OVPC Mesh: 3D Free-space Representation for Local Ground Vehicle Navigation

Fabio Ruetz[1], Emili Hernández[2], Mark Pfeiffer[1], Helen Oleynikova[1], Mark Cox[2], Thomas Lowe[2], Paulo Borges[2]

Abstract— This paper presents a novel approach for local 3D environment representation for autonomous unmanned ground vehicle (UGV) navigation called *On Visible Point Clouds Mesh* (OVPC Mesh). Our approach represents the surrounding of the robot as a watertight 3D mesh generated from local point cloud data in order to represent the free space surrounding the robot. It is a conservative estimation of the free space and provides a desirable trade-off between representation precision and computational efficiency, without having to discretize the environment into a fixed grid size. Our experiments analyze the usability of the approach for UGV navigation in rough terrain, both in simulation and in a fully integrated real-world system. Additionally, we compare our approach to well-known state-of-the-art solutions, such as Octomap and Elevation Mapping and show that OVPC Mesh can provide reliable 3D information for trajectory planning while fulfilling real-time constraints.

I. INTRODUCTION

Unmanned Ground Vehicles (UGVs) have many potential applications in a broad range of industries, such as agriculture, search and rescue, and mining. However, many of these applications require navigation in rough, unstructured, or complex terrain. In such scenarios, 2D approaches fail to accurately represent the environment, and 2.5D maps still struggle to handle objects such as overhangs. 3D maps can accurately represent the complete environment, but usually have to be discretized to a minimum voxel size. A coarse discretization is computationally efficient, yet does not allow for an accurate representation of small objects, non-axis-aligned surfaces, or inclines. A fine one is computationally not tractable for real-time applications.

We present a local, robot-centric 3D map representation that explicitly models the free space around the robot to allow fast, conservative local planning. Our approach, *On Visible Point Clouds Mesh* (OVPC Mesh), molds a watertight 3D mesh onto a point cloud simplified with the *general hidden point removal operator* [1], as shown in Figure 1. Using this approach, a point cloud can be reduced to the visible points from a certain viewpoint. The volume encapsulated by this mesh is guaranteed to be a conservative estimation of free space. The mesh represents the information boundary, as the volume inside the mesh is free space while the outside is considered unknown. We show how the constructed OVPC
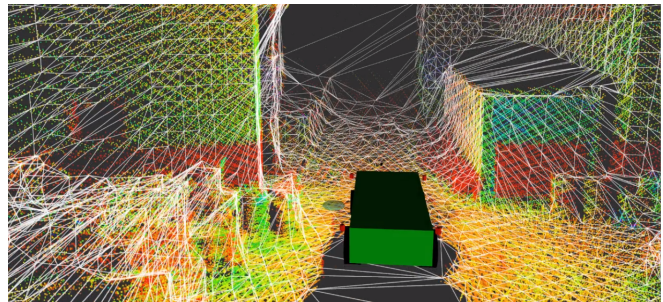
Fig. 1. Raw local point cloud (colored by intensity) obtained from a 3D LiDAR and the extracted watertight mesh (white edges). The mesh encapsulates the robot and gives a conservative three-dimensional free-space estimate of the local robot surrounding. This environment representation can also be used for rough-terrain motion planning.

mesh can be used to estimate traversability, which can in turn be used directly in a variety of path planning techniques.

Compared to voxel-based methods, the associated computational complexity is low, which allows for a real-time deployment on robotic platforms with limited computational resources.

The capabilities of our approach are analyzed both in simulation and on a real robotic platform. We show a comparison to existing state-of-the-art 2.5D and 3D solutions, namely Elevation Mapping [2] and Octomap [3], respectively. Our results show successful autonomous navigation of a UGV in both structured and unstructured environments, including steep off-road inclines and overhangs. Our contributions can be summarized as follows:

- Application of the general hidden point removal operator to generate a conservative free space representation from a raw 3D point cloud for navigation purposes.
- Traversability analysis on the 3D watertight mesh for rough terrain navigation.
- System evaluation on a UGV, using the traversability information for local planning purposes.

II. RELATED WORK

A common map representation for ground vehicles is the 2D grid map [4]. Such maps are computationally efficient and a robust solution for ground robot navigation in structured environments [5]. 2D grid maps have even been extended to navigation on rough terrain by performing terrain classification for each cell of the map [6]. However, since the map is still only 2D, it is very limited in what kind of environments it can represent – and attempting to navigate in less structured

environments, such as those featuring overhangs leads to either missing valid paths or executing unsafe ones.

