Creating Navigable Space from Sparse Noisy Map Points

Zheng Chen and Lantao Liu

Abstract—We present a framework for creating navigable space from sparse and noisy map points generated by sparse visual SLAM methods. Our method incrementally seeds and creates local convex regions free of obstacle points along a robot’s trajectory. Then a dense version of point cloud is reconstructed through a map point regulation process where the original noisy map points are first projected onto a series of local convex hull surfaces, after which those points falling inside the convex hulls are culled. The regulated and refined map points allow human users to quickly recognize and abstract the environmental information. We have validated our proposed framework using both a public dataset and a real environmental structure, and our results reveal that the reconstructed navigable free space has small volume loss (error) comparing with the ground truth, and the method is highly efficient, allowing real-time computation and online planning.

I. INTRODUCTION

Simultaneous localization and mapping (SLAM) has been shown as an excellent solution when a robot needs to move in environments that are unknown and GPS is unavailable. We are interested in the problem of reconstructing navigable space from a cloud of 3D map points that are produced from SLAM methods. Particularly, we investigate the map points that are sparse and noisy, and are generated and converted from visual features detected by a vision system (either monocular or stereo vision) carried by a robot. The map points represent obstacles in the environment, however, they are not immediately useful for robot navigation because these visual feature based 3D map points can be very inaccurate for describing obstacle’s position information, especially if we compare them with the point clouds generated from other high-definition ranging sensors such as LIDAR. The objective of this work is to extract navigable space from sparse map points, the result of which can be used for future collision-free navigation and motion planning.

To maximize the constructed free space from the noisy and irregular map points, we opt to use a convex region growing strategy [1] which iteratively looks for a sequence of polyhedra with maximum inscribed ellipsoids using quadratic programming and semidefinite programming. We then develop a framework by incrementally seeding and creating local convex regions along a robot’s trajectory. To build a complete navigable volume (tunnel) for the entire map, we reconstruct a regulated version of point cloud which looks similar to the outputs of high-definition ranging/depth sensors. The regulation is built on a projection process on the generated convex hulls. This is done by projecting the original noisy points around the polyhedra onto the surfaces of those polyhedra, after which those points falling inside the convex hulls are culled. The regulated point cloud also allows human users to quickly recognize and abstract the environmental information.

We have validated our proposed framework using both KITTI dataset and a real environmental structure. Our results reveal that the reconstructed navigable free space has small volume loss (error) comparing with the ground truth, and the method is highly efficient, allowing real-time computation and online planning.

To the best of our knowledge, this is the first time to extract free space from 3D sparse and noisy map points transformed from visual features using point regulation operation.

II. RELATED WORK

Point cloud processing methods have been well studied in past years. Existing point cloud methods can be categorized
as surface reconstruction, volume reconstruction, model fitting, and kernel-based regression frameworks. Specifically, a large set of early work aims to build surfaces from point clouds. For instance, moving least squares (MLS) based methods [2], [3] were developed to reconstruct point clouds output from laser scans; projection-based greedy strategies [4] were used to achieve incremental surface growing and building. Signed distance function [5] and Bayesian method [6] have also been investigated for surface reconstruction. Some other works, like [7], [8], and [9], adopted volume carving mechanism to obtain free space given a set of points. Typically these methods first decompose the space into cells using 3D triangulation techniques, and then the visibility constraints are used to carve out those cells passed by the visibility line. Different from the surface-based and volume-based reconstruction schemes, RANSAC based model fitting methods [10], [11] have been used to capture the spatial structure of the given set of points. Online kernel-based learning methods [12] have also been proposed to implement terrain estimation from the point cloud of a LIDAR scanner.

Related work also includes various mapping approaches as our work utilizes the 3D map points generated from existing mapping methods. Existing map forms for robots navigation include, for example, occupancy grid map [13], 3D OctoMap [14], signed distance map [15], [16], topological map [17], [18], and convex region growing map [1], [19], [20], etc. In this work we are particularly interested in the map points generated from sparse visual features such as those from ORB-SLAM [21] and SVO [22].

However, in our problem where only sparse map points are provided as inputs, existing point cloud processing methods expose limitations. First, the majority of existing methods [2], [3], [4], [5], [6] assume that the points are captured by ranging sensors such as Lidars, sonars or depth cameras, and therefore the points are dense and evenly (uniformly) distributed like a mesh surface. We cannot use those approaches, because we are not able to get well estimated normal and curvature of point set from those techniques. Volume reconstruction methods [7], [8], [9] need to implement 3D triangulation for all the points. This however is not necessary if building a navigable space is the final goal: we want to build a map with only a minimal set of points defining the free space, instead of all of the (possibly noisy) points. In addition, the computation requirement for the 3D triangulation and the post-processing can be prohibitive. Plane fitting methods [10], [11] also fail in our case due to the high ambiguity of points structures.

