2020 IEEE International Conference on Robotics and Automation (ICRA)
31 May - 31 August, 2020. Paris, France

Voxel Map for Visual SLAM

Manasi Muglikar, Zichao Zhang and Davide Scaramuzza

***Abstract*— In modern visual SLAM systems, it is a standard practice to retrieve potential candidate map points from overlapping keyframes for further feature matching or direct tracking. In this work, we argue that keyframes are not the optimal choice for this task, due to several inherent limitations, such as weak geometric reasoning and poor scalability. We propose a voxel-map representation to efficiently retrieve map points for visual SLAM. In particular, we organize the map points in a regular voxel grid. Visible points from a camera pose are queried by sampling the camera frustum in a raycasting manner, which can be done in constant time using an efficient voxel hashing method. Compared with keyframes, the retrieved points using our method are geometrically guaranteed to fall in the camera field-of-view, and occluded points can be identified and removed to a certain extend. This method also naturally scales up to large scenes and complicated multi-camera configurations. Experimental results show that our voxel map representation is as efficient as a keyframe map with 5 keyframes and provides significantly higher localization accuracy (average 46% improvement in RMSE) on the EuRoC dataset. The proposed voxel-map representation is a general approach to a fundamental functionality in visual SLAM and widely applicable.**

I. INTRODUCTION

Simultaneous Localization and Mapping (SLAM) is fundamental to robotics and plays a pivotal role in various real-world applications, such as augmented/virtual reality and autonomous driving. The past decade has witnessed rapid progress in this field. Today, state-of-the-art SLAM systems, specifically visual-inertial SLAM, execute in real-time on power and memory constrained devices and provide accurate and robust estimate. Despite the remaining challenges in this field [1], SLAM has reached the maturity that enables successful commercial applications (e.g., [2]). Keyframe-based SLAM, among other paradigms such as filter-based method, is arguably the most successful one nowadays. In particular, keyframe-based SLAM relies on the joint nonlinear optimization of keyframes and visible landmarks, namely bundle adjustment (BA) [3], and achieves superior accuracy than filter-based methods [4].

Following seminal work of [5], most state-of-art sparse SLAM systems use parallel threads for tracking (i.e., compute real-time poses for image stream) and BA to alleviate the computational overhead of the nonlinear optimization. The central task of the tracking process, for both direct and feature-based methods, is to find 2D-3D correspondences between the observations in the current image and the map (e.g., 3D points). While different types of maps are used for dense SLAM (e.g., Truncated Signed Distance Field in [6], surfels in [7]), little work has been done exploring alternative map representations for sparse SLAM. Sparse keyframe-based methods use information from nearby-keyframes to associate images to map points. This is a powerful heuristic that is used in many successful systems [5], [8], [9], [10]. Another class of SLAM systems use geometric primitives (e.g., meshes [11] or planes [12], [13])

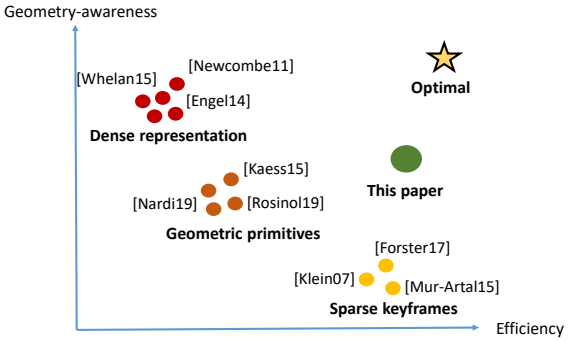


Fig. 1: The optimal SLAM systems should be efficient and have geometrical understanding of the map (denoted by the golden star). Direct methods (red dots) associate each keyframe with a semi-dense depth map. They have more scene information but are not efficient. Sparse keyframe-based SLAM (yellow dots) associate features in the current frame to 3D points from nearby overlapping keyframes. While they are computationally efficient, they do not provide higher level understanding of the geometry of the scene. Other representations using geometric primitives (orange dots) balance geometric information and efficiency, but make assumptions on the scene and do not achieve efficiency as sparse-keyframe methods. This paper proposes a voxel-map for sparse SLAM, that tries to move one step towards optimal map representations for SLAM.

