
- We check the restricted height of obstacle region only by ray tracing instead of all areas to reduce calculation costs. With this scheme, dynamic objects can be handled well.

II. Related Work

Generally, dense navigation maps are built in two stages. The first step is to generate the poses of sensors together with the positions of sparse features textures. Employed to minimize the difference between two clouds of points, the algorithm of Iterative closest point (ICP) [3] can be used to localize robots. Using the approach of Structure from Motion (SfM) [4], global positions and orientations of the camera can be obtained by matching 3D-3D matches between the compressed model and the captured images. This method can also generate accurate localization results quickly.

The second step is to create dense map by projecting 3D points onto a global coordinate frame. There are many popular dense map representations, including occupancy grid map, OctoMap, and elevation map. Occupancy grid map [5] divides the area into square grids of the same size. Each grid maintains free, occupied, and unknown space information. Using a series of computer algorithms in probabilistic robotics, occupancy map addresses the problem of generating maps from uncertain and noisy sensor measurement data when the robot pose is known. The OctoMap [6] is based on octrees and uses probabilistic occupancy estimation. The critical contribution of this map representation is that it reduces the memory requirement by locally combining coherent map volumes, both in the mapped free areas and the occupied space. This compression method significantly reduces storage memory and improves efficiency, so many famous dense mapping frameworks employ this data structure. Combining with Truncated Signed Distance Function(TSDF) [7], online real-time 3D dense mapping [8] [9] becomes possible. Several robotics applications require a 3D model of the environment, especially airborne and underwater missions. However, it is redundant for the unmanned ground vehicle to navigate. Therefore, many ground mobile robots adopt 2.5-dimensional elevation maps for motion planning, such as wheeled robots and footed robots. The elevation mapping [10] [11] method provides a data structure for representing the environment of robots operating in a complex environment or on rough terrains. An elevation map stores the height of the surface in the environment in each cell of a discrete grid. This map structure has already satisfied the planning demands of ground mobile robots, and more detailed height information has made it possible for robots to plan with various forms of motion, especially for quadruped robots.

After a dense map is generated, we search for the drivable area with the information above. The early approaches use some traditional image-based algorithms [12] to segment the drivable ground through texture and color information. In recent years, deep learning methods have made significant progress in the field of computer vision. As a data-driven approach, the deep neural network does not extract hand-designed features which is the low-order visual information of the image pixel. The network can also automatically update the model parameters using inverse propagation during training. However, the learning method [13] [14] consumes vast computational resources, which is not acceptable in the mobile robot case. Image-based and learning methods require the operation to project the drivable part on the image to a dense map of 3-dimensional or 2.5-dimensional through the camera model. Nevertheless, this method of external projection puts a heavy demand on the external reference calibration, which ignores the readily geometric information of the dense map. Practically, employing a priori information of flat ground is more appropriate. The flat and non-flat road geometry [15] can be obtained from the dense map as the valid and robust features for road obstacle detection. Also, the segmentation of the drivable area combined with the driving state models achieves good results.

The work most relevant to us was published by [16]. The proposed method incorporates the drift, uncertainties of the state estimation, and a noise model of the distance sensor. It provides open-source packages and publishes a detailed explanation. However, this project is too slow to meet the demands of real-time mapping. In this condition, we accelerate it with GPU and add a structure-based drivable area detection, so that the entire system can establish a dense map with drivable region in real-time.

III. Method

The structure of the whole system is shown in Fig.2. We retrieve the localization results from the ICP algorithm working on lidar (Lidar1 in Fig.5) placed horizontally and updated under the Extended Kalman Filter paradigm. By applying the interpolation algorithm of IMU data, the output frequency of the localization can be higher. In this paper, we focus on the representation of the dense map, including processing the raw data from range sensor (Lidar2 in Fig.5), updating variance of each grid of map and dealing with dynamic objects. We installed the lidar tilt down to make more valid points project to the neighboring map area, making the elevation map more accurate.