Another common map representation for UGV operation in unstructured environments is the digital elevation map (DEM) [7]. DEMs store a height estimate in each grid cell to capture the 2.5D geometry of the environment. Classical DEMs suffer from two short-comings for our target applications: first, they lack the ability to represent overhangs, and second, their world-centric global formulation require accurate global pose estimates. These are often not available on real robots, where local odometry errors can accumulate to large global drifts. Droeschel et al. [8] attempt to overcome the need for consistent global localization by expressing the map in a robot-centric frame. Additionally, Fankhauser et al. [9] also consider motion estimation uncertainty into their probabilistic 2.5D map. Our approach is also robot-centric, but in contrast to these methods, is full 3D, allowing it to handle overhangs and other complex geometry.

Triebel et al. [10] extend DEMs to global multi-level surface maps to overcome the overhangs challenge and loop-closures. They do not model overhangs explicitly, but require a minimum height distance between two surfaces to ensure a safe traversability; otherwise they are closed off.

The easiest way to accurately represent overhangs is in a full 3D map. Many approaches exist, most of which discretize the environment into some fixed minimum voxel or block size due to the huge computational and storage requirements of such maps. Octomap [3] uses octrees to allow the system to scale to large spaces, but the octree structure requires a tree traversal for look-ups or insertions, slowing down map queries. Voxblox [11] instead uses a voxel hashing strategy, which only allocates fixed-size blocks in those regions of the map that contain data to scale to large areas without slowing down map access. In contrast to Octomap and Voxblox, ewok [12] uses a fixed-size robot-centric map and a ring buffer to store 3D occupancy information. In contrast to these methods, our approach is a 3D, robot-centric map representation that does *not* rely on a fixed grid-size discretization, allowing us to more accurately represent local structure of different scales.

Krüsi et al. [13] present a 3D mapping approach specifically designed for UGV navigation on rough terrain. The map is stored as a global point cloud and the surface is reconstructed only locally for traversability analysis. Despite avoiding discretization issues, this strategy requires precise global localization and point cloud alignment to avoid global map misalignment.

Meshes are another way to represent the environment in 3D. For example, [14] proposes a fast mesh reconstruction algorithm for noisy, non-uniform point clouds, focusing on the mesh accuracy as well as the meshing speed at near real-time speed. Gingras et al. [15] generate navigable spaces as triangular 3D meshes for a ground robot. These meshes represent only the surface areas the robot can traverse, excluding all non-traversable areas, which is conceptually similar to our approach. However, map updates with new sensor measurement take multiple seconds, making it un-

suitable for real-time applications. Our method demonstrates runtimes comparable to or faster than discretized methods, without trading off accuracy on small obstacles.

III. METHOD

The goal of our approach is to develop a conservative 3D representation of the environment which is computationally efficient and can later be used for trajectory planning for a ground robot. In order to reduce the overall size of the point cloud for mesh generation we use the *general hidden point removal* operator (GHPR) where the *visible* points from a certain viewpoint are determined. From the sparsified point cloud, a watertight 3D triangle mesh is generated, bounding the local free space, which is accurate enough to determine the traversability of a car-size UGV.
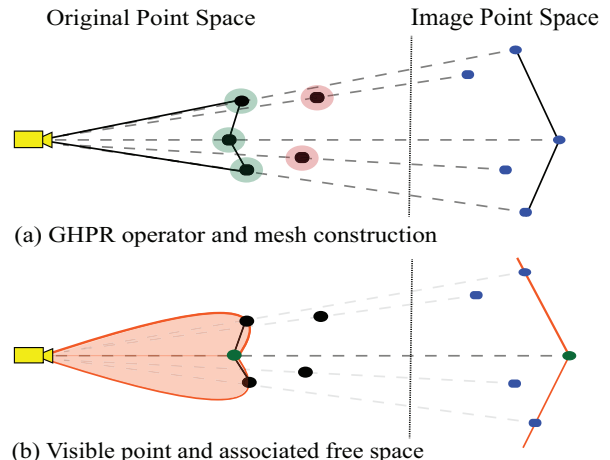
A. Mesh Generation

Fig. 2. Mesh generation based on the GHPR algorithm: (a) Transformation to generate the image points (blue) from the original point set (black). Convex hull over image points generates connectivity between the points in the image space and can be applied to the original point set(black full lines). Points in the original space are visible (green transparent) if their images lie on the convex hull, else they are considered non-visible (red transparent). (b) Extending the edges from a visible point (green) to its neighbors to infinity are mapped to curves in the original point space. These curves bound an area or a volume in 2D or 3D (transparent orange), and are guaranteed to be free of other observations.