Ideally, the map representation should, (i) be aware of the geometry of the scene and (ii) be efficient in terms of computation time and memory. Fig. 1 shows how different map representations perform on these axes. The ideal representation should allow better geometric reasoning, which brings higher accuracy, but still be at par with keyframe-based methods in terms of efficiency. We compare the effectiveness of sparse keyframe-based map representations along these axes:

Geometry-awareness: In sparse SLAM, using keyframes and their visible points (i.e., covisibility graph) as the map only allows limited geometric reasoning. The co-visibility graph has no notion of occlusion, and it is difficult to determine and filter occluded points, which may cause wrong data association and erroneous estimation. It is ideal that the points retrieved from the map coincide with the field-of-view

The authors are with the Robotics and Perception Group, Dep. of Informatics, University of Zurich , and Dep. of Neuroinformatics, University of Zurich and ETH Zurich, Switzerland— http://rpg.ifi.uzh.ch.
This research was supported by the National Centre of Competence in Research (NCCR) Robotics, through the Swiss National Science Foundation, the SNSF-ERC Starting Grant and Sony R&D Center Europe.

(FoV) of the camera. Unfortunately, there is little geometric guarantee for the points from overlapping keyframes. There may be false positives and missing points.

Efficiency: The effectiveness of keyframes in bundle adjustment comes from the fact that they retain most of the information compared with using all the frames [4]. For a local map of N keyframes, increasing N will improve the robustness in general but will result in a longer query time, despite the fact that we are interested in a spatial area of *fixed size* (i.e., the camera frustum). Moreover, the design of keyframe systems becomes complicated for non-trivial multiple camera systems. For example, setting the images from all cameras as keyframes retain the most information but introduces high redundancy.

Therefore, we argue that using keyframes is not optimal for data association in the tracking process, despite the temptation of using one common representation for different tasks(i.e., BA and point retrieval). In view of the above problems, an ideal map representation for SLAM should be designed for efficient, accurate, geometry-aware points retrieval, rather than simply reusing the keyframes from BA.

In this work, we propose a voxel-map representation that is *scalable* and *geometry-aware*. By representing the environment as voxels, the map coverage can be specified directly, instead of implicitly depending on keyframe parameters. Retrieving points from the map is equivalent to accessing the voxels in the area of interest, and the voxel hashing method proposed in [14] allows *constant query time* for a fixed camera frustum (i.e., fixed number of voxels), regardless of the number of voxels/points in the map. Moreover, since voxels are simply containers for 3D points, modifying the information in the voxel-map (e.g., adding points from a newly added keyframe) is trivial. To query candidate points for data association in SLAM, we propose a raycasting-based method. In particular, we raycast selected pixels from a regular grid in the image to the map and collect the points in the voxels along the rays. Despite its simplicity, this method has two key advantages. First, the points returned by our raycasting method are guaranteed to be all the 3D points in the map that fall in the FoV of the camera, for which keyframe-based methods can only rely on the weak covisibility assumption (see Fig. 6). Second, once we have encountered sufficient 3D points in the nearby voxels along the ray, we can stop checking farther voxels. This gives our method the ability to reason about occlusion to a certain extent. Arguably, this does not provide a complete geometric reasoning as a dense model but is much more efficient. To the best of our knowledge, this is the first work aiming at incorporating a voxel-based representation in sparse SLAM systems. As a general approach, our voxel-map representation can be used as a build block for a wide range of SLAM systems.

The rest of the paper is structured as follows. Section II introduces the voxel map representation for SLAM and describes the raycasting-based query method using the proposed voxel-hashing based map. Section III demonstrates how the proposed method can be integrated into a modern SLAM pipeline. The evaluation of the proposed map representation and the comparison with standard keyframes are presented in Section IV. We then conclude the paper with some discussion in Section V.

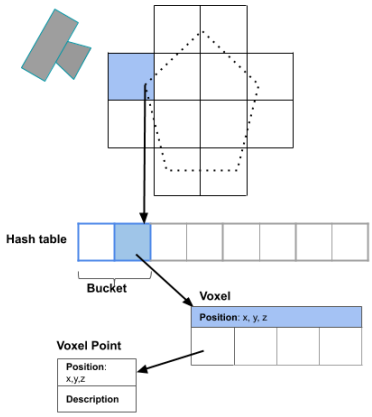


Fig. 2: Voxel hashing data structure. The map is stored as a hash table. The hash function maps the integer world coordinates to hash buckets.