III. Method Overview

An autonomous robot needs to know clearly the obstacle space and free space before taking actions. We desire the map that describes the environment to include precise 3D information of obstacles just like the map point cloud constructed from a 3D LIDAR sensor. However, this is challenging or even unrealistic due to the sparse and noisy visual features detected. One possible way to “relax” this problem is to build an approximate navigable space with conservative surfaces to “bound” (represent) obstacles. In other words, we aim at obtaining a conservative but safe approximation of a free, navigable space instead of an exact representation of the obstacle surfaces. The input of our work is the 3D map points (point cloud) generated from existing sparse visual odometry or mapping methods.

A. Challenges

Different from the points generated from ranging sensors which capture dense 3D position information with high accuracy, the map points constructed by visual odometry or SLAM methods are much irregular, resulting in great difficulty for accurately describing 3D object information. Specifically, there are a couple of challenges:

- **High variation on the density of map points.** The map points are typically converted from visual features by visual odometry or SLAM methods. In general the visual features are not evenly or uniformly distributed in the field of view, regardless of the feature detectors/descriptors employed. Consequently, the converted 3D map points are distributed unevenly in 3D space and are visually nowhere surface-like. See Fig. 2.

- **Large noise on point’s position.** Ideally, the map points of a smooth surface like a wall should be located on a plane, but in reality the computed map points are rather scattered randomly in a 3D volume around the surface area, leading to great difficulty for reconstructing a surface.
**Misleading information.** Different from conventional data processing methods where a small number of outliers are viewed as noise points, in our context the seeming outliers could be the majority and cannot be simply discarded. Also highly dense batches of points do not necessarily mean the area is valuable/important but perhaps contain much misleading information and thus require extra effort to remove, thin, or regulate them.

Typical existing surface and volume reconstruction methods (e.g., [3], [4], [5], [9]) do not work as they usually focus on triangulation for all the points but do not provide estimation of normal and curvature information from sparse and noisy points.

It is also worth mentioning that, although each input map point may contain multiple pieces of information (for example, in the ORB-SLAM, a map point not only contains the 3D position information for an extracted feature key point, but also records information of the original ORB feature descrtor as well as the view direction from a corresponding keyframe optical center), in this work, we only take advantage of the 3D position information for each map point.

**B. Free Space Extraction and Construction**

Instead of reconstructing the point cloud (obstacles) as in most existing work [3], [4], [10], we are interested in building the free space directly through discarding the “obstacle space”. An additional advantage of doing this is that we immediately obtain the obstacle-free area in which we can model planning modules directly.

Specifically, we opt to grow a convex hull under the spatial constraints of the map points. We adopt to use the Iterative Regional Inflation by Semidefinite (IRIS) method [1]. IRIS is a convex optimization process to find a large and convex region given a set of constraints in 3D space. The method alternates between searching for a set of hyperplanes that separate a convex space from the obstacles using quadratic programming and searching for the maximum ellipsoid which inscribe the convex space previously found using semidefinite programming.

Formally, an ellipsoid is represented as an image of a unit ball:

\[
\mathcal{E} = \{ x = C\tilde{x} + d \mid \| \tilde{x} \|_2 \leq 1 \} 
\]  

(1)

where \( x \) is the 3D points, \( \tilde{x} \) represents the points inside a unit ball. \( C \) and \( d \) are the coefficients of the affine transformation.

The polyhedron (3D polytope) is represented by a set of linear inequalities:

\[
Q = \{ x \mid Ax \leq b \} 
\]  

(2)

where \( Ax \leq b \) denotes a series of constraints and \( A, b \) are a coefficient matrix and a vector, respectively.