In order to make the meshing operation real-time capable, we use GHPR, which is commonly used in computer graphics to find visible points on objects from a given viewpoint, without explicitly reconstructing surfaces. As shown in Figure 2, the original point set (OPS), which is obtained from a local point cloud, is transformed to an image point set (IPS) using a radial transformation (see Figure 2). Making use of the transformation, the visible point classification is reduced to the problem of forming a convex hull over the (transformed) IPS. A point is considered visible if its image lies on the convex hull in the image space.

As Figure 2 shows, the connectivity information of the convex hull in the image space can be transferred to the original space in order to construct a triangle mesh on the visible points. The resulting mesh is closed and watertight,
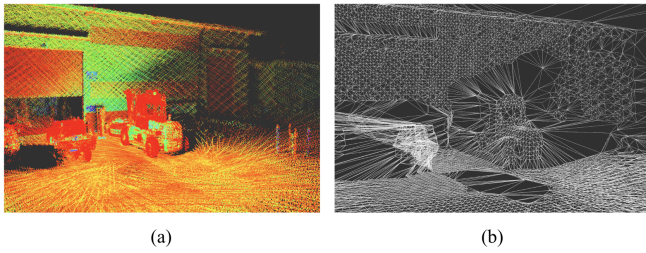
Fig. 3. Example of an OVPC Mesh from point cloud obtained with our UGV: (a) intensity colored point cloud; (b) corresponding watertight triangle mesh generated from the point cloud.

and it only depends on the visible points (from the current viewpoint) of the OPS.

By construction, this method solves the visibility of point classification dependant on free space associated with each point in the OPS; free from any other points. The shape is controlled by the kernel and limited by the point's neighbours (Figure 2(b)). The kernel parameter γ defines the minimum required space for a point to be visible. This results in all visible points lying on the boundary region free of other observations. We use the exponential kernel and given the shape of the curves forming the boundary, and we assume that given two visible points their connecting edge will lie within free space. Under the assumptions of perfect sensor readings and all edges lying in free space, the generated mesh is a conservative bound of the free space surrounding the robot. The interested reader should refer to [1] for more detailed explanation.
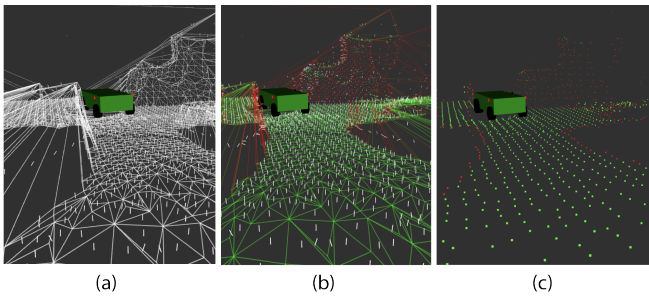

Fig. 4. Assessing the traversability on the mesh faces and generating a map for navigation. Figure (a) shows the generated water-tight mesh and the normal estimation of each face as short white line segments, (b) shows each face classified in traversable (green) and non-traversable (red) using geometric constraints described in Section III-B and (c) shows the extracted point cloud representation where each point is classified based on its traversability. Note that the outsides of the traversable area are strictly non-traversable which allows for conservative navigation.

B. Map Representation for Path Planning

Once the mesh is generated, we perform a traversability analysis on each polygon to constrain the local free space according to our vehicle specifications. For this analysis, we use the surface angle α: this is the angle between the surface normal and the gravity-aligned world frame. A mesh polygon is *traversable* if: the surface angle is smaller than the threshold $\alpha_{\max}$ and lies within the interval [$-\pi/2$, $\pi/2$]; and the maximum height difference among its vertices is smaller than $\delta_{h,\max}$. Otherwise the polygon is set as *non-traversable*.

Inspired by [13], a point cloud is formed from the vertices of the mesh. Each point is assigned a normal vector, representing the estimated surface, and a binary traversability score. The normal is calculated as mean of all normals from the adjacent faces. A point is only considered traversable if all the neighboring faces are traversable, see Figure 4(c). This point cloud is used as a map representation for local planning.

IV. RESULTS

In this section we describe our specific hardware and software implementation followed by the results in simulation, the ones obtained with a fully integrated UGV and a comparison of our approach against two state-of-the-art methods.

A. Hardware and Software Setup

A John Deere Gator automated by CSIRO [16] was used as the test platform. The vehicle is equipped with a Velodyne PUCK VLP-16 LiDAR and a Microstrain GX3 IMU. The LiDAR is mounted at a height of $1.88\,\mathrm{m}$ above the robot frame at a $45°$ angle, as illustrated in the bottom right of Figure 7. The LiDAR assembly is additionally mounted on a spinning base, which rotates at approximately $0.5\,\mathrm{Hz}$ [16], and LiDAR measurements are streamed in at $20\,\mathrm{Hz}$.