II. Voxel Hashing for SLAM

In this section, we explain the data structure used to represent the voxel-map and how it can be integrated within a SLAM pipeline.

A. Voxel hashing data structure

The proposed voxel-map representation is based on the work of Nießner et al. [14]. It stores the map as a hash table as shown in Fig. 2. In particular, the world is comprised of voxels. Each allocated voxel stores its position in the world coordinates, and a list of voxel points (i.e., actual 3D points of interest, such as 3D landmarks in SLAM) that are within this voxel. As for the voxel points, each of them has a position in the world coordinate and also a description. This description is used for 2D-3D data association in the tracking process (i.e., frame-to-map alignment). For feature-based methods, the description of a point is its feature descriptor, whereas for direct methods, the description is the image patch around the location where the feature is extracted.

We use an efficient C++ implementation of hash table to manage the allocation and retrieval of voxels. Each entry in the hash table contains a pointer to an allocated voxel. The allocated voxels are accessed from the hash table using a hashing function of the voxels' world coordinates. Specifically, the hash value $H(\cdot)$ is computed as [14], [15]:

$$H(x,y,z) = ((\lfloor x \rceil \cdot p_1) \otimes (\lfloor y \rceil \cdot p_2) \otimes (\lfloor z \rceil \cdot p_3)) \; mod \; n, \quad (1)$$

where p_1, p_2 and p_3 are large prime numbers, $\lfloor \rceil$ the rounding operation, $\otimes$ the bit-wise XOR operator, mod the modulo operator, n the hash table size

The goal of voxel hashing is to manage 3D points efficiently. Combining the voxel representation and the hashing method, we can get the points inside a given region (e.g., the camera viewing frustum) in constant time, regardless of

the map size. Alternatives, such as raw point clouds and keyframes, do not scale well as the map size increases. For unstructured point clouds, each point needs to be checked individually to determine whether it falls in the area of interest. Using overlapping keyframes is often a good heuristic, but one still needs to exhaustively check the points in these keyframes, since there is no guarantee that an arbitrary point in an overlapping keyframe will fall in the viewing frustum of the current frame. In other words, keyframes and voxels can be both viewed as proxies of the underlying 3D points of interest, but using hash values based on the voxel positions enable us to directly get points in a precisely defined 3D area of interest, whereas overlapping keyframes can only rely on the weak covisibility assumption. With the voxel-map, several technical details need to be taken care of:

Voxel size: An appropriate voxel size has to be chosen for efficient performance. In the extreme cases, if one point occupies a tiny voxel or all the points live in one voxel, the voxel map representation is no different from raw point clouds.

Resolving collisions: Due to the nature of the hash function, collisions can occur if multiple voxels get mapped to the same hash value. To handle these collisions, we divide the hash table into buckets. Each bucket corresponds to a unique hash value. These buckets store the hash entries as a list. In the event of a collision, we accommodate the new voxel by adding the pointer to the corresponding bucket list. With a reasonable selection of hash table size and bucket size, the collisions can be kept minimum.

B. SLAM map management with voxels

In general, a map in SLAM stores 3D geometric objects, such as points and lines, against which a new frame can localize. A map is updated over time (e.g., add/delete points, update the information of existing points), and should support efficient query in the tracking process (e.g., what are the possibly matched points in a newly coming image?). Below, we describe the corresponding functionalities in our voxel-map. Note that we do not discard keyframes completely, as keyframe-based BA is still necessary for optimizing the map. Our voxel-map is instead a more efficient organization of the 3D points to facilitate data association.

Insert point: We find the target bucket in the hash table using the hash function (1) on the world coordinates of the point. We then iterate over the hash entries in the bucket. If a voxel exists in the space of the point, we add the point to the voxel. If the position of the point is already occupied (i.e., equality up to a certain precision), we update the description of the point with the newly added one. If such a voxel does not exist, a new voxel is created and added as a hash entry to the bucket; the point is then added to the newly created voxel.

Delete point: Deletion is performed similar to insertion. We first calculate the hash value for the point position and then iterate over the hash entries in the target bucket till we find the voxel that fits the point position. The point is then deleted from the voxel.

Query map: To query 3D points at a given location, we first convert the world coordinates of the query location to integer world coordinates then compute the hash value. If there exists a bucket corresponding to this hash value, we iterate over all the hash entries in the bucket till we find the voxel that fits the query location. If the hash entry exists, we return the pointer to the allocated voxel, which contains exactly the 3D points around the query location. We will describe next how this strategy can be used to query possible visible points from a given pose.