The notation of the ellipsoid takes a form of affine transformation, which has a straightforward meaning when we try to maximize the volume of the ellipsoid: the volume is proportional to the determinant \( \log \det C \). Let \( v_k \) be the vertices of the obstacles and \( \zeta \) be the set of obstacles. The problem could be formulated as follows:

\[
\begin{align*}
\text{maximize} & \quad \lambda, b, C, d \\
\text{subject to} & \quad a_j^T v_k \geq b_j, \quad \text{for all points } v_k \in \zeta_j, \\
& \quad \text{for } j = 1, \cdots, N \\
& \quad \text{for } k = 1, \cdots, K \\
& \quad \sup_{\| \tilde{x} \| \leq 1} a_j^T (C\tilde{x} + d) \leq b_j, \quad \forall j = 1, \cdots, N
\end{align*}
\]  

(3)

where \( a_j \) is a vector representing the \( j^{th} \) row of \( A \), \( b_j \) is the \( j^{th} \) element of \( b \), \( N \) is the number of obstacles and \( K \) is the number of vertices on obstacle \( \zeta_j \). The first constraint requires all vertices of obstacles to be on one side of the obtained hyperplanes while the second constraint requires all points on the ellipsoid to be on the other side of those obtained hyperplanes.

**C. Seeding and Building Local Convex Regions**

A hurdle that prevents us from directly utilizing IRIS lies in that, the obstacles in our case are not convex polyhedrons, but a cloud of sparse, noisy, and cluttered map points.

To address the problem, first, we sample points as *seed points* along the robot’s trajectory with a certain interval. Then the evident outlier (noise) map points are filtered and eliminated. An outlier map point can be identified if the map point is far away from both the sampling point on robot’s trajectory and the centroid of a patch of point cloud around that sampling point. Since the map points in 3D space are viewed as obstacles, instead of finding the closest point on obstacles to the ellipsoid as mentioned in the standard IRIS process, we opt to look for the closest point to the ellipsoid among the nearest neighboring points. Therefore, we employ the \( k \)-nearest neighbors (\( k \)-NN) method for each sampling point in the filtered point cloud.

After that, the convex region growing process starts from each seed point and terminates when the ellipsoid touches some points and its growth rate is below some predefined threshold. Because such growing process only occurs inside the free space, the final extracted result is always guaranteed to be obstacle-free (and collision-free if the kinodynamic motion is planned inside this space).

In greater detail, let \( S = \{ s_i \} \) be the set of seed points sampled from the trajectory, and \( P_i \) be the set of nearest neighbors of \( s_i \), where \( i \) is defined as \( i = 1, \cdots, Q \), indexing

![Fig. 3: Each black solid line represents an obstacle boundary. The green region represents free space. The left figure shows an ideal situation. However, if constrained by noisy obstacle points, certain space may be lost as shown on the right.](image-url)
D. Point Cloud Regulation

We are also interested in generating the form of point cloud that is similar to high-definition 3D ranging sensors because a dense and clean form of point cloud can help human quickly recognize and abstract environmental information. Such “cleaned” map point cloud is particularly useful for the human-in-the-loop system. To achieve this, we propose a method that regulates the original noisy map points by projecting all points on the corresponding convex hull surfaces.

The high-level flowchart can be seen in Fig. 5. The input for the whole reconstruction process consists of the seed point set sampled from the trajectory, their related nearest points, and the generated convex hulls using IRIS. For each seed point and its nearest points as well as its associated convex hull, we first make a projection of the nearest points onto the adjacent surfaces of the convex hull. Next, since the convex hulls are pairwise overlapped, the projection of nearest points on hull surfaces may cause some points to fall inside other convex hulls, and thus those points have to be removed. After that, the points will be down-sampled using the Voxel Grid Filter approach. This will make the points uniformly distributed. The down-sampled points will then be triangulated using projection-based incremental triangulation method [4] to form a mesh and by doing so, we create mesh edges between points. After building the mesh, we perform an interpolation and add more points on the edges of the built mesh, resulting a denser point cloud. Finally, we use Moving Least Squares (MLS) [2] to smooth the points, and the output is the reconstructed point cloud. The blocks in green refine the points and can be achieved using existing tools (e.g., using suites from PCL [23]). Here we focus on describing the point regulation components in red blocks, as briefly illustrated in Fig. 5.

1) Points projection: For each convex hull $\tilde{c}_i$, we project the points in $\tilde{P}_i$ on the meshes of $\mathcal{M}_i$ and obtain the projected points $\tilde{P}_i$. Let $\mathcal{P}_{C_i}$ be the points that fall inside other convex hulls in $\tilde{P}_i$. As illustrated in Fig. 7(a), all segment lines connecting the seed point of $\tilde{P}_i$ and the projected point will be inside $\tilde{P}_i$ and have no intersection with any boundary of the convex hull. In our case, if any projected point on a convex hull $A$ falls inside another hull $B$, the segment connecting the seed point of $B$ with the projected point will be inside $B$ and has no intersection with the boundary of $B$. All segment lines connecting the seed point of $B$, that means the projected points is outside the hull $B$. We then remove those points.

