Elevation Mapping for Locomotion and Navigation using GPU

Takahiro Miki, Lorenz Wellhausen, Ruben Grandia, Fabian Jenelten, Timon Homberger, Marco Hutter

***Abstract*— Perceiving the surrounding environment is crucial for autonomous mobile robots. An elevation map provides a memory-efficient and simple yet powerful geometric representation for ground robots. The robots can use this information for navigation in an unknown environment or perceptive locomotion control over rough terrain. Depending on the application, various post processing steps may be incorporated, such as smoothing, inpainting or plane segmentation. In this work, we present an elevation mapping pipeline leveraging GPU for fast and efficient processing with additional features both for navigation and locomotion. We demonstrated our mapping framework through extensive hardware experiments. Our mapping software was successfully deployed for underground exploration during DARPA Subterranean Challenge and for various experiments of quadrupedal locomotion.**

I. INTRODUCTION

For mobile robots that operate autonomously in an unknown environment, it is crucial to use their onboard sensors to perceive the surroundings. For planning their movements, robots often create a map to fuse the sensor measurements over time since the sensor coverage is limited. An occupancy grid can be used for indoor navigation on flat ground with obstacles [10], [41]. This two-dimensional map shows whether each cell is occupied or not, thus robots can use it to plan their path to the target point while avoiding obstacles for example. However, this map is not suitable for environments with rough terrain with inclinations or steps. To solve this problem, a 2.5D elevation map is often used which can hold the height of the ground for each cell [18], [26], [13], [14]. In this map, a 2D grid stores the height information of each cell to represent the terrain geometry. To further extend the common 2.5D representation to multi-level structures such as a bridge, Triebel et al. proposed to construct a multi-level surface map [43]. Some methods use voxels to represent spatial information in 3D. Occupancy probabilities of voxels are updated based on sensor measurements [20], [33]. Along with its ability to model more general structures than a 2.5D grid, the 3D grid has a higher memory requirement compared to a 2.5D elevation map with the same grid size.

In rough terrain navigation, many planners use a cost map calculated according to local geometric features such as the slope or surface roughness [46], [19], [11]. Additionally, some navigation tools are based on a foothold score, as the sum of geometric features [45]. Besides mobile robot navigation, legged robots also benefit from perceiving their surroundings to control their locomotion. An elevation map provides a representation of the geometric landscape that can be directly embedded into a control pipeline. It was used to plan trajectories and select footholds in a number of works [26], [17], [12], [23], [25], [44]. To accomplish both navigation and locomotion tasks, it is crucial to build an elevation map and perform post-processing in real-time.

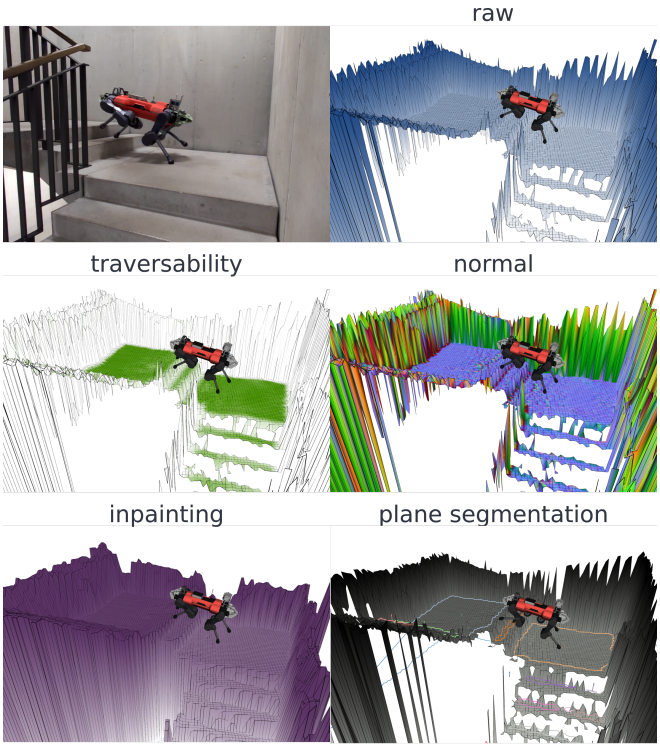

Fig. 1. Proposed elevation mapping framework on GPU. The point cloud processing, traversability estimation and normal calculation is performed on GPU. In addition, inpainting, smoothing filters or plane segmentation features are provided for legged locomotion control.

As an existing mapping framework, Fankhauser et al. present a probabilistic robot-centric elevation map on Central Processing Unit (CPU) with Robot Operating System (ROS) integration [13], [14]. It transforms depth measurements, represented as point clouds, to the local frame of the robot and updates each cell's terrain height in a Kalman-Filter-like fashion. This method was used for locomotion tasks [12], [7] or navigation tasks [46]. However, it is not efficient enough to process large amounts of point cloud data in real time for faster agile robot movements than the previous work [12]. This motivated our development of a terrain map-

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 780883.
This research was supported by the Swiss National Science Foundation (SNSF) as part of project No.188596.
This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101016970.
All authors are with Robotic Systems Lab, ETH Zurich

ping framework with GPU acceleration. A similar approach was used in the recent work of Pan et al. in which a local map update was performed on GPU and a global map was created by registering the local map [35]. However, the main focus of their software is large range navigation and they incorporate down-sampling in the point cloud data. It does not consider the legged locomotion tasks that requires fine grained geometric representation.