A. Online dense mapping

1) Transform raw lidar points: With the localization results, the relation between the Lidar1 frame L_1 and the inertial frame I is determined. The corresponding transform matrix is $T^I_{L_1}$. With the calibration, we get the external reference between the two lasers $T^{L_1}_{L_2}$, which represents the

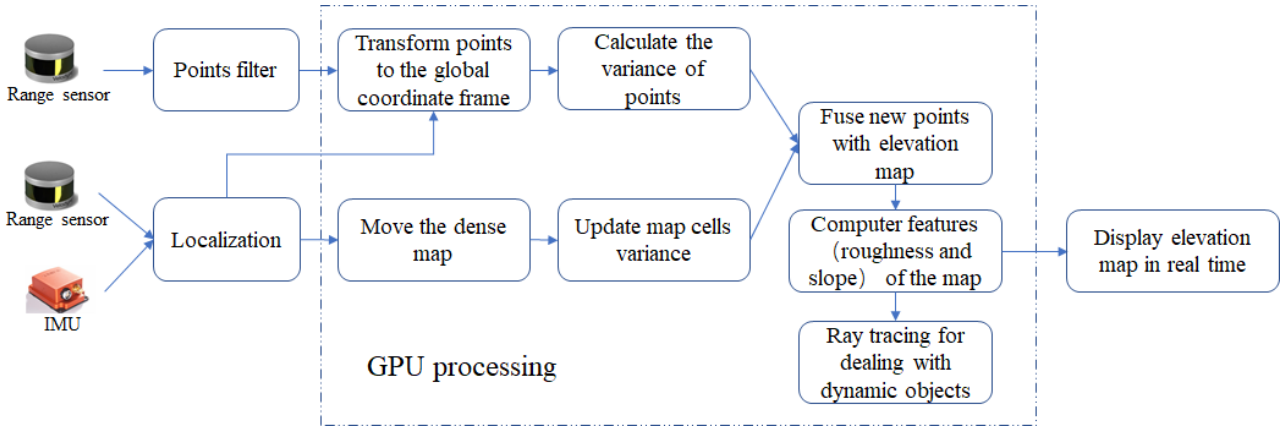


Fig. 2. Overview of our system. There are two threads in our system, one thread generates poses of the sensor in the global coordinate system, and the other one is used to build the dense map. We perform area filtering on the data from the downward tilted lidar, reducing the amount of subsequent calculations, and passing the processed points to GPU for processing. Combining mix calculation of CPU and GPU, and using multi-threaded computing, we can finally display the elevation map with drivable area information in real-time.

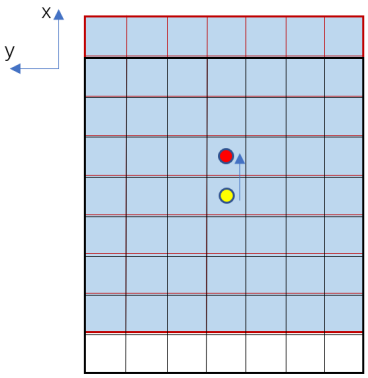


Fig. 3. When the robot shifts 1 unit from the starting yellow position along the X-axis to the red position, the last row of stored data is deleted and waits for the latest measurement data update, and the other data remains unchanged. The value $X_{StartIndex}$ changes from 0 to 1. With this value, the actual position of the map can correspondingly project to the storage location, e.g., data of the cell (1, 1) can be obtained by querying the storage location (0, 1).

relation from Lidar1 to Lidar2. The transform matrix from the Lidar2 frame to the Lidar1 frame can be inferred by the formula below:

$$T_{L_2}^{I} = T_{L1}^{I} \; o \; T_{L_2}^{L_1} \tag{1}$$

The operator o is used to indicate a concatenation of transformations. The points $P_i^{L_2} = (x_i, y_i, z_i)$ collected by Lidar2 can be transformed to the world frame I:

$$P_i^{I} = T_{L_2}^{I} \; o \; P_i^{L_2} \tag{2}$$

2) Move the map: In the robot-centric elevation grid map, each movement of the robot is accompanied by the movement of the grid map. This process causes the data at the edge of the partial grid map to be cleared and the position of the retained data in the map to change. In fact, mutual positional relationships between the retained data never changed. Based on this finding, after each movement, we don't update the existing map data, but update the position shift value $X_{StartIndex}$ and $Y_{StartIndex}$ according to the movement of the robot. The updated map can use this value to query data from all locations on the map. The specific case is shown in Fig.3.