C. Point query with raycasting

A necessary function in the tracking process in SLAM is to query the points that is possibly visible in the current frame. This is essentially the same as getting all the points that fall in the camera FoV (up to a cut-off distance). Since our hashing function is based on the voxel position, we directly sample the frustum in a raycasting manner. In particular:

- **Image plane sampling and raycasting**: We first sample pixels from a regular grid on the image plane. Then we backproject these sampled pixels to bearing vectors in 3D space, resulting in r rays $\{R_i\}_{i=1}^{r}$. These rays essentially samples the camera FoV. Note that the rays are expressed in the camera frame.
- **Ray sampling**: For each ray R_i, We sample s points $\{{}^{c}S_j^i\}_{j=1}^{s}$ from a depth range of D_{min} to D_{max}. These points are again expressed in the camera frame c, which can be precomputed offline.
- **Points query**: In the context of SLAM, at the query time, we usually have a prior of the current camera pose (e.g., previous camera pose or IMU propagation). With this prior, we can transform the aforementioned sample points to the world frame ${}^{w}S_j^i$ ($i = [1, r],\ j = [1, s]$), and query the voxels at these sample points using the hashing method in Section II.

The first two steps (shown in Fig. 3 left) essentially discretize the camera frustum in the given distance range. Since the discretization is done in the camera frame, it only needs to be computed *once* (possibly offline). The time complexity of our approach is $\mathcal{O}(r \cdot s)$. Since the r and s are independent of the map size, the query time remains constant, even as the map grows. In contrast, the map query time for keyframe-based map is $\mathcal{O}(k)$, where k is the number of keyframes and increases with the map size.

In addition, we would like to emphasize that the sampling order for each ray is from the closest voxel to the most distant. In this way, when querying the map along each ray, we only return the first occurring voxel, thus avoiding the occluded voxels along the same ray, as illustrated in Fig. 3.

III. Case Study: SVO with voxel map

To demonstrate the practical value of our map representation, we adapt a state-of-the-art keyframe-based sparse SLAM pipeline SVO [9] to use the voxel-hashing based map and raycasting points query. It is worth-noting that the proposed method is a general building block for SLAM, and widely applicable.

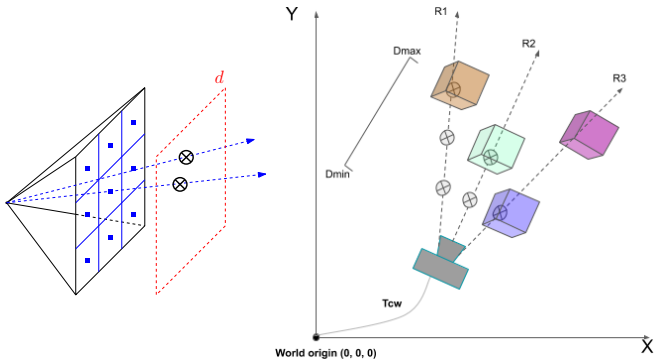


Fig. 3: **Left**: Sampled pixels (blue) on the image plane are backprojected to rays in 3D, which are then sampled at a discrete set of distances (only two rays and one distance plane, d, is shown here to avoid cluttering). **Right**: Points query with raycasting. Here the voxels are colored only for the ease of illustration.

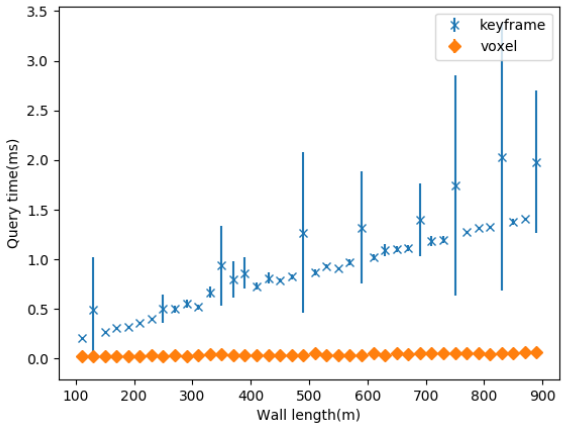


Fig. 4: Comparison of time taken for querying the map as the map size increases.