As another approach of creating an elevation map, the voxels can be used to project a 3D map onto a 2D grid structure [11], [19], [34]. There are some methods that utilize GPU for faster point cloud processing [3], [34]. However, to create a high-resolution terrain map, the voxel grid size needs to be small which requires a large amount of memory. In addition, the robot needs to maintain both a voxel grid and an elevation map at the same time.

This work presents an elevation mapping framework integrated with ROS using GPU for efficient point cloud registration and ray casting. In this mapping framework, we developed and integrated multiple features that were used for legged robot navigation and locomotion. In addition, we addressed issues we encountered during the field deployments. For example, state estimation drift can cause artifacts in the map and to address this, we introduce height drift compensation. Also, an overhanging obstacle near the robot will appear as a wall and prevents a planner to plan below it. We introduced a raycasting method for visibility clean up and an exclusion area to remove virtual artifacts when the robot gets close to them. Furthermore, we integrate a learning-based traversability filter [45], which gives a traversability value based on the local geometry and enhanced the elevation map through improvements such as *upper bound* calculation or overlap clearance for navigation tasks. For locomotion, we integrated various smoothing filters used in [23], [22] and plane segmentation used in [16].

We demonstrate our framework through extensive experiments. Our implementation supported legged locomotion and navigation research and formed the basis for perceiving surrounding terrains used by learning-based controllers [30], [38], [29], model-based controllers [23], [22], [16], or others such as navigation [45], [47] or learning occlusion filling [40]. In addition, the proposed elevation mapping solution was successfully used for underground exploration missions during Defense Advanced Research Projects Agency (DARPA) Subterranean Challenge [9] where team CERBERUS [8], [42] deployed our mapping on four quadrupedal robots, ANYmal [21].

Our contribution is as follows,

- Developing a GPU-based elevation mapping implementation, supporting a variety of different filters and other features for legged locomotion and mobile robot navigation.
- Validating the real-world applicability through extensive experiments.

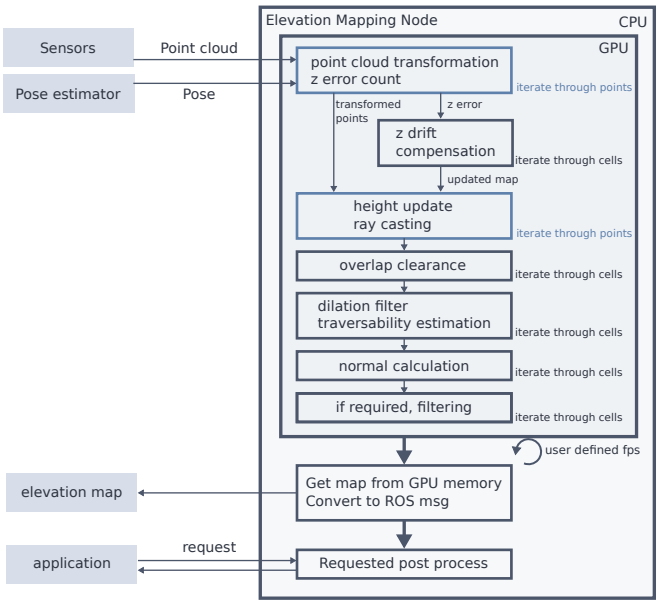

Fig. 2. Overview of processing. The point cloud data from the sensor and the pose information from the pose estimator are inputs to the GPU. After transforming the point cloud and calculating the z drift errors, the current map estimation is adjusted to match the latest sensor measurement. Then the main height map update and ray casting are performed. As the map is updated, various filters are applied. The map data is transferred to CPU memory with a user-defined frequency to publish as a ROS message to reduce unnecessary data transmission.

- Open sourcing elevation mapping software. [1]

II. Methods

In this section, we describe the methods used to construct the elevation map on GPU and the additional features used for locomotion and navigation.

We first describe the overview of our pipeline (Section II-A) and show the core methodology to update the map (Section II-B,II-C). Then, we introduce features we developed to improve the map quality (Section II-D, II-E, II-F). At last, we summarize the post-processing features used for navigation and legged locomotion (Section II-G, II-H, II-I).

A. Overview

Fig. 2 shows the overview of our software. The input is the point cloud collected from depth sensors and the robot's estimated pose. The latter can be provided by a Simultaneous Localization and Mapping (SLAM) based pose estimation or odometry system.

First, this data is transferred to GPU memory. Then, the points are transformed to the user specified map frame using the ROS tf library. At the same time, height drift error is calculated and stored for the downstream process. Based on this height error, the map is shifted so that the map matches the latest sensor measurement. Following the adjustment of the current map to the latest measurements, the height is updated through each point measurement iteration. Within

[1] https://github.com/leggedrobotics/elevation_mapping_cupy

the same iteration, we perform ray-casting to remove the penetrated objects and update the upper-bound layer. Then, the operations which iterate through each cell in the map are performed (overlap clearance, traversability estimation, normal calculation, and additional filtering if required).

The map is published in a user-defined frequency. With this frequency, the map data is transferred to CPU memory and published as a GridMap [15] message to reduce unnecessary data transfer between GPU and CPU. Based on the request, the post-processing on CPU is calculated and returned to the application node.

B. Implementation details

We use ROS [37] for inter-process communication and Cupy[32] to implement parallel computing on GPU. We used Cupy because of its simple API in Python and the ability to define custom Compute Unified Device Architecture (CUDA) Kernels. The Python interface allows using deep learning frameworks such as PyTorch [36] with the same GPU memory. We used roscpp instead of rospy mainly due to the slow serialization of ROS messages on rospy.