3) Add new measurements to map: We can consider the height of measuring points as a Gaussian probability distribution model $\tilde{p} \sim N(p, \sigma_p^2)$ with mean p and variance σ_h^2. Observed by the range sensor, the height of the point P_i in the map frame can be determined according to the previous subsection. Simultaneously, according to the range sensor noise models [17], σ_h^2 can be derived.

As we can see, the elevation grid map frame is defined with the translation of Lidar1/robot, the dense map data needs to be updated whenever the mobile robot moves relative to the inertia frame I. However, the localization results are not completely accurate, we need to update the variance of valid data after each movement of the map. After obtaining these updated value, the optimal estimated height $\hat{h}^+$ and variance σ_h^{2+} of the corresponding cell in the elevation map can be updated according to Kalman filter:

$$\hat{h}^+ = \frac{\sigma_p^2 \hat{h}^- + \sigma_h^{2-} p}{\sigma_h^{2-} + \sigma_p^2} \qquad \sigma_h^{2+} = \frac{\sigma_h^{2-} \sigma_p^2}{\sigma_h^{2-} + \sigma_p^2} \tag{3}$$

$(\hat{h}^-, \sigma_h^{2-})$ represnts the height measurement and the variable of the cell before updating.

For multiple different height measurements fall in the same cell, we use the Mahalanobis distance to evaluate the difference between the new measured height and the existing height. With this value, we can adopt suitable update strategies [18] to fuse the data to make the final result closer to the height of real terrain as shown in Alg.1.

Algorithm 1 Update strategy for multiple measurements

Observed point collected by the Lidar2, P_i
Z-axis direction value of the point P_i, h_{P_i}
Variance of the point P_i, $\sigma^2_{P_i}$
The cell corresponding to the location of the point P_i , E_i
Elevation of the cell E_i, h_{E_i}
Variance of the cell E_i, $\sigma^2_{E_i}$
for all P_i **do**
  if E_i is NULL **then**
    $h_{E_i} \Leftarrow h_{P_i}$
    $\sigma^2_{E_i} \Leftarrow \sigma^2_{P_i}$
  else
    $M_{distance} \Leftarrow \frac{|h_{E_i} - h_{P_i}|}{\sigma_{E_i}}$
    if $M_{distance} > threshold$ and $h_{E_i} < h_{P_i}$ **then**
      $h_{E_i} \Leftarrow h_{P_i}$
      $\sigma^2_{E_i} \Leftarrow \sigma^2_{P_i}$
    else $\{M_{distance} < threshold\}$
      $h_{E_i} \Leftarrow \frac{h_{E_i}\sigma^2_{P_i} + h_{P_i}\sigma^2_{E_i}}{\sigma^2_{P_i} + \sigma^2_{E_i}}$
      $\sigma^2_{E_i} \Leftarrow \frac{\sigma^2_{P_i}\sigma^2_{E_i}}{\sigma^2_{P_i} + \sigma^2_{E_i}}$
    end if
  end if
end for

B. Calculate drivable region

The drivable area is to be calculated after obtaining the 2.5-dimension elevation map. Given that robots have very limited computation resources, the easily calculated variables like surface normal and neighbor height deviation are utilized to evaluate the traversability of each cell [19]. The grid structure of the elevation map brings the benefits to the system that it can speedily access the coordinate and the height of the nearest neighbors for each cell.

1) Surface normal vector: Using Singular Value Decomposition(SVD) [20], the surface normal vector can be extracted from the noisy surface representation of the elevation map. The calculated surface normal vector is assigned to a specific grid on the elevation map. A classical calculate method is to shift the origin to p_i which contains three-dimensions information of $P_{(x,y)}$ and fit a plane $S_i = n_{ix}x + n_{iy}y + n_{iz}z$ with surface points Q_i in the neighboring area. i.e., solve:

$$\min_x \left\| [Q_i - 1_k {p_i}^T] n_i \right\|_2 \tag{4}$$

Where k represents the num of points in the neighboring region, Q_i is a $3 \times N$ matrix with three-dimensions coordinates of K neighbor points and $n_i = [n_{ix}, n_{iy}, n_{iz}]$ is surface normal vector. This optimization problem can be solved by computing the SVD ($U\Sigma V^T$) of $[Q_i - 1_k {p_i}^T]$. The minimizer $\hat{n}_i$ is the vector in V that corresponds to the smallest singular value in Σ, see [21]; thus, the normal vector is derived.