Our mesh approach requires the LiDAR sensor readings and the local pose transforms [16] for our pipeline. No global localization is needed during the mesh construction and the local path computation steps.
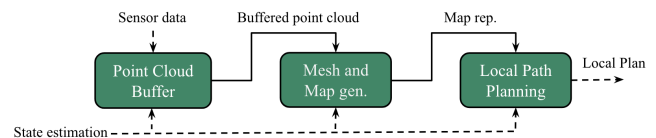

Fig. 5. Navigation pipeline overview: the sensor data is accumulated in the point cloud buffer, filtered and sub-sampled; then, the mesh and map generation module computes the watertight triangle mesh using the GHPR operator, which is then used for traversability analysis. Finally, the local planner generates a trajectory from the map, which is passed on to a trajectory follower.

The system was implemented in C++ using ROS [17], running on an external Dell Precision 7520 (Intel i7-4910MQ CPU@$2.90\,\mathrm{GHz}$ and $16GB$ of RAM) connected via Ethernet. An overview of our processing pipeline is depicted in Figure 5 and consists of the following three steps:

1) Point cloud buffer: In the first step a point cloud buffer aligns, filters, sub-samples and crops the data to generate a robot-centric point cloud from a fixed number (20-30) of previous scans. At the upper edges of the bounding box used to crop and generate the point cloud 4 virtual points are added to prevent the mesh from collapsing in flat environments. In our tests, the resulting scans contain between 6,000 to 27,000 points depending on the environment.

2) Mesh and map generation: A watertight triangle mesh is then generated around the robot and the traversability of the faces is evaluated. The mesh vertices are used to compute the visible point cloud used for path planning, where each

point contains a normal and traversability information. This step is described in detail in Section III-A and III-B.

3) Local path planning: We used the RRTconnect [18] sampling based path planner from OMPL [19] in combination with the Reeds-Shepp [20] state space to generate a local plan given a local goal. The local goal is obtained from a given global plan and has to lie within the mesh at a minimum distance ahead of the robot. During planning, each sampled state in $SE(2)$ is projected onto the map using the closest visible point in Euclidean space. The state is aligned to the local surface using the normal of the visible point, while preserving the heading. Collisions are detected by checking all visible points within a bounding box according to the platform dimensions, which is aligned with the sampled pose. Re-planning is enforced if (i) the current path is in collision after a map update, (ii) the current goal is too close to the current pose of the robot or (iii) a time threshold has been exceeded ($1\,\mathrm{s}$). The maximum computation time for planning is set to $0.1\,\mathrm{s}$. If successful, the local path is passed on to a trajectory follower [21].

B. Evaluation in simulation

Merging multiple point cloud scans together can lead to misalignments. Thus, we explore how our method performs when merging noisy, misaligned scans and how well the underlying geometry can be described by using our method instead of the raw point clouds. An accurate estimation of the geometry is important, as the traversability estimation depends on the normals of the mesh faces. We benchmark our approach against an other normal estimation technique [22] based on raw point clouds. We use the angular error to evaluate the accuracy and variance between the estimated normal and ground truth. This was evaluated in a $20\,\mathrm{m}\times20\,\mathrm{m}$ scene with a flat ground plane and an inclined plane with a defined slope α. We vary α between $0°$ and $35°$ with $0.1°$ increments to determine if the inclination has any effect on our normal estimation.

To simulate noisy bundled LiDAR measurements, we bundle 5 simulated point clouds together, each containing points spaced $0.2\,\mathrm{m}$ apart and $\pm0.05\,\mathrm{m}$ uniform noise in each axis. A uniform error in orientation of $\pm0.5°$ and translation of $0.1\,\mathrm{m}$ was added to the ground and slope, representing the pose uncertainty. Each scene is evaluated for a given angle α. The robot was positioned in the centre of the ground plane with the viewpoint at $1.88\,\mathrm{m}$ above ground (LiDAR mount).

As a comparison we use the standard normal estimation method [22] available in PCL [23] to estimate the normal of a point. A radius of $r = 0.3\,\mathrm{m}$ was used to perform the nearest neighbour search.

We use the angular error as the performance metric, i.e. the angular distance between the estimated and the ground truth normals.