SVO is a hybrid pipeline that combines the advantages of direct and feature-based methods. In particular, it first aligns the new image with the previous image by minimizing the photometric error over a sparse set of patches with known depths. This gives a good prior about the pose of the new frame. With the given prior, the pipeline finds the keyframes that overlap with the new frame. The overlapping keyframes are found by projecting selected points from the current frame to the keyframes in the local map (ordered by distance to current keyframe) until a set of M overlapping keyframes are found. Since in most translation motion cases the closest M keyframes with overlap are the latest M keyframes, the average query time depends only on the value of M and not on the map size. The pipeline further searches for matches in the new image for the points from these keyframes by Lucas-Kanade tracking [16]. Once the correspondences are established, the pose is estimated through a motion-only bundle adjustment. The pipeline also has a separate mapping thread, which uses a robust Bayesian filter [17] for depth estimation.

To integrate our proposed voxel-hashing map with SVO, we make the following adaptations:

Map query for motion estimation: The pose estimate from sparse image alignment gives a prior to find potential correspondences. Instead of checking overlapping keyframes, we directly sample the camera frustum as described in Section II-C. Our raycasting-based map query return a set of voxels that are visible, without occluded voxels, from the camera frustum. Here we assume points from the same voxel do not occlude each other, which is mostly true considering the sparsity of the map. Then the points within these voxels are tracked as in the original pipeline.

Map management: If the depth uncertainty for a point reduce to a certain threshold in the depth filter, a new 3D point is initialized at the estimated depth. This point is then inserted in the voxel map as described in Section II and used for camera pose tracking afterwards. Moreover, during the tracking stage, certain points are labeled as outliers (e.g., in the motion-only bundle adjustment). These points are removed from the voxel map as described in Section II.

IV. EXPERIMENTAL EVALUATION

We organize our experimental results in two parts: simulation and real-world experiments. In simulation, we compared the performance of our voxel-map and raycasting map query with a naive-keyframe-based method, in terms of map query time and occlusion handling. We also performed experiments on EuRoC [18] dataset, where we focused on the pose estimation accuracy of overlapping keyframe map, as described in Section III and our proposed approach.

A. Simulation

Map query time: The goal of this experiment is to show that our method scales better than keyframes as the map size grows. To this purpose, we simulated a map that consists of a straight wall. We then queried the map (i.e., get visible 3D points) at 10 locations along a line parallel to the wall. For different map size, we increased the length of the wall from 100m to 900m. To ensure same density of map points, map points increased from 1000 to 9000. We established two different map representations:

Naive-Keyframe: We sampled keyframes evenly on the wall, so that each point belongs to a unique keyframe. The maximum number of points in each keyframe was fixed to 100. Therefore, as the length of the wall increases, the number of keyframes in the map also increases. This is to mimic an exploration scenario, where the map keeps expanding. To query the visible points at a given pose, we iterated over the points in the keyframes, and once there is one point visible from the query pose, we considered this keyframe overlapping with the query pose and continued to the next keyframe. The query time was the total time of checking all the keyframes.

Voxel-hashing: We allocated sufficient voxels to hold all the map points. The voxel grid size was fixed to 2m. At query time, we used the raycasting based method described in Section II-C to return a list of visible points.

We compared the map query time for the above representations as the map sized varied. Each experiment was repeated

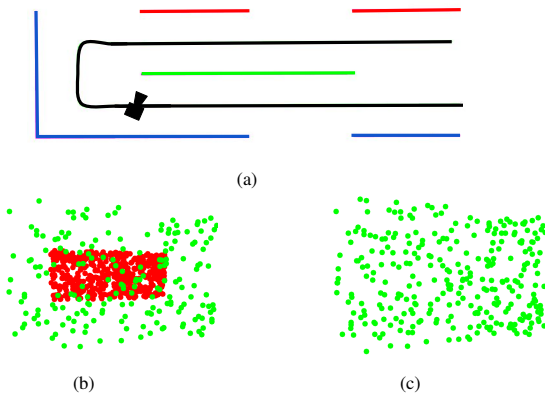
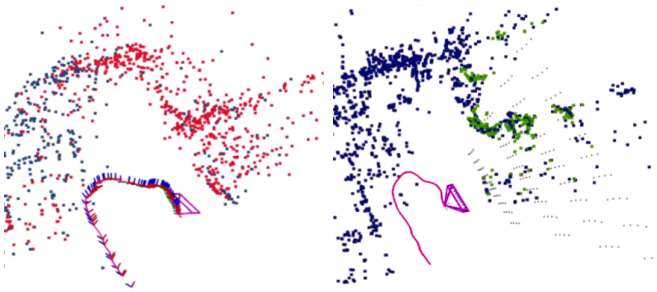