2) Neighbor height deviation: It is common to use the neighbor height deviation in the evaluation of the roughness. The deviation can be derived using the formula below:

$$H_d = |h_{P_{(x,y)}} - \overline{h}| \tag{5}$$

Where $h_{P_{(x,y)}}$ is the height of the cell $P_{(x,y)}$ and $\overline{h}$ is the mean height of all the grids in an $N \times N$ window centered on the cell $P_{(x,y)}$.

3) Grid Scores: A grid score $Score(x, y) \in [0, 1]$ related to coordinate is introduced to each cell of the dense map. The score value denotes the probability of traversability. Referring to [22] [23], we utilize a simple and effective formula introduced in

$$Score_{(x,y)} = max(1 - w_s \frac{v_{slope}(x,y)}{v_{s_{crit}}} - w_r \frac{v_{rough}(x,y)}{v_{r_{crit}}}, 0) \tag{6}$$

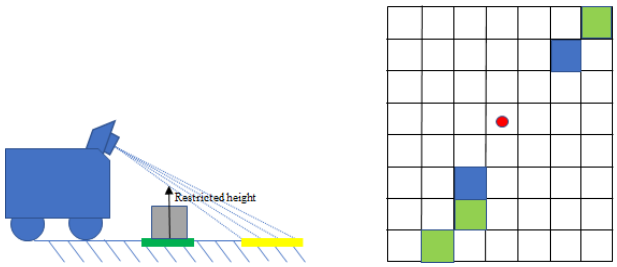
The variable w_s and w_d represent the weight factor of slope and roughness, and v_{crit} denotes the max allowed value of the corresponding feature. A cell with $score = 1$ indicates that it is an accessible area, and a cell with $score = 0$ might be an obstacle with high possibility. The score combines the evaluation of slope (deviated from surface normal vector) and roughness (neighbor height deviation) is adaptive to the real-time application.

C. Process dynamic objects

Due to the use of the cumulative method and the strategy of the map update, when there are dynamic obstacles, such as pedestrians and moving vehicles, leaving the original location, the original height of the cell still retains. It causes a terrible error because the drivable area is detected as an impassable area. Ray tracing can be used to deal with the problem of the error height value caused by dynamic objects. The fact that an object can be observed implies an important principle: there are no other obstacles between the sensor and the object. So regarding the ray tracing model in Fig.4(a), we use real-time lidar data to check if the existing cell in the elevation map blocks the light path, and if so, clear it. For a dense map with drivable regions, we only need to check the cell with a low score of the traversability. As shown in Fig.4(b), the red point represents the robot, and blue cells represent obstacles, at which we only need to check the height constraints by the observation points that fall in the green cell, without having to calculate the height constraints of all cells.

IV. Experiment

We first show the installation method of the hardware device in Fig.5. The data of Lidar1 and IMU is used to get localization results at a high frequency. In order to get more raw measuring data to fall in the effective area, we install Lidar2 which is used to establish the dense elevation map with 15 degree incline to the horizontal plane. At the same time, we employ the synchronization method of two



(a) (b)

Fig. 4. Ray tracing. (a) Ray tracing model. The yellow area is our observation region. With the geometric relationship, we can obtain the restricted height of the area between the robot position and the observation point. (b) We only need to calculate the restricted height of the obstacle area, which greatly reduces the amount of calculation.