The top plot in Figure. 6 shows our method mean angular error of $3.25°$ over the full range of slopes. The variance is bound and consistent for all tested slope angles. In practice this error can be observed by our traversability estimation when classifying traversable points as non-traversable. This
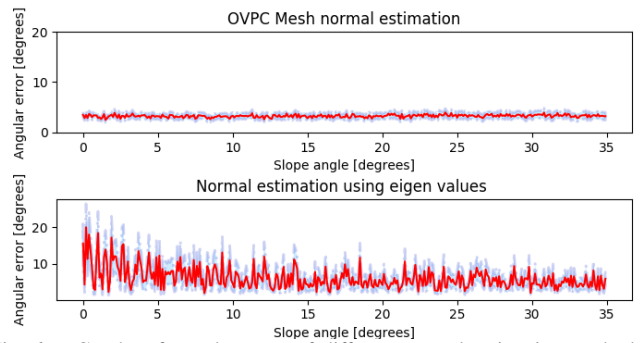

Fig. 6. Graphs of angular error of different normal estimation methods on noisy, misaligned point clouds. Mean angular error (red) and $\pm\sigma$ (blue). **Top:** The angular error is consistent, mean of $3.25°$ over all slope angles. The variance is also lower comparing to the method using eigenvalues. **Bottom:** The method using eigenvalues shows a higher angular error on average and higher variance.

can be mainly observed on the ground plane, where small triangles are oriented in such a way that they close the mesh to make it watertight which potentially results in steep inclines.

The eigenvalue method [22] shows significantly worse results for the angle estimates given the same noisy point cloud data, with higher angular error and variance. Our method uses the local surroundings of a mesh surface to generate the normal estimations. We adjusted the radius for nearest neighbor search of the eigenvalue method to match the size of this area to obtain fair results. Increasing the number of nearest neighbours or the coverage radius will make the estimation smoother and reduce the variance in this scenario, but will lead to biased angle estimation in the presence of more obstacles and complex environments. Also, as shown in [13], it requires additional computations on the point cloud to capture discontinuities and surface roughness. OVPC Mesh captures those situations implicitly due to coverage area used for the surface estimation. In essence this experiment shows that our proposed method is capable of accurately estimating the local geometry using noisy point cloud scans.

C. Experimental results

We also conducted a real-world evaluation of the fully integrated system by performing three fully autonomous long runs navigating a total distance of more than $1.5\,\mathrm{km}$, which capture a variety of different environments at the Queensland Centre of Advanced Technology (QCAT) in Brisbane, ranging from structured industrial sites to unstructured slopes on steep inclines. Examples of the different scenarios are shown in Figure 9. Parameters used for the experiments are chosen as follows: $\gamma = -0.03$, $\delta_{h,\mathrm{max}} = 0.25\,\mathrm{m}$, $\alpha_{\mathrm{max}} = 30°$, the viewpoint coincides with the LiDAR mount position and the velocity was set at $1\frac{\mathrm{m}}{\mathrm{s}}$. In general, the map representation performed well in structured and semi-structured environments, which were encountered on all 3 runs. Thin obstacles, such as posts, small trees or walls were accurately detected and avoided while the global plan was followed. The robot successfully navigated through these areas without any human intervention.

TABLE I
STATISTICS OF THE RUNS SHOWN IN IN FIGURE 7.

Run	Time [s]	Distance [m]	Color
Run 1	542	$\sim$605	Red
Run 2	503	$\sim$425	Yellow
Run 3	520	$\sim$480	Green

The end of yellow and red run consisted of a steep slope in a unstructured area. Driving up this slope introduced severe vibrations into the system. In general, this led to an increase in noise but the traversability analysis was still robust enough for navigation. In several instances, strong bumps and the resulting errors in the state estimation were observed to increase the noise such that the traversability estimation failed. This forced the robot to stop and re-plan in order to find a new trajectory, resulting in jerky movements. Note that these issues appear only with the fully integrated system when navigating in rough terrain and were not introduced by our OVPC Mesh method.
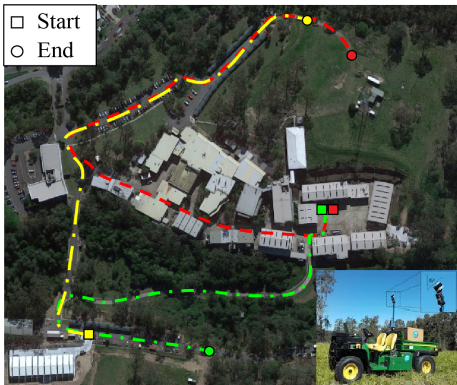

Fig. 7. Three different fully autonomous runs performed at QCAT. The overall covered distance is more than 1.5 km and includes both structured and off-road environments. **Bottom right:** The John Deere Gator test platform. The LiDAR sensor is mounted at a $45°$ angle and on a rotating base, allowing the robot to observe more of its environment even with a limited vertical field of view.