Fig. 5: The top view of the simulation environment to test map query in the presence of an occlusion 5a. The map consists of a long U-shaped corridor. The walls in the map are colored by distance from the camera with red being far away , green closer and blue outside the FoV. The camera is moving along a trajectory (black) while changing viewing direction. Fig. 5b shows map query using keyframe map. Fig. 5c shows map query using voxel map



(a) Map query using keyframes. (b) Map query using voxel map.

Fig. 6: Results of querying keyframe-map and voxel-map at the same pose (purple pyramid) in a SLAM pipeline. Blue (both): all points in the map; Red: visible points queried from overlapping keyframes; Green: visible points queried from the proposed voxel-map; Gray: sample points in the raycasting-based query (Section II-C).

5 times. The results, plotted in Fig. 4, show that for a smaller map size, all methods have similar query time. However, as the map size increased, the query time for the keyframe-based method increased almost linearly; in contrast, the query time using the proposed method remains constant. This makes our method particularly useful to manage map points at a large scale.

Geometric awareness: In this experiment, we were interested in validating the occlusion handling capability of our method. We simulated a corridor-like environment and a camera trajectory through the corridor, as shown in Fig. 5a. The naive-keyframe-based and voxel-map were generated in a similar manner as in the previous simulation. We then queried the map for visible points from the poses along the camera trajectory at the point of turn(as shown in Fig. 5a). The visible points were then projected into the image plane for visualization, the color of the points indicates distance from camera (red being farther away and green being closer). The camera was looking at two planes of points at different distances. Ideally, the points on the plane at the farther distance(i.e., red points) should be occluded by the nearer ones (i.e., green points). While the naive-keyframe query had no notion of occlusion (Fig. 5b); our method, thanks to the raycasting query scheme, was able to recognize the farther points along the same ray as occluded (Fig. 5c).

B. Real-world experiments

We ran monocular version of SVO in two configurations on EuRoC: 1) the original pipeline with an increase in the number of keyframes in map from 5 to 30 ; 2) the adapted pipeline described in Section III, which stores all the landmarks in the map, since this has no overhead in saving candidates in the map. Note that we omitted the results on *V1_03* and *V2_03*, as the aggressiveness in these two sequences made both pipelines (vision-only) unstable so that no meaningful comparison could be made. The Root Mean Squared Error (RMSE) on the tested sequences were computed using the evaluation protocol (SIM3 alignment with all the frames) in [19]. The results are shown in Table. I. We also computed the time required for the entire pipeline per frame of the dataset. The results are shown in Table II. It can be seen from Table I that, as the map size increases from 5 to 30 keyframes, the RMSE reduces overall. The results here might vary from the reported values from [9] due to different parameters for the experiment. Table II shows that with an increase in the map size, the computation time increases. This shows the efficiency-accuracy trade-off in keyframe map in terms of computation time and estimation error. In contrast, the voxel map is able to get better accuracy for lower computation time. It does not always outperform all the keyframe sequences because the performance of the voxel map depends on the environment and the voxel size. In the case of *MH_04*, the voxel map did not have a consistent scale across the trajectory, which caused higher RMSE. We emphasize that our method performs similar to the lowest keyframe map size (5 keyframe map i.e., KF5) in terms of computation time, while achieving a higher accuracy. For example, we achieve an improvement ranging from 3% (on *MH_05*) to 80% (on *MH_02*), with an average of 46.2%, as compared to KF5, on the EuRoC sequences. The color of the voxel map indicates the performance with respect to KF5, blue indicating the improvement of the voxel map over KF5 and red indicating the worst performance. We also show the qualitative results in *MH_01* in Fig. 6, where we can see that the points returned by voxel map were much more consistent with the FoV of the camera.