C. Height cell update

The height in each cell is updated through a Kalman filter formulation as done in [14]. We iterate through each point in parallel using custom CUDA kernels.

$$h = \frac{\sigma_p^2 h + \sigma_m^2 p_z}{\sigma_m^2 + \sigma_p^2}, \tag{1}$$

where h is the estimated height of the cell, σ_p^2 is the variance of the point calculated from a sensor's noise model, and p_z is the height measurement of the point. σ_m^2 is the estimated variance of the cell, initialized to a large number in the beginning. As the noise model, we used, $\sigma_p^2 = \alpha_d d^2$, where d is the distance of the point from the sensor and α_d is a user-defined parameter, simplified the model of Nguyen et al where the sensor noise variance increases quadratically with the distance [31].

Also, the variance of each cell is updated as below.

$$\sigma_m^2 = \frac{\sigma_m^2 \sigma_p^2}{\sigma_m^2 + \sigma_p^2} \tag{2}$$

We also add a constant time variance, σ_t^2 at a constant rate to increase the variance if the cell is not updated.

We extend the formulation of [14] with additional features to generalize elevation mapping to scenarios including overhanging structures and walls. To remove outliers, we have an outlier check based on the Mahalanobis distance as also done in the previous work [14]. Here, we reject points that have a larger height difference than a certain Mahalanobis distance. In addition, to reject ceilings or overhanging objects while simultaneously capturing sloped terrain in the map, we introduced an exclusion area using ramp parameters as shown in Fig. 3. θ_a defines an angle of the exclusion area line. b, c and d determine the offset and maximum height as shown in Fig. 3. By ignoring the points above the shown lines, the map correctly expresses the small opening near the robot required for navigating through. Furthermore, there is an issue of vertical edges or walls if we naively apply the equation 1. The resulting height becomes lower than the actual height because there are multiple measurements of different heights on the same cell which are effectively averaged. To resolve this, we count the number of points that fall within the same cell, and if it is higher than a threshold, we ignore the points lower than the currently estimated height. Since this happens before the outlier detection, which increases the variance of the cell, this can make the edge sharper.

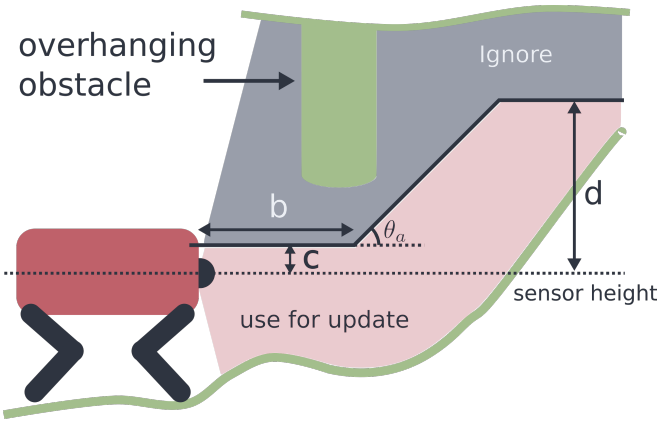

Fig. 3. Exclusion area for point measurements. To avoid creating the ceiling or close overhanging obstacle as an artifact, we introduce an exclusion area defined by parameters θ_a, b, c, d.

D. Height drift compensation

Since it is difficult to perfectly estimate the state onboard, the estimated sensor position with respect to the map often drifts. Height drift is the most influencing factor since it causes artifacts in the map. Therefore, a simple drift compensation method is developed to reduce this issue by calculating the error between the current sensor measurement and the map, and adjusting the map accordingly. As seen in Fig. 4, the error ϵ_i is calculated for each point p_i. To avoid projection errors in rough or steep areas, we only use relatively flat structure to calculate the error by thresholding the cell's traversability value (see II-G). Once all points have been processed, the average error $\frac{1}{n}\sum_i^n \epsilon_i$ is added to the elevation layer in the map, where n is the number of points used to calculate the error.

E. Visibility cleanup

In case there are dynamic obstacles, the old obstacles remain in the map until the estimated variance becomes large enough to pass the outlier rejection. Visibility cleanup is performed to remove this remaining estimate by performing ray casting. For each point, a ray from the sensor origin to the measurement point is cast and we iterate through this ray with a certain step size.

To clear dynamic obstacles, we check if a ray went through it by comparing the ray's height and the cell's estimated height. If a point on the ray is below the estimated height minus its variance σ_i^2, this cell is considered to be penetrated

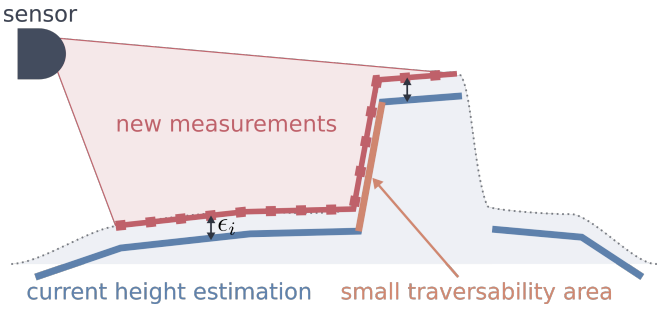

Fig. 4. Height drift compensation. To reduce the effect of the state estimation drift, we perform a height drift compensation using the new measurements. The height error between the new measurements and the current height estimation is calculated while excluding the area with low traversability area to prevent miscalculation using points with various heights at the same cell. At last, the whole map is adapted to the new measurements.

by the ray, and thus, the height value is removed:

$$\text{if } p_i^z < h_i - \sigma_i, \text{ remove.} \tag{3}$$

, where p_i^z represents a height of a point along a ray and h_i represents the height of the cell that the point p_i belongs to. As seen in Fig. 5, in case of the point p_j, p_j^z is higher than h_j so this cell is not removed.