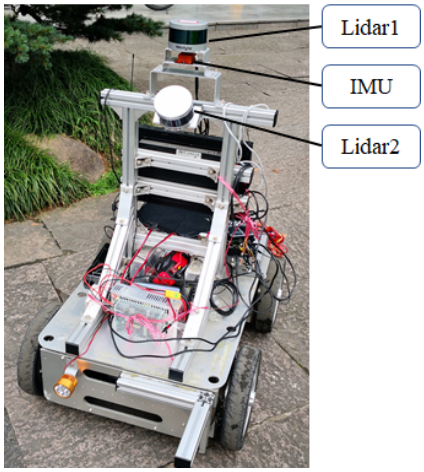
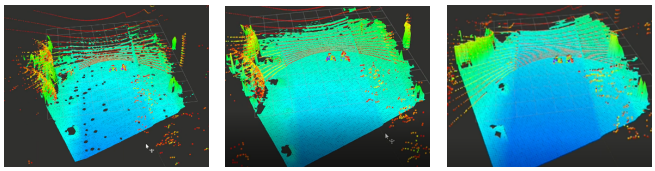


Fig. 5. Unmanned ground vehicle used to collecting data. The data from Lidar1 and IMU is calculated to generate localization results. The data from Lidar2 is used to construct the dense map.

lidars. With pulse signal of IMU as the trigger signal, the two lidars can collect data at the same frequency (10Hz) at almost the same time, thus reducing the error due to out of sync. The whole mapping system is implemented with C++ and operated on the Robot Operating System (ROS) in industrial personal computer (IPC) with Intel i7-8700 (CPU) and Nvidia 1060 (GPU). To satisfy the requirements of low weight and low power in some mobile robot applications, we also test our approach in TX2 (a light-weight, low-power embedded system with GPU).

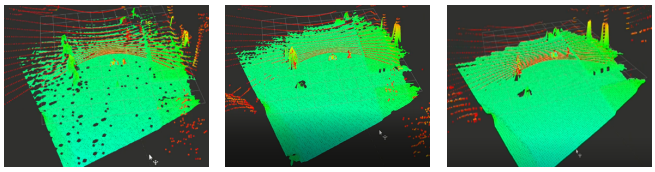
A. Comparision with elevation map

Considering the running time and the demands of the local map for planning, we construct the dense map with a size of $12m \times 12m$ and set the resolution to 0.1m. In other words, $120 \times 120 = 14400$ grids of the map are generated and updated with each observation and movement of the robot. With the data collected on the campus, we use the same localization results and the same parameter of the dense map



(a) (b) (c)

Fig. 6. Experiment results in a static environment. (a) *elevation map* in IPC. (b) Our approach in TX2. (c) Our approach in IPC. The red lines consists of a lot of points are raw measurement data obtained by sensor.



(a) (b) (c)

Fig. 7. Experiment results in a dynamic environment. (a) *elevation map* in IPC. (b) Our approach in TX2. (c) Our approach in IPC.

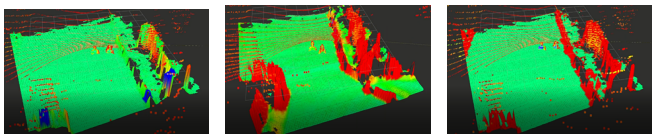
to estimate the methods of an open-source system which can be called *elevation map* and ours in IPC and TX2, in respect of the map construction speed, accuracy, and processing quality of dynamic obstacles.

We recorded the average running time of the three systems in each step in the Tab.1. Whether it is running in IPC or TX2, each step of our method costs less computation time than *elevation map*, especially processing dynamic objects. According to the computation time, our system can reach a high frequency, but *elevation map* can only reach 2Hz, which also leads to the problem that shown in the Fig.6 in a static environment. The dense map constructed by *elevation map* has more areas that are not updated, resulting in more apparent voids. Slow update rate can also cause partial areas to be updated without using multiple measurements, resulting in inaccurate terrain heights in the elevation map, especially when the mobile robot suddenly vibrates and produces inaccurate localization results.