We evaluated the effect of voxel size on the performance and computation time. We ran the same pipeline of voxel-based map with SVO on the dataset *MH_01*. We increased the voxel size gradually from 0.5m (extreme scenario where a voxel contains a very small number of points.) to 20m (extreme scenario where a voxels contain all the map points.) and calculated the inlier ratio, RMSE and average total time required by the pipeline per frame. The inlier ratio is computed as the ratio of successfully reprojected points and the total map points returned by the query. Therefore, higher inlier ratio indicates better map points were retrieved by the query. The results in Table III show that the voxel size

TABLE I: RMSE(m) original error on tested sequences from the EuRoC datasets. In bold is the best value in the column and the underlined is the second best value. For the voxel map, the blue color indicates it performed better than KF5, red indicates the worst performance.

Algorithm	MH_01	MH_02	MH_03	MH_04	MH_05	V1_01	V1_02	V2_01	V2_02
Ours	0.120	**0.083**	0.856	2.575	0.902	**0.266**	**0.686**	0.336	0.825
KF5	0.255	0.432	1.968	**1.315**	0.930	0.706	1.130	0.709	1.060
KF10	**0.085**	0.474	0.765	1.343	1.081	0.516	0.974	0.356	**0.711**
KF15	0.094	0.167	0.852	1.393	0.989	0.647	0.876	0.281	0.849
KF20	0.092	0.155	0.712	1.552	0.908	0.553	0.959	**0.253**	0.842
KF25	0.117	0.251	0.726	1.829	0.895	0.558	0.996	0.307	0.867
KF30	0.144	0.367	**0.671**	2.042	**0.791**	0.478	0.949	0.259	0.824

TABLE II: Average total time (ms) for each frame. In bold is the best value in the column and underlined is the second best value.

Algorithm	MH_01	MH_02	MH_03	MH_04	MH_05	V1_01	V1_02	V2_01	V2_02
Ours	4.818	**4.539**	**5.309**	5.208	5.042	4.096	5.655	3.850	5.358
KF5	**4.642**	4.690	5.459	**5.071**	**4.753**	**3.686**	**5.244**	**3.349**	**4.603**
KF10	5.732	5.882	6.429	5.708	5.980	4.418	6.044	3.968	5.401
KF15	6.266	6.139	7.004	6.545	6.289	4.722	6.167	4.234	5.780
KF20	6.950	6.766	7.995	7.035	6.290	4.968	6.560	4.551	6.065
KF25	6.976	6.957	8.049	6.390	6.564	5.232	6.811	4.784	6.502
KF30	7.229	7.477	8.534	6.560	6.656	5.435	7.202	5.054	6.832

TABLE III: Effect of the voxel size on RMSE of MH_01 sequence. Bold values represent the lowest values in the columns and underlined are the second best values.

Voxel size(m)	Time(ms)	Inlier ratio	RMSE(m)
0.5	9.286	0.378	0.778
5	5.618	**0.822**	0.278
10	5.767	0.762	**0.217**
15	**5.378**	0.721	0.225
20	5.551	0.715	0.499

affects the estimation error as can be evident from the inlier ratio and RMSE. For smaller voxel sizes, the voxel map acts as point cloud representation; however, due to discretization of the space, we miss many map points resulting in a lower inlier ratio and a higher RMSE. The time is also significantly higher as we have to search for many voxels along the rays from the frustrum, which increases the query time. For extremely large voxel sizes, the inlier ratio is also low as we include some points that do not fall in the FoV of camera; while this affects the inlier ratio, the effect on RMSE is acceptable. With this experiment, we show that to get better estimation performance, we need to choose an appropriate voxel size.

C. Discussion

Using the voxel-map brings better geometric reasoning than keyframes, as shown in Section IV-A. Moreover, it is possible to further generate more complicated/useful geometric representations from a regular voxel grid (e.g., distance field for planning [20]), which opens the door to better geometric reasoning for sparse SLAM. Although the focus of this work is not about accuracy, the scalability of our method could benefit the accuracy of SLAM pipelines as well. In general, being able to maintain and query a larger map brings better accuracy for SLAM pipelines. In practice, however, this is usually limited by the computational power for real-time applications. Therefore, we believe our method, due to the constant query time (regardless of map size), is a valuable tool for accurate SLAM pipelines, as shown in Section IV-B. We believe that this approach is a step towards the direction of using maximally efficient map representations (in terms of accuracy, efficiency and geometry awareness) for SLAM.