The removal-law can cause jittering in between two consecutive updates, e.g. when the ray hits the obstacle at a small angle. To prevent this, we additionally check the last updated time and surface normal. The cell is removed only if the cell has not been updated for a specific time and fulfills the normal condition,

$$|\mathbf{r} \cdot \mathbf{n}| > \alpha_n, \tag{4}$$

where $\mathbf{r}$ represents a unit vector parallel to the ray and $\mathbf{n}$ represents the estimated surface normal of the cell. α_n is a user defined threshold.

F. Overlap clearance

When the robot navigates through multi floor environments, the 2.5D elevation map faces an issue of remaining old height estimates from the previous upper or lower floor. To handle this, we introduced an overlapping cleaner: It clears the height value whose difference from the robot's height is larger than a threshold if it is close to the robot. This simple method works well in multi floor environment where the robot goes up or down stairs.

G. Learning based traversability filter

An important benefit of performing elevation mapping on GPU is that data is already loaded into memory and we can therefore perform efficient terrain analysis using neural networks without any processing overhead caused by data transfer between CPU and GPU. Specifically, we deploy a simple Convolutional Neural Network (CNN) model trained to output traversability values for robot navigation [45]. It is implemented in PyTorch [36] and is light enough to run at full map update rate.

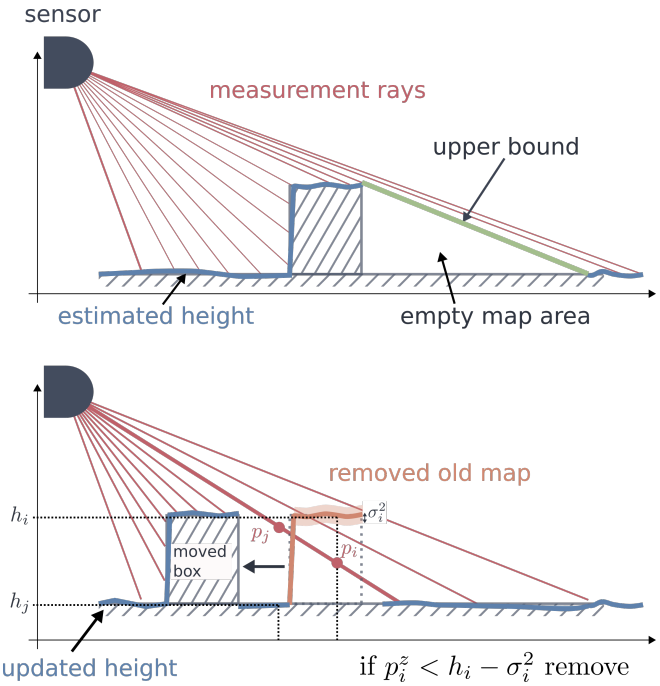

Fig. 5. Visibility cleanup and *upper bound* calculation using ray casting. The above figure shows the *upper bound* region. Based on the measured ray, the *upper bound* value in the area without a valid estimation is updated. The bottom figure shows the visibility cleanup. If the ray goes through the old map, that cell is removed to update the map with dynamic obstacles faster.

H. Height upper bound layer

When we perform the visibility cleanup, we also use the ray casting steps to compute a maximal terrain height for unobserved map cells, similar to the idea of virtual surfaces [19], which we store in a map layer dubbed *upper bound*. If a ray passes through a cell, we know that the ground height cannot be larger than the ray height, since we would have otherwise observed the ground. This is useful in case of obstacle occlusion, as holes in the map need to somehow be interpreted by the downstream modules. A navigation planner, for example, would typically either optimistically fill the holes with an image inpainting algorithm[47] or simply consider them untraversable. The *upper bound* layer helps distinguish between safe patches caused by obstacle occlusions (which typically exhibit small inclinations in the *upper bound*) and unsafe larger drops which cause steeper ray angles resulting in larger inclinations.

I. Features for legged locomotion

Elevation maps can be used not only for navigation, but also for foothold planning for legged robots. Some additional post processing is often required to use in optimization based locomotion controllers. We provide optional features that are particularly useful for such applications.

1) Minimum filter: The completeness of the elevation map is crucial for reliable and robust motion planning in model based controllers. We provide an implementation of the inpainting filter presented in [22], which replaces empty grid cells with the minimum found along the occlusion border.

2) Smoothing filter: Gaussian and box-blur filters have been used in [23], [22] for generating smooth representations of the terrain. Median filtering has been reported as being more effective for artifact rejection [22]. We provide combinations of several smoothing filters that can be found in the `opencv` library [6], including Gaussian, box-blur and median filter.

3) Plane segmentation: In an optimization-based approach to perceptive locomotion, the terrain is often represented by a set of primitive geometric shapes to optimize over. In favour of low computation time, it is beneficial to globally extract these features before the optimization is started. We provide algorithms to segment the elevation map into continuous planar regions. The outer boundary and potential holes contained inside the region are returned as polygons. This preprocessing is used in the whole-body MPC approach presented in [16].

III. Experiments and Results

To evaluate our mapping framework, we first compared our mapping framework with existing open source software [14] as a baseline. Then, we validate the usability of our mapping framework through legged locomotion and navigation experiments using the ANYmal platform [4]. The robot used in our experimens is equipped with two Robosense Bpearl [2] LiDARs or four Intel Realsense [1].