Ray tracing is used to process dynamic objects. This method is simple and effective, but it spends most of the time. As the Tab.1 shows, our approach reduces the computation time on dynamic obstacles, and the processing results are

TABLE I
TIME COMPARISION

Step	*Elevation map* (ms)	Our approach (ms)	Our approach (ms)
Hardware	IPC	IPC	TX2
Update map variance	8	2	0.5
Process points	248	64	19
Calculate the features	-	43	15
Process dynamic objects	893	4	1
Pointcloud callback	548	167	46



(a) (b) (c)

Fig. 8. Different features of the dense map: The different colors of each grid represent the values of the features. (a) Roughness. The horizontal ground is green, the concave place is blue, and the raised place is red. (b) Slope. The color of the grid closer to red represents the larger gradient, and the color closer to green represents the terrain is flatter. (c) Traversability. Calculated by roughness and slope, the drivable area with high traversability score is shown in green, and the area with low traversability score is shown in red.

shown in the Fig.7 in a dynamic environment. *Elevation map* removes some of the erroneous observations, but there are still evident object trails compared to our method. This is because the ray tracing implementation of *elevation map* is too slow (at a frequency of 1Hz) so that it has to create a new thread which needs to be triggered by a timer at 1Hz to process. However, we only need to check the restricted height on the obstacle grid, which improves the system performance. Furthermore, we accelerate this step through GPU so that we can employ serial computing in the main program to achieve the effect of clearing obstacles in real-time.

B. Analyze drivable area results

Compared with *elevation map*, we add a practical judgment method based on geometry to divide dense elevation map to the drivable area. In the step of detection, we set the window size of $0.5m \times 0.5m$ to calculate slope and roughness of the grid which needs to be detected. This design not only highlights the locality of the feature and prevents the use of too many adjacent grid data and also decreases the computational burden.

Through the actual test and experiment, we set slope weight and roughness weight to 0.4 and 0.6. We show the experimental results at the same place, and it can be seen from the Fig.8 that the feature of slope or roughness can segment the drivable region roughly. We binarize the calculated score of traversability, which is used to determine the drivable region. The grid with the score above the threshold is defined as the drivable area and the grid below the threshold indicates obstacles or areas that not suitable for passage. We can easily convert the dense elevation map to the costmap for navigation. In this experiment, we set the threshold to 0.7. Assuredly, the method that combines the two features achieves more competitive results.

V. Conclusion

In this article, we propose an elevation dense map method with drivable information. The dense elevation map is constructed by range sensor data, and the corresponding sensor poses. Multiple observations are used to update the terrain heights based on the probability distribution model, which yields good results. With the obtained elevation dense map, we use the geometry of the adjacent grid to estimate the traversability and check the height limit of these areas by ray tracing to clear dynamic obstacles with a small amount of calculation. The output of the system can generate 2-dimensional cost maps with occupied grid information, which can be easily used in classic planning algorithms, such as A* [24] and rapidly exploring random tree (RRT) [25]. Also, the 2.5-dimensional elevation map can be used in more complex motion planning situation of quadruped robots. Because of the adoption of GPU acceleration and multi-threaded parallelism, our experimental results are significantly better than *elevation map* and produce competitive results in real-time.

In the future, we will focus on constructing a large-scale global dense map for many robotics applications, such as exploring and rescuing. The global map built by fusing local maps with odometry systems is capable of achieving closed-loop optimization and can be maintained so that the data of the map does not always grow with the increase of the robot exploration time in a bounded environment. Such a system is significant for long-term robotic tasks.

VI. Acknowledgment

This work was supported in part by the National Key R&D Program of China (2018YFB1309300), in part by Science and Technology on Space Intelligent Control Laboratory (No.HTKJ2019KL502002) and in part by the Fundamental Research Funds for the Central Universities.

References

[1] Dissanayake M W M G, Newman P, Clark S, et al. A solution to the simultaneous localization and map building (SLAM) problem[J]. IEEE Transactions on robotics and automation, 2001, 17(3): 229-241.

[2] Ljung L. Asymptotic behavior of the extended Kalman filter as a parameter estimator for linear systems[J]. IEEE Transactions on Automatic Control, 1979, 24(1): 36-50.

[3] Chetverikov D, Svirko D, Stepanov D, et al. The trimmed iterative closest point algorithm[C]//Object recognition supported by user interaction for service robots. IEEE, 2002, 3: 545-548.

[4] Leutenegger S, Lynen S, Bosse M, et al. Keyframe-based visualinertial odometry using nonlinear optimization[J]. The International Journal of Robotics Research, 2015, 34(3): 314-334.