An illustration in 2D scenario is shown in Fig. 7. We take two convex hulls: $\tilde{c}_i$ and $\tilde{c}_i+1$. The red dots $s_i$ and $s_{i+1}$ are the two adjacent seed points. The projected points $\tilde{P}_i$ on $\tilde{c}_i$ are those yellow points whereas the projected points $\tilde{P}_{i+1}$ on $\tilde{c}_{i+1}$ are those blue points. The points in $\mathcal{P}_{C_i}$ are noted with $\tilde{p}_i^1$ and $\tilde{p}_i^2$, and the points in $\mathcal{P}_{C_{i+1}}$ are marked with $\tilde{p}_{i+1}^1$, $\tilde{p}_{i+1}^2$ and $\tilde{p}_{i+1}^3$. In Fig. 7(a), all segment lines connecting the seed point $s_i$ and the points in $\mathcal{P}_{C_{i+1}}$ are within $\tilde{c}_i$. While all the segment lines connecting the seed point $s_{i+1}$ and the points in $\mathcal{P}_{C_i}$ are within $\tilde{c}_{i+1}$.

In contrast, in Fig. 7(b) any segment line connecting the
Fig. 6: The point cloud regulation and refinement flowchart. Yellow blocks are the inputs. Red blocks are the critical steps for noisy point regulation. The green blocks denote the refinement steps, and the blue block is the output of the whole process.

Fig. 7: The culling operation is performed for two adjacent convex hulls. We judge whether the points need to be removed by using the visibility lines. A visibility line is the segment connecting a seed point and a projected point. If the visibility segment between a seed point and a projected point is inside a convex hull, this projected point is determined as the point to be culled.

The culling process allows us to harvest those “safe” map points. We can observe that the obtained map points are typically located on the outer surface of the convex hulls, but they might not be satisfactorily smooth and dense. Techniques shown in green blocks of Fig. 6 are then employed to further refine the result.

IV. EXPERIMENT

We validate our proposed method using both KITTI dataset and a real built structure/environment. The sparse map points are generated through running the ORB_SLAM2 [24]. For the real environment experiment, we use TurtleBot3 Waffle which is equipped with Intel RealSense R200 camera carried by the robot for capturing the environment data. We use the monocular mode of ORB_SLAM2 so that only 2D images are used for extracting visual features.

Fig. 8: (a) The raw point cloud obtained directly from ORB_SLAM2; (b) The filtered point cloud using the Statistical Outlier Removal Filter method.

Fig. 9: The created free space consists of a series of overlapping convex hulls. The facets of the convex hulls are represented by blue meshes while the points around the hulls are the nearest neighboring points. We treat these points as obstacles during the convex hull growing process.

A. Experiment with KITTI Dataset

As mentioned in Sect. III outlier points from the original point cloud need to be eliminated first. To do so, the Statistical Outlier Removal Filter tool provided by PCL [23] is adopted. A comparison between before and after filtering can be seen in Fig. 8. The filtered result provides a point cloud input to our framework. The created free space using our proposed method is shown in Fig. 9. In general, the created convex regions can cover the free space inside the point cloud while satisfying the obstacle-free property—no point will be included inside the convex regions.
Based on the constructed free space, we implement the procedure shown in Fig. 6 and obtained the results as shown in Fig. 10. Fig. 10(a) reveals the regulated point cloud after projection on convex hull, from which we can observe that the cloud are cleaner than the original noisy map points. We then further refine the result and obtain a denser and smoother point cloud as shown in Fig. 10(b). This improved point cloud well shapes the free space while keeping the general distribution of the original point cloud (no point appears on the top side as no feature is detected from the sky).

B. Experiment in Real Environment

Although KITTI dataset provides great convenience for conducting simulation, it is extremely difficult to evaluate the performance as the outdoor environment captured in the video is complex and cannot be accessed. To evaluate a set of basic performances, we build an environment with a regular shape whose geometry can be exactly measured and calculated. Such ground truth greatly facilitates comparison and analysis for our reconstruction results. The built environment and its corresponding dimension information are shown in Fig. 11.

To avoid environmental symmetry which could easily provide features of great similarity and result in larger chances of loop closure errors, we design the testing environment in a simple and asymmetric form. Additionally, random pictures (textures) and small objects are placed in the environment to guarantee sufficient features can be detected and used. Our reconstruction result can be seen in Fig. 12.