A. Baseline comparison

1) Feature evaluation: Here, we first evaluate the effectiveness of our features by comparing the quality of the map using the same data collected by the ANYmal (Fig. 6). In the first two row (Fig. 6 A-F), the robot was equipped with two Bpearls and the state estimation had a large height bias as shown by the trajectory of the robot (shown in red line). With our drift compensation, the gap between the old map and the latest map updated by the new sensor measurement was smaller than the one without this feature (Fig. 6 A-C). The map of the baseline creates a large gap and also some artifacts. This artifacts was due to the slow visibility cleanup rate as seen in the following experiment.

In the second row (Fig. 6 D-F), we evaluate the visibility cleanup feature using the same data as Fig. 6 A-C. In this setting, we disabled the drift compensation and compared the map quality. As seen in Fig. 6D, our map with visibility cleanup feature did not have the wall which is visible in Fig. 6E. This is because the new measurement could remove the old map through ray casting. The baseline framework has the visibility cleanup feature, however, the update is performed in a slower rate than the point cloud measurements to reduce the computational load. As seen in Fig. 6F this slow update rate could not clear all artifacts.

Lastly we compared the map with an overhanging obstacle in front of the robot. As seen in Fig. 6G-I, our method could exclude the obstacle while the baseline method represented it as a wall.

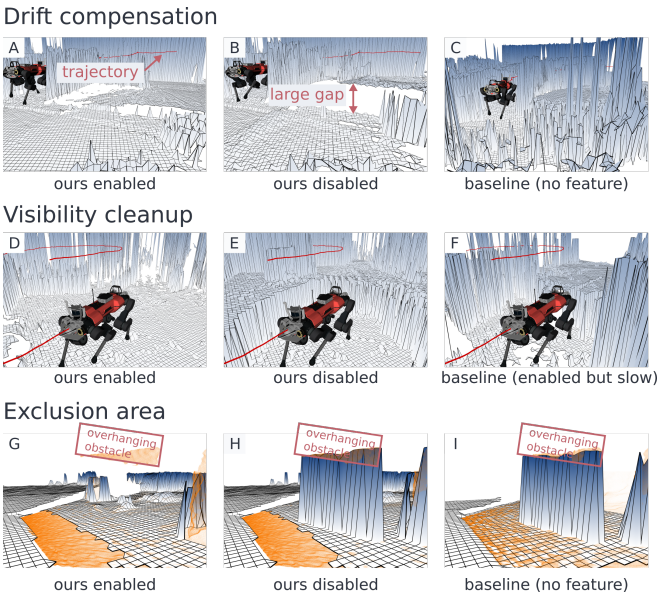

Fig. 6. Feature comparison. We evaluated the effectiveness of each features by comparing the map quality with feature disabled setting. In addition, we compared with the baseline method [14]. Drift compensation: when the odometry has a large height drift, it causes a large gap as seen in B. Our drift compensation filter can reduce this by matching the current map to the new measurement. Visbility cleanup: Our visibility cleanup is performed on every point cloud measurement instead of slower rate as done in the baseline. This results in much cleaner map as seen in D. Exclusion area: We placed an overhanging obstacle in front of the robot. As seen in G, our method could exclude this while the baseline method represented as an obstacle.

2) Processing time comparison: We compared the processing time of our elevation mapping pipeline on GPU with the baseline. We measured the calculation time of processing the point cloud on two devices, a desktop PC (Ryzen9 3950x, NVIDIA RTX 2080Ti), as well as a Jetson Xavier that is integrated on the robot. To measure the calculation time with different number of points, we randomly sampled from a point cloud data and republished to the mapping pipeline. We used the map size of 10×10 m with 4cm resolution and the calculation time was measured on GPU for our pipeline and on CPU for the baseline. In our mapping method, the processing includes all features as shown in Fig. 2: point cloud transformation, noise handling, height error count, drift compensation, height update, ray casting, traversability estimation and normal calculation. In the case of the baseline method, it includes the point cloud transformation, noise handling and updating each height but does not include other features such as visibility cleanup or filtering.

The result is shown in Fig. 7. Since our processing is done on GPU, the processing time remained short even for large numbers of points. While the baseline method's calculation time grew with a steeper rate with respect to the number of points. This shows that our mapping can process point cloud data more efficiently than the baseline although our mapping is performing additional computations such as traversability estimation and ray casting.

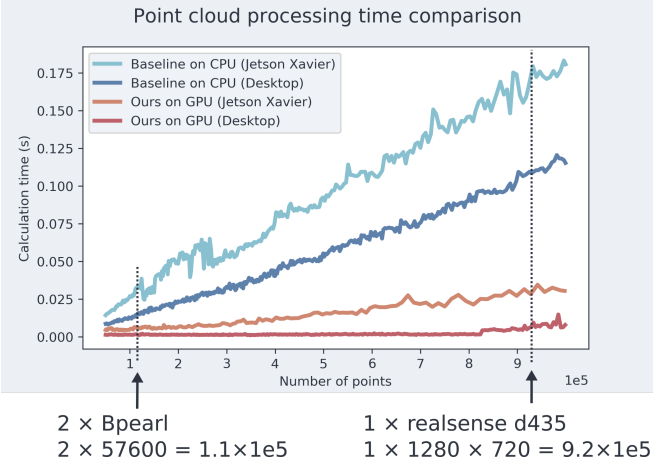

Fig. 7. Point cloud processing time comparison between our method and an existing implementation on CPU [14]. The calculation time increases more with the baseline method while it stays comparatively low with our method. We indicated number of points with 2xBpearl [2] and 1xrealsense [1] on the plot. Our mapping pipeline could process the data in real-time, while the baseline method had a considerable delay on onboard PC (Jetson).