[5] Elfes A. Using occupancy grids for mobile robot perception and navigation[J]. Computer, 1989, 22(6): 46-57.

[6] Hornung A, Wurm K M, Bennewitz M, et al. OctoMap: An efficient probabilistic 3D mapping framework based on octrees[J]. Autonomous robots, 2013, 34(3): 189-206.

[7] Werner D, Al-Hamadi A, Werner P. Truncated signed distance function: experiments on voxel size[C]//International Conference Image Analysis and Recognition. Springer, Cham, 2014: 357-364.

[8] Oleynikova H, Taylor Z, Fehr M, et al. Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning[C]//2017 Ieee/rsj International Conference on Intelligent Robots and Systems (iros). IEEE, 2017: 1366-1373.

[9] Ling Y, Shen S. Realtime dense mapping for online processing and navigation[J]. Journal of Field Robotics, 2019, 36(5): 1004-1036.

[10] Kweon I S, Kanade T. Extracting topographic terrain features from elevation maps[J]. CVGIP: image understanding, 1994, 59(2): 171-182.

[11] Pfaff P, Triebel R, Burgard W. An efficient extension to elevation maps for outdoor terrain mapping and loop closing[J]. The International Journal of Robotics Research, 2007, 26(2): 217-230.

[12] Andresen F, Davis L, Eastman R, et al. Visual algorithms for autonomous navigation[C]//Proceedings. 1985 IEEE International Conference on Robotics and Automation. IEEE, 1985, 2: 856-861.
[13] Ronneberger O, Fischer P, Brox T. U-net: Convolutional networks for biomedical image segmentation[C] //International Conference on Medical image computing and computer-assisted intervention. Springer, Cham, 2015: 234-241.
[14] Chen L C, Papandreou G, Kokkinos I, et al. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs[J]. IEEE transactions on pattern analysis and machine intelligence, 2018, 40(4): 834-848.
[15] Labayrade R, Aubert D, Tarel J P. Real time obstacle detection in stereovision on non flat road geometry through" v-disparity" representation[C]//Intelligent Vehicle Symposium, 2002. IEEE. IEEE, 2002, 2: 646-651.
[16] Fankhauser P, Bloesch M, Hutter M. Probabilistic terrain mapping for mobile robots with uncertain localization[J]. IEEE Robotics and Automation Letters, 2018, 3(4): 3019-3026.
[17] P. Fankhauser, M. Bloesch, D. Rodriguez, R. Kaestner, M. Hutter, and R. Siegwart, Kinect v2 for mobile robot navigation: Evaluation and modeling, in Proc. Int. Conf. Adv. Robot., 2015, pp. 388394.
[18] A. Kleiner and C. Dornhege, Real-time localization and elevation mapping within urban search and rescue scenarios, J. Field Robot., vol. 24, pp. 723745, Aug. 2007.
[19] Fankhauser P, Bjelonic M, Bellicoso C D, et al. Robust rough-terrain locomotion with a quadrupedal robot[C]//2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018: 1-8.
[20] Klasing K, Althoff D, Wollherr D, et al. Comparison of surface normal estimation methods for range sensing applications[C]//2009 IEEE International Conference on Robotics and Automation. IEEE, 2009: 3206-3211.
[21] Golub G H, Van Loan C F. Matrix computations[M]. JHU press, 2012.
[22] Winkler A W, Mastalli C, Havoutis I, et al. Planning and execution of dynamic whole-body locomotion for a hydraulic quadruped on challenging terrain[C]//2015 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2015: 5148-5154.
[23] Kolter J Z, Rodgers M P, Ng A Y. A control architecture for quadruped locomotion over rough terrain[C]//2008 IEEE International Conference on Robotics and Automation. IEEE, 2008: 811-818.
[24] Hart, P. E.; Nilsson, N. J.; Raphael, B. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths".IEEE Transactions on Systems Science and Cybernetics SSC4.4(2): 100107.doi:10.1109/TSSC.1968.300136.
[25] Kuffner J J, LaValle S M. RRT-connect: An efficient approach to single-query path planning[C]//Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065). IEEE, 2000, 2: 995-1001.