D. Comparison to other methods

We compare our method to two map representations commonly used for robot path planning: Octomap [3], a full 3D representation, and Elevation Mapping [2], a 2.5D height map. The purpose is to evaluate our method in terms of map representation quality and computational performance in challenging environments for UGV navigation.

1) Runtime benchmarking: We compare the runtime of generating a local map for planning from our method to Octomap and Elevation Mapping. For each computation a buffered point cloud and corresponding pose was used. These point clouds are the result of our point cloud buffer and contain 8'000- 27'000 points. Pose and point cloud were randomly chosen from our real-world data sets and are from all over the test site seen in Figure 7.

Figure 8 shows the runtime distributions of OVPC Mesh, Octomap and Elevation Mapping. Our method is not only faster on average ($\mu = 16.78$ ms (our method) vs. 30.91 ms (Octomap) vs. 36.68 ms (Elevation Mapping)) but also has a lower spread and therefore more predictable runtimes
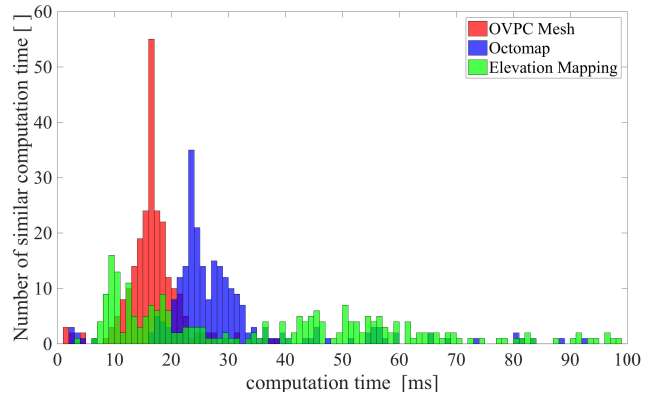

Fig. 8. Single point cloud map generation computation time histograms: OVPC Mesh (red) requires a mean time of 16.78 ± 5.04ms. Octomap (blue) requires 30.91 ± 18.66ms to integrate the same data. Elevation Mapping (green) generates the fused map in 36.68 ± 25.10ms.

($\sigma = 5.04$ ms vs. 18.66 ms vs. 25.10 ms), which is especially desirable for fully integrated real-world systems.

2) Handling of overhangs: Overhanging obstacles express a challenge for UGV navigation pipelines. In rough terrain this requires a simultaneous traversability analysis of the ground and an analysis of the overhangs. Since the LiDAR sensor is mounted at an exposed position at the very top of the vehicle, we need to pay particular attention to overhangs for autonomous navigation of our system.

Figure 10 (d) shows an example of an overhanging structure at the QCAT car park, which has sunroofs above the parking spaces. Figure 10 (a) depicts our 3D world representation with traversable faces colorized in green and non-traversable in red. The roof and ground are clearly separated w.r.t. traversability and the mesh extends from under the roof, allowing for local navigation. This is further illustrated in a wide angle shot (Figure 10 (e)). Additionally, the figure also shows the robustness of our approach in terms of capturing thin objects such as poles and small trees, and its traversability classification in a challenging environment.

Figure 10 (b) shows the 3D voxel representation of the overhang scenario using Octomap. It captures the overhang accurately and also extends from under the overhang.

In addition to Octomap, we tested Elevation Mapping in the overhang test scenario. The result using this 2.5D approach is depicted in Figure 10 (c). Despite being capable of modeling the area not covered by the roof sufficiently well, the area underneath the roof is not modeled at all. We assume that this is caused by the point exclusion heuristic, which operates based on the point height and distance to the sensor. Tuning parameters could not mitigate this effect.