TABLE I

CALCULATION TIME PER FEATURES.

number of points	43017	
point transform & z error count	1.194	ms
drift compensation	0.742	ms
height update & ray casting	0.648	ms
overlap clearance	0.003	ms
traversability	4.102	ms
normal calculation	0.168	ms
total	6.857	ms

TABLE II

POINT CLOUD PROCESSING FREQUENCY WITH DIFFERENT SENSORS.

Sensor	Number of points	map update Hz	sensor Hz
Realsense filtered	6276	49.4	60
Realsense raw	407040	16.1	60
Bpearl	43074	19.99	20

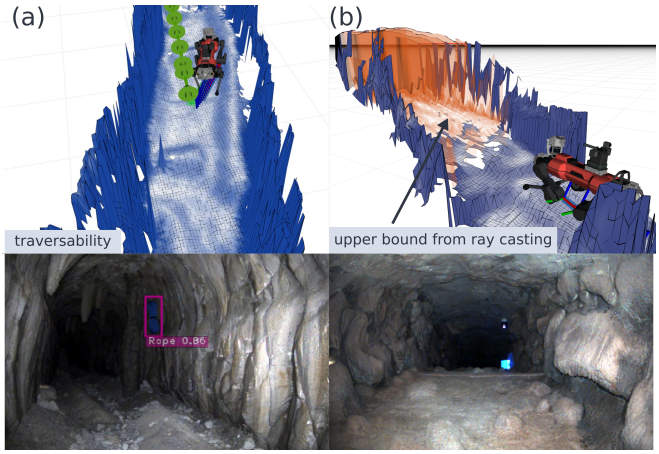

Fig. 8. Elevation map used during DARPA SubT Challenge final. The bottom images are from the on-board camera on the robot. (a) The robot navigating through a rough and narrow cave section. The traversability layer was used for local navigation. The blue color represents small traversability value and white means high traversability. (b) The robot was on a steep and narrow slope. At the end of the slope there was a flat part as shown in the bottom image. The orange map shows the upper bound layer that is calculated from the ray casting. This layer helps planning on a slope where the area are not visible due to occlusion.

B. Performance analysis

In addition, we measured the computational time per each features to analyze which component takes time. The map size setting was the same as III-A.2. The result is on TABLE I. We used Bpearl sensor data to measure the calculation time for each component on Jetson Xavier and collected 1000 measurements. As a result, the traversability filter took the most time compared to the other components. Therefore, improving this module could improve the performance in the future.

Additionally, to check the real-time performance on different sensor configuration, we measured the map update frequency on Jetson Xavier. This frequency shows how many times the map was updated using the sensor measurements. As seen in TABLE II, there are three settings: Realsense filtered, Realsense raw and Bpearl. The map setting was the same as III-A.2. The *Realsense filtered* setting uses a down-sampled point cloud using voxel filter implementation [39]. In this setup we could achieve the update rate of almost 50 Hz, which is fast enough for locomotion. The *Realsense raw* setting uses the raw point cloud coming from the sensor. Although this point cloud contains large number of point cloud, we could update the map in 16 Hz, which is still fast enough for our use case. The *Bpearl* uses the raw point cloud coming from the Bpearl sensor. Since this sensor is sparse compared to depth image of Realsense, we could update the map in the same frequency as the sensor measurements.

C. Locomotion and navigation applications

We validate the usability of our mapping framework through legged locomotion and navigation experiments using the ANYmal platform [4]. We will first present the usage of the elevation map for navigation in the context of the DARPA Subterranean Challenge (Fig. 8). As odometry source, we rely either on kinematics based leg odometry [5] or a fused odometry using a time offset compensating Extended Kalman Filter [28] that combines Inertial Measurement Unit (IMU) measurements and LiDAR SLAM [24] pose estimates.

1) Using traversability value for local path planning: An important component in the autonomous mission stack is the local navigation. Our elevation map was used as a basis for local navigation to choose a feasible path and preventing the robot to walk into dangerous places such as cliffs. Given a mid range target position defined from an exploration planner [27], the local planner [45] plans a path using the upper bound layer and the traversability layer.

2) Using elevation map for locomotion: To perceive the surrounding information, the reinforcement learning-based controller [30] sampled heights around each foot from our elevation map. More details are in section III-D.1.

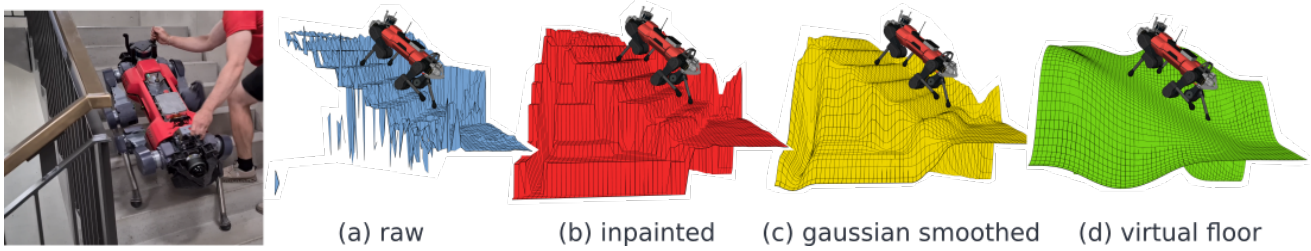

Fig. 9. Different height layers used in TAMOLS [22]: First, the occluded regions in the raw map are in-painted. Then, two additional layers are derived from it, gradually increasing the smoothness.

D. Legged locomotion

In addition to mobile robot navigation, our elevation map was used for perceptive locomotion. We demonstrated the usability with three different controllers: A reinforcement learning based controller and two model-based controllers.