To evaluate the experiment result, we compare the boundary of the created free space, that is, the boundary of the convex hulls with the real boundary of the environment. The error is defined as the distance between these two versions of boundaries. Since calculation of the difference between two 3D hull-surfaces is not easy, we thus make some approximations. First, we project the vertices of the convex hulls onto the $x-y$ plane. Then we separate the boundary of the environment into 8 segments which form 8 areas as shown in Fig. 13. For each area, we use some cells to discretize it, for example, see the yellow cells in area 1 in Fig. 13. Some of the previously projected vertices will fall into the cells, and for each cell we pick the point that is closest to the boundary of the environment as the boundary point of our created free space. For those cells which do not contain any projected points, we take the middle point of the most inner edge of the cell as the boundary point of free space, and we name those middle points as false points. The places which have false points typically have large reconstruction errors, which will be seen later. Finally, we obtain a set of approximate boundary points of the created free space. Connecting those points will form an approximate boundary line. The distance from an approximate boundary point to the corresponding boundary edge represents the reconstruction error for the corresponding cell. The reconstruction error for each edge could be calculated through the sum of squared differences.
For each boundary a vicinity area is investigated. The areas are labeled with red bold numbers. To discretize the constructed area, a set of 2D grids are used. The yellow grids in area 1 are shown as an example. The red star represents the origin of the map frame (i.e., the robot’s starting position).

Fig. 13: For each boundary a vicinity area is investigated. The areas are labeled with red bold numbers. To discretize the constructed area, a set of 2D grids are used. The yellow grids in area 1 are shown as an example. The red star represents the origin of the map frame (i.e., the robot’s starting position).

Fig. 14: Top view of the created free space and the computed approximate boundary points, shown as dark blue points. The green lines represent the approximate boundaries of the free space.

Fig. 15: Boundary errors are calculated in the respective areas. The areas’ labels and the corners of the environment are marked with numbers and capital letters, respectively.

Fig. 16: (a) Number of approximated boundary points in each area. (b) For each area, the SSD is calculated. While the average squared error is the corresponding SSD error divided by the number of boundary points.

(SSD). The approximate boundary lines in our experiment result is shown in green lines in Fig. 14

To facilitate evaluation of the reconstruction error, we divide the areas into two types: (1) outer areas including areas 1 – 4 and (2) inner areas including areas 5 – 8, as illustrated in Fig. 13. Based on the error metric defined above, the reconstruction error is shown in Fig. 15. In our experiment, we measure the unit proposed by ORB SLAM2 as follows: 1 cm = 0.012 unit. The squared sum of and average boundary errors are given in Fig. 16.

From Fig. 15 we can see that both outer boundary errors and inner boundary errors tend to grow when the robot moves to the corner of the built environment. This is due to the intrinsic limitation of visual SLAM methods where visual feature points are harder to be detected while the camera is rotating, causing the number of feature points to decrease and the estimation error to increase around corners. Comparing between the outer boundary errors and inner boundary errors, we can also observe that the values of the former are larger than the latter, which also could be seen in Fig. 16(b). We can see that both two errors are very small when the robot moves in a straight path along a corridor. Such varying property of errors is consistent with the characteristics of visual SLAM methods which work particularly well for straight line movements but can be less robust for rotations.

The time costs of different parts in our proposed framework are presented in Table 18 where the input point clouds information is given in Table 17. As seen in Fig. 18, the total time for creating the free space for the whole map in our real environment experiment is only around 1.5 seconds whereas the KITTI 05 sequence requires 16.8403 seconds. However, if we examine the time consumed per seed point (a seed is sampled around every one second), they are in the same level, i.e., the times consumed per seed in real environment and with KITTI 05 sequence are 0.0412 second and 0.0723 second, respectively. Such limited time cost completely satisfies the requirement for real-time planning and navigation within finite horizons.
For the part of dense point cloud reconstruction, it is relatively time-consuming as revealed in Fig. 18. However, the regulated and refined dense point cloud is not required for robot’s navigation and planning, instead the convex hulls can be a necessary and critical module which can be easily achieved in real-time computation. The dense point cloud facilitates human recognition and abstraction of the environment, but this component can be executed only periodically or based on the human’s infrequent requests instead of real-time updates.

V. CONCLUSIONS

In this paper, we have presented a framework to create navigable space from sparse and noisy map point cloud generated by sparse visual SLAM methods. Our method first builds local convex regions, from which we further regulate and refine the original noisy and sparse map points to obtain a denser and smoother point cloud that well describes the environment. We have validated our proposed framework using both a public dataset and a real environmental structure, and our proposed method is validated to be robust to highly noisy map points and efficient for real-time planning and navigation.