V. DISCUSSION

OVPC Mesh has shown to be suitable for local UGV navigation in structured and unstructured environments. However, there are still several open challenges which are listed below. Although more robust than raw point clouds, the mesh generation process is sensitive to outliers in the LiDAR data, which can make the mesh unusable for planning. Applying a voxel filter, minimum 2 measurements per voxel to generate a
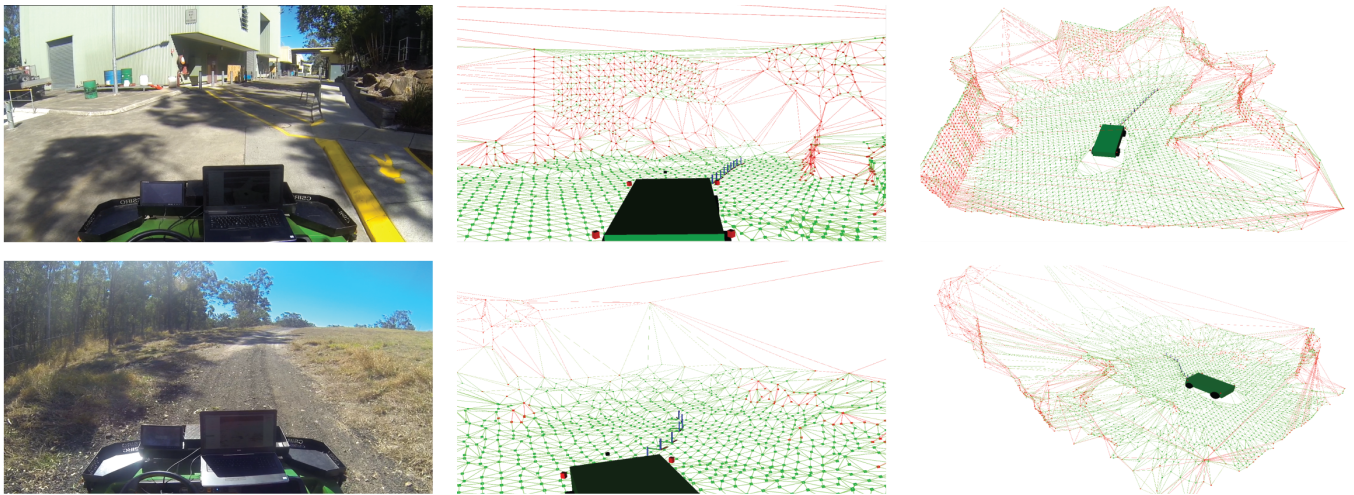
Fig. 9. The left column depicts the view from an on board camera during the autonomous navigation in structured (**top**) and unstructured (**bottom**) environments. The middle column shows the same scene but from the view of the robot inside of the mesh. The faces of the mesh are classified as traversable (green) and non-traversable (red). The local plan can be seen as a series of axes extending from the front of the robot. The right column shows the same scene from outside of the mesh and visualizes the size of the area considered for local navigation.
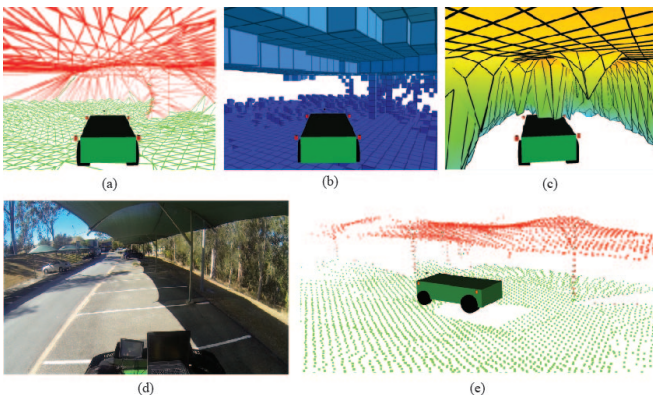

Fig. 10. A comparison of how different mapping techniques handle overhangs, shown from the UGV on-board camera in (d). (a) shows our method, with the mesh colored by non-traversable (red) and traversable (green), shown from a different perspective in (e), where thin obstacles, poles and thin trees, are accurately classified. The mesh extends from under the obstacle, allowing for local navigation. In (b), Octomap captures the sun cover and ground accurately. In (c), Elevation Mapping struggles with overhang, roof and floor outside of the overhang are reconstructed but not under the overhang.

point, has shown to be sufficient to remove the most extreme outliers. The discretization of the voxel filter is required for downsampling and use centroids of voxels instead of its centers, reducing the discretization error introduced.

Furthermore, the kernel parameter γ, which determines the free space required for a point to be visible, has a strong effect on the mesh generation (see Section IV-A.2). Empirical evaluation showed that values $\gamma \in [-0.01, -0.03]$, in combination with the exponential kernel [1], generate the most suitable results for UGV navigation in the environments we target in this work. When choosing the kernel values too close to zero nearly all points become visible and are connected by edges, as little space is required per point to be visible. Larger values will result in a sparser mesh. The values chosen should be in accordance with the spacing of the points in the point cloud and is application and sensor dependent.

Using the convex hull algorithm to compute a triangular mesh ensures that it is watertight. As a drawback, this process also generates triangles that do not represent the environment but connect points at the boundaries, where no LiDAR data is provided. However, such blind spots (like negative obstacles) are a common issue linked to local sensing and it is beyond the scope of this paper.