1) Learning based locomotion: The controller [30], which was also used during the DARPA SubT Challenge, utilized the elevation information from our map. One distinct feature of our mapping approach was the height drift compensation. When using leg odometry [5] as a source of sensor position estimates in the walking experiments, we observed a tendency for vertical drift, especially when the robot was significantly slipping. The drift causes artifacts in the map that can appear as a step. Enabling drift compensation helped the controller achieving smoother locomotion. In long missions on challenging surfaces we observed that instead relying on the fusion based reference odometry with IMU and LiDAR SLAM combined with drift compensation reduces these issues even further. With this approach we prevent leg slippage induced odometry drift from directly affecting the elevation map quality.

2) Model based locomotion: Our elevation mapping framework has been used in two model based locomotion planners, utilizing the fast update rate. In [23], filters for computing terrain normal and standard deviation were used to compute a foothold score. The authors of [22] proposed to use a virtual floor (see Fig. 9) as base pose reference, computed with box-blur, dilation and median filters. In [16], a plane segmentation was used to define a foot hold constraint for optimization based controller.

IV. CONCLUSIONS

In this paper, we present an elevation mapping pipeline on GPU. Thanks to the GPU acceleration, the point cloud processing is more efficient compared to similar baseline software. In addition, several methods such as height drift compensation, upper bound calculation, or exclusion area were introduced to make the map more reliable. We showed that our mapping methods can address issues which cause inaccurate map with the baseline framework. We further demonstrated the usefulness of our mapping for both navigation and legged locomotion through hardware experiments. Our map was deployed during DARPA Subterranean Challenge and was used for local navigation as well as perceptive locomotion. Besides, our map supported successful application for legged locomotion over complex terrain. The features such as smoothness filters or plane segmentation were integrated into our mapping framework, making this as a convenient tool for legged locomotion research.

REFERENCES

[1] Intel realsense (2021, april). `https://www.intelrealsense.com/`.

[2] Rs-bpearl (2021, april). `https://www.robosense.ai/en/rslidar/RS-Bpearl`.

[3] Occupancy homogeneous map. `https://github.com/csiro-robotics/ohm`, 2018.

[4] ANYbotics. ANYmal. `https://www.anybotics.com/anymal-autonomous-legged-robot/`, 2021. [Online; accessed June-2021].

[5] Michael Bloesch, Marco Hutter, Mark A Hoepflinger, Stefan Leutenegger, Christian Gehring, C David Remy, and Roland Siegwart. State estimation for legged robots-consistent fusion of leg kinematics and imu. *Robotics*, 17:17–24, 2013.

[6] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[7] Russell Buchanan, Lorenz Wellhausen, Marko Bjelonic, Tirthankar Bandyopadhyay, Navinda Kottege, and Marco Hutter. Perceptive whole-body planning for multilegged robots in confined spaces. *Journal of Field Robotics*, 38(1):68–84, 2021.

[8] Cerberus. Team cerberus. `https://www.subt-cerberus.org/`, 2021. [Online; accessed June-2021].

[9] DARPA. Darpa subterranean challenge competition rules final event. `https://www.subtchallenge.com`, 2021. [Online; accessed June-2021].

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

[11] David D Fan, Kyohei Otsu, Yuki Kubo, Anushri Dixit, Joel Burdick, and Ali-Akbar Agha-Mohammadi. Step: Stochastic traversability evaluation and planning for risk-aware off-road navigation. *arXiv preprint arXiv:2103.02828*, 2021.

[12] Péter Fankhauser, Marko Bjelonic, C Dario Bellicoso, Takahiro Miki, and Marco Hutter. Robust rough-terrain locomotion with a quadrupedal robot. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5761–5768. IEEE, 2018.

[13] Péter Fankhauser, Michael Bloesch, Christian Gehring, Marco Hutter, and Roland Siegwart. Robot-centric elevation mapping with uncertainty estimates. In *Mobile Service Robotics*, pages 433–440. World Scientific, 2014.

[14] Péter Fankhauser, Michael Bloesch, and Marco Hutter. Probabilistic terrain mapping for mobile robots with uncertain localization. *IEEE Robotics and Automation Letters*, 3(4):3019–3026, 2018.

[15] Péter Fankhauser and Marco Hutter. A Universal Grid Map Library: Implementation and Use Case for Rough Terrain Navigation. In Anis Koubaa, editor, *Robot Operating System (ROS) – The Complete Reference (Volume 1)*, chapter 5. Springer, 2016.

[16] Ruben Grandia, Fabian Jenelten, Shaohui Yang, Farbod Farshidian, and Marco Hutter. Perceptive locomotion through nonlinear model predictive control. *submitted to IEEE Transactions on Robotics*, 2022.

[17] Ioannis Havoutis, Jesus Ortiz, Stephane Bazeille, Victor Barasuol, Claudio Semini, and Darwin G Caldwell. Onboard perception-based trotting and crawling with the hydraulic quadruped robot (HyQ). In

2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 6052–6057. IEEE, 2013.

[18] M. Herbert, C. Caillas, E. Krotkov, I.S. Kweon, and T. Kanade. Terrain mapping for a roving planetary explorer. In *Proceedings, 1989 International Conference on Robotics and Automation*, pages 997–1002 vol.2, 1989.

[19] Thomas Hines, Kazys Stepanas, Fletcher Talbot, Inkyu Sa, Jake Lewis, Emili Hernandez, Navinda Kottege, and Nicolas Hudson. Virtual surfaces and attitude aware planning and behaviours for negative obstacle navigation. *IEEE Robotics and Automation Letters*, 6(2):4048–4055, 2021.