REFERENCES

[1] R. Deits and R. Tedrake, “Computing large convex regions of obstacle-free space through semidefinite programming,” in Algorithmic foundations of robotics XI. Springer, 2015, pp. 109–124.
[2] C. E. Scheidegger, S. Fleishman, and C. T. Silva, “Triangulating point set surfaces with bounded error,” in Symposium on Geometry Processing. Citeseer, 2005, pp. 63–72.
[3] S. Fleishman, D. Cohen-Or, and C. T. Silva, “Robust moving least-squares fitting with sharp features,” in ACM transactions on graphics (TOG), vol. 24, no. 3. ACM, 2005, pp. 544–552.
[4] 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. IEEE, 2009, pp. 3218–3223.
[5] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, “Surface reconstruction from unorganized points.”
[6] J. R. Diebel, S. Thrun, and M. Brünig, “A bayesian method for probable surface reconstruction and decimation,” ACM Transactions on Graphics (TOG), vol. 25, no. 1, pp. 39–59, 2006.
[7] D. Lovi, N. Birkbeck, D. Cobzas, and M. Jagersand, “Incremental free-space carving for real-time 3d reconstruction,” in Fifth international symposium on 3D data processing visualization and transmission (3DPVT), 2010.
[8] A. Romanoni and M. Matteucci, “Incremental reconstruction of urban environments by edge-points delaunay triangulation,” in 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2015, pp. 4473–4479.
[9] Y. Ling and S. Shen, “Building maps for autonomous navigation using sparse visual slam features,” in 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2017, pp. 1374–1381.
[10] L. Nan and P. Wonka, “Polyfit: Polygonal surface reconstruction from point clouds,” in Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 2353–2361.
[11] H. Isack and Y. Boykov, “Energy-based geometric multi-model fitting,” International journal of computer vision, vol. 97, no. 2, pp. 123–147, 2012.
[12] R. Hadsell, J. A. Bagnell, D. F. Huber, and M. Hebert, “Accurate rough terrain estimation with space-carving kernels,” in Robotics: Science and Systems, vol. 2009, 2009.
[13] A. Elhes, “Using occupancy grids for mobile robot perception and navigation,” Computer, vol. 22, no. 6, pp. 46–57, 1989.
[14] 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.
[15] H. Oleynikova, A. Millane, Z. Taylor, E. Galceran, J. Nieto, and R. Siegwart, “Signed distance fields: A natural representation for both mapping and planning,” in RSS 2016 Workshop: Geometry and Beyond-Representations, Physics, and Scene Understanding for Robotics. University of Michigan, 2016.
[16] H. Oleynikova, Z. Taylor, M. Fehr, J. Nieto, and R. Siegwart, “Voxelblox: Building 3d signed distance fields for planning,” arXiv, pp. arXiv—1611, 2016.
[17] F. Blochliger, M. Fehr, M. Dymszyk, T. Schneider, and R. Siegwart, “Topomap: Topological mapping and navigation based on visual slam maps,” in 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018, pp. 1–9.
[18] H. Oleynikova, Z. Taylor, R. Siegwart, and J. Nieto, “Sparse 3d topological graphs for micro-aerial vehicle planning,” in 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2018, pp. 1–9.
[19] R. Deits and R. Tedrake, “Efficient mixed-integer planning for uavs in cluttered environments,” in 2015 IEEE international conference on robotics and automation (ICRA). IEEE, 2015, pp. 42–49.
[20] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, “Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments,” IEEE Robotics and Automation Letters, vol. 2, no. 3, pp. 1688–1695, 2017.
[21] R. Mur-Artal and J. M. Montiel, and J. D. Tardos, “Orb-slam: a versatile and accurate monocular slam system,” IEEE transactions on robotics, vol. 31, no. 5, pp. 1147–1163, 2015.
[22] C. Forster, M. Pizzoli, and D. Scaramuzza, “Svo: Fast semi-direct monocular visual odometry,” in 2014 IEEE international conference on robotics and automation (ICRA). IEEE, 2014, pp. 15–22.
[23] P. URL, Point Cloud Library. [Accessed Feb 1, 2019], http:// pointclouds.org/.
[24] R. Mur-Artal and J. D. Tardós, “Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras,” IEEE Transactions on Robotics, vol. 33, no. 5, pp. 1255–1262, 2017.