Lastly, the mesh is viewpoint dependent and any change on it will require to generate a new mesh and re-compute the path on it, which is suitable for our UGV applications since the average computation time of the whole process is less than $20\,\mathrm{ms}$.

VI. CONCLUSIONS

In this paper, we presented a novel method to build a local 3D free space representation, and showed its applicability in UGV rough terrain navigation. We generated a local map consisting of a watertight, closed mesh which is a conservative representation of the visible free space, and computed the traversability of each mesh face to generate a map representation suitable for local path planning according to our vehicle constraints.

The accuracy of our traversability estimation was validated in simulation, even in the presence of significant sensor noise. We demonstrated our approach through autonomous navigation on a fully integrated real system, navigating through structured, semi-structured, and unstructured environments, including challenges of low hanging ceilings and rough off-road terrains with steep inclines. Lastly, we compared our approach to two state-of-the-art solutions both qualitatively and quantitatively. The results show that our approach provide an environment representation suitable for autonomous rough terrain navigation while being computationally efficient compared to existing methods.

REFERENCES

[1] S. Katz and A. Tal, "On the visibility of point clouds," in *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec 2015, pp. 1350–1358.

[2] P. Fankhauser and M. Hutter, *A Universal Grid Map Library: Implementation and Use Case for Rough Terrain Navigation*, A. Koubaa, Ed. Springer, 2016, vol. 1.

[3] 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, no. 3, pp. 189–206, 2013.

[4] D. V. Lu, D. Hershberger, and W. D. Smart, "Layered costmaps for context-sensitive navigation," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2014, pp. 709–715.

[5] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige, "The office marathon: Robust navigation in an indoor office environment," in *2010 IEEE International Conference on Robotics and Automation*, May 2010, pp. 300–307.

[6] M. Hselich, N. Handzhiyski, C. Winkens, and D. Paulus, "Spline templates for fast path planning in unstructured environments," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2011, pp. 3545–3550.

[7] S. Lacroix, A. Mallet, D. Bonnafous, G. Bauzil, S. Fleury, M. Herrb, and R. Chatila, "Autonomous rover navigation on unknown terrains: Functions and integration," *The International Journal of Robotics Research*, vol. 21, no. 10-11, pp. 917–942, 2002.

[8] D. Droeschel, M. Schwarz, and S. Behnke, "Continuous mapping and localization for autonomous navigation in rough terrain using a 3D laser scanner," *Robotics and Autonomous Systems*, 2017.

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

[10] "Multi-level surface maps for outdoor terrain mapping and loop closing," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006, pp. 2276–2282.

[11] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: Incremental 3D Euclidean Signed Distance Fields for on-board MAV planning," in *IEEE International Conference on Intelligent Robots and Systems*, vol. 2017-Septe, 2017, pp. 1366–1373.

[12] V. Usenko, L. Von Stumberg, A. Pangercic, and D. Cremers, "Real-time trajectory replanning for MAVs using uniform B-splines and a 3D circular buffer," in *IEEE International Conference on Intelligent Robots and Systems*, vol. 2017-Septe, 2017, pp. 215–222.

[13] 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.

[14] Z. C. Marton, R. B. Rusu, and M. Beetz, "On fast surface reconstruction methods for large and noisy point clouds," in *2009 IEEE International Conference on Robotics and Automation*, May 2009, pp. 3218–3223.

[15] D. Gingras, T. Lamarche, J. Bedwani, and . Dupuis, "Rough terrain reconstruction for rover motion planning," in *2010 Canadian Conference on Computer and Robot Vision*, May 2010, pp. 191–198.

[16] P. Egger, P. V. K. Borges, G. Catt, A. Pfrunder, R. Siegwart, and R. Dub, "Posemap: Lifelong, multi-environment 3d lidar localization," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2018, pp. 3430–3437.

[17] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[18] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation*, vol. 2, 2000, pp. 995–1001 vol.2.

[19] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robotics Automation Magazine*, vol. 19, no. 4, pp. 72–82, Dec 2012.

[20] P. Soueres and J. . Laumond, "Shortest paths synthesis for a car-like robot," *IEEE Transactions on Automatic Control*, vol. 41, no. 5, pp. 672–688, May 1996.

[21] J. L. Giesbrecht, D. Mackay, J. Collier, and S. Verret, "Path tracking for unmanned ground vehicle navigation," *Technical Memorandum*, no. December, 2005.

[22] R. B. Rusu, "Semantic 3d object maps for everyday manipulation in human living environments," Ph.D. dissertation, Technische Universitaet Muenchen, Germany, 2009.

[23] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 2011.