[20] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous robots*, 34(3):189–206, 2013.

[21] Marco Hutter, Christian Gehring, Dominic Jud, Andreas Lauber, C Dario Bellicoso, Vassilios Tsounis, Jemin Hwangbo, Karen Bodie, Peter Fankhauser, Michael Bloesch, et al. Anymal-a highly mobile and dynamic quadrupedal robot. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 38–44. IEEE, 2016.

[22] Fabian Jenelten, Ruben Grandia, Farbod Farshidian, and Marco Hutter. Tamols: Terrain-aware motion optimization for legged systems. *submitted to IEEE Transactions on Robotics*, 2021.

[23] Fabian Jenelten, Takahiro Miki, Aravind E Vijayan, Marko Bjelonic, and Marco Hutter. Perceptive locomotion in rough terrain–online foothold optimization. *IEEE Robotics and Automation Letters*, 5(4):5370–5376, 2020.

[24] Shehryar Khattak, Huan Nguyen, Frank Mascarich, Tung Dang, and Kostas Alexis. Complementary multi–modal sensor fusion for resilient robot pose estimation in subterranean environments. In *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1024–1029. IEEE, 2020.

[25] D Kim, D Carballo, J Di Carlo, B Katz, G Bledt, B Lim, and Sangbae Kim. Vision aided dynamic exploration of unstructured terrain with a small-scale quadruped robot. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2464–2470. IEEE, 2020.

[26] J Zico Kolter, Youngjun Kim, and Andrew Y Ng. Stereo vision and terrain modeling for quadruped robots. In *2009 IEEE International Conference on Robotics and Automation*, pages 1557–1564. IEEE, 2009.

[27] Mihir Kulkarni, Mihir Dharmadhikari, Marco Tranzatto, Samuel Zimmermann, Victor Reijgwart, Paolo De Petris, Huan Nguyen, Nikhil Khedekar, Christos Papachristos, Lionel Ott, Roland Siegwart, Marco Hutter, and Kostas Alexis. Autonomous teamed exploration of subterranean environments using legged and aerial robots. In *2022 IEEE International Conference on Robotics and Automation (ICRA)*, Philadelphia (PA), USA, 2022. IEEE.

[28] S Lynen, M Achtelik, S Weiss, M Chli, and R Siegwart. A robust and modular multi-sensor fusion approach applied to mav navigation. In *Proc. of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, 2013.

[29] Yuntao Ma, Farbod Farshidian, Takahiro Miki, Joonho Lee, and Marco Hutter. Combining learning-based locomotion policy with model-based manipulation for legged mobile manipulators. *IEEE Robotics and Automation Letters*, 7(2):2377–2384, 2022.

[30] Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 7(62):eabk2822, 2022.

[31] Chuong V. Nguyen, Shahram Izadi, and David Lovell. Modeling kinect sensor noise for improved 3d reconstruction and tracking. In *2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization Transmission*, pages 524–530, 2012.

[32] ROYUD Nishino and Shohei Hido Crissman Loomis. Cupy: A numpy-compatible library for nvidia gpu calculations. *31st confernce on neural information processing systems*, 151, 2017.

[33] Helen Oleynikova, Zachary Taylor, Marius Fehr, Roland Siegwart, and Juan Nieto. Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[34] Timothy Overbye and Srikanth Saripalli. G-vom: A gpu accelerated voxel off-road mapping system, 2021.

[35] Yiyuan Pan, Xuecheng Xu, Xiaqing Ding, Shoudong Huang, Yue Wang, and Rong Xiong. Gem: Online globally consistent dense elevation mapping for unstructured terrain. *IEEE Transactions on Instrumentation and Measurement*, 70:1–13, 2021.

[36] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[37] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[38] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Conference on Robot Learning*, pages 91–100. PMLR, 2022.

[39] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[40] Maximilian Stölzle, Takahiro Miki, Levin Gerdes, Martin Azkarate, and Marco Hutter. Reconstructing occluded elevation information in terrain maps with self-supervised learning. *IEEE Robotics and Automation Letters*, 7(2):1697–1704, 2022.

[41] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

[42] Marco Tranzatto, Frank Mascarich, Lukas Bernreiter, Carolina Godinho, Marco Camurri, Shehryar Masaud Khan Khattak, Tung Dang, Victor Reijgwart, Johannes Loeje, David Wisth, et al. Cerberus: Autonomous legged and aerial robotic exploration in the tunnel and urban circuits of the darpa subterranean challenge. *Journal of Field Robotics*, 2021.

[43] Rudolph Triebel, Patrick Pfaff, and Wolfram Burgard. Multi-level surface maps for outdoor terrain mapping and loop closing. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2276–2282, 2006.

[44] Octavio Antonio Villarreal-Magaña, Victor Barasuol, Marco Camurri, Michele Focchi, Luca Franceschi, Massimiliano Pontil, Darwin G. Caldwell, and Claudio Semini. Fast and continuous foothold adaptation for dynamic locomotion through cnns. *IEEE Robotics and Automation Letters*, 4(2):2140–2147, 2019.

[45] Lorenz Wellhausen and Marco Hutter. Rough terrain navigation for legged robots using reachability planning and template learning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6914–6921, 2021.

[46] Martin Wermelinger, Péter Fankhauser, Remo Diethelm, Philipp Krüsi, Roland Siegwart, and Marco Hutter. Navigation planning for legged robots in challenging terrain. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1184–1189, 2016.

[47] Bowen Yang, Lorenz Wellhausen, Takahiro Miki, Ming Liu, and Marco Hutter. Real-time optimal navigation planning using learned motion costs. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9283–9289, 2021.