V. Conclusions and Future Work

In this paper, we proposed a scalable and geometry-aware voxel-map for sparse SLAM, aiming to replace keyframes for data association in the tracking process. The map is organized in voxels, and each voxel can be accessed in constant time using a hashing function on its location. Using the voxel-hashing method, visible points from a camera pose can be efficiently queried by sampling the camera frustum in constant time, which makes the proposed method scale well with large scenes. Moreover, by sampling the frustum in a raycasting fashion, we were able to handle occlusion, which is not possible using keyframes. We validated the advantages of the proposed method over keyframes using simulation as well as on real-world data with a modern visual SLAM pipeline.

As future work, we would like to explore the use of the voxel-map in non-trivial multi-camera configurations, where the map management with keyframes becomes complicated. How to efficiently generate high-level geometric structure, such as meshes and dense surface, from the voxel map is also of interest for robotic applications such as motion planning.

References

[1] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. D. Reid, and J. J. Leonard, "Past, present, and future of simultaneous

localization and mapping: Toward the robust-perception age," *IEEE Trans. Robot.*, vol. 32, no. 6, pp. 1309–1332, 2016.
[2] "Oculus Quest," https://www.oculus.com/quest/.
[3] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon, "Bundle adjustment – a modern synthesis," in *Vision Algorithms: Theory and Practice*, ser. LNCS, W. Triggs, A. Zisserman, and R. Szeliski, Eds., vol. 1883. Springer Verlag, 2000, pp. 298–372.
[4] H. Strasdat, J. Montiel, and A. Davison, "Real-time monocular SLAM: Why filter?" in *IEEE Int. Conf. Robot. Autom. (ICRA)*, 2010.
[5] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *IEEE ACM Int. Sym. Mixed and Augmented Reality (ISMAR)*, Nara, Japan, Nov. 2007, pp. 225–234.
[6] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, "KinectFusion: Real-time dense surface mapping and tracking," in *IEEE ACM Int. Sym. Mixed and Augmented Reality (ISMAR)*, Oct. 2011, pp. 127–136.
[7] T. Whelan, S. Leutenegger, R. F. Salas-Moreno, B. Glocker, and A. J. Davison, "ElasticFusion: Dense SLAM without a pose graph," in *Robotics: Science and Systems (RSS)*, 2015.
[8] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM: a versatile and accurate monocular SLAM system," *IEEE Trans. Robot.*, vol. 31, no. 5, pp. 1147–1163, 2015.
[9] C. Forster, Z. Zhang, M. Gassner, M. Werlberger, and D. Scaramuzza, "SVO: Semidirect visual odometry for monocular and multicamera systems," *IEEE Trans. Robot.*, vol. 33, no. 2, pp. 249–265, 2017.
[10] J. Engel, V. Koltun, and D. Cremers, "Direct Sparse Odometry," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 3, pp. 611–625, Mar. 2018.
[11] A. Rosinol, T. Sattler, M. Pollefeys, and L. Carlone, "Incremental visual-inertial 3d mesh generation with structural regularities," *IEEE Int. Conf. Robot. Autom. (ICRA)*, 2019.
[12] M. Kaess, "Simultaneous localization and mapping with infinite planes," pp. 4605–4611, 2015.
[13] F. Nardi, B. D. Corte, and G. Grisetti, "Unified representation and registration of heterogeneous sets of geometric primitives,," *IEEE Robot. Autom. Lett.*, vol. 4, pp. 625–632, 2019.
[14] M. Niessner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-time 3d reconstruction at scale using voxel hashing," *ACM Trans. Graph.*, 2013.
[15] M. Teschner, B. Heidelberger, M. Mueller, D. Pomeranets, and M. Gross, "Optimized spatial hashing for collision detection of deformable objects," in *Proc.of Vision, Modeling, Visualization VMV03*, 2003.
[16] S. Baker and I. Matthews, "Lucas-kanade 20 years on: A unifying framework," *Int. J. Comput. Vis.*, vol. 56, no. 3, pp. 221–255, 2004.
[17] G. Vogiatzis and C. Hernández, "Video-based, real-time multi view stereo," *Image Vis. Comput.*, vol. 29, no. 7, pp. 434–441, 2011.
[18] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The EuRoC micro aerial vehicle datasets," *Int. J. Robot. Research*, vol. 35, pp. 1157–1163, 2015.
[19] Z. Zhang and D. Scaramuzza, "A tutorial on quantitative trajectory evaluation for visual(-inertial) odometry," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2018.
[20] 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/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2017.