*Preprint: submitted to 3DV 2017

3D Line Segments Extraction from Semi-dense SLAM

Shida He Xuebin Qin Zichen Zhang Martin Jagersand
University of Alberta

Abstract

Despite the development of Simultaneous Localization and Mapping (SLAM), there lacks efficient methods for representing and processing their large scale point clouds. In this paper, we propose to simplify the point clouds generated by the semi-dense SLAM using three-dimensional (3D) line segments. Specifically, we present a novel approach for 3D line segments extraction. This approach reduces a 3D line segments fitting problem into two two-dimensional (2D) line segments fitting problems, which takes advantage of both image edge segments and depth maps. We first detect edge segments, which are one-pixel-width pixel chains, from keyframes. We then search 3D line segments of each keyframe along their detected edge pixel chains by minimizing the fitting error on both image plane and depth plane. By incrementally clustering the detected line segments, we show that the resulting 3D representation for the scene achieves a good balance between compactness and completeness.

1. Introduction

Extracting geometric information from a stream of images is an important yet challenging topic in computer vision community. The 3D knowledge of the scene is essential for a variety of applications, such as photogrammetry, robot navigation, augmented reality, etc. With the development of Structure from Motion (SFM) and Multi-View Stereo (MVS), one can easily reconstruct a 3D point cloud from a set of images or a video stream. Alternatively, 3D point cloud of a scene can also be obtained in real-time using a monocular Simultaneous Localization and Mapping (SLAM) system.

However, there are several limitations in representing a scene using 3D point clouds. First, points in a point cloud are usually stored independently and present no structural relationships. Second, point clouds require large storage space. However, they are redundant in terms of representing the geometric structure of a scene. For example, it only takes two 3D points to define a linear structure but there are usually hundreds or thousands of 3D points along the structure in a semi-dense or dense point cloud. It severely reduces the efficiency of post-processing and analysis on point clouds.

On the contrary, line segments preserve the structural information of a scene efficiently. Several line segments based 3D reconstruction algorithm have been proposed [6, 15]. These methods rely on efficient line segment detection and inter-keyframe matching, which are difficult for certain scenes. However, with the dense or semi-dense point clouds available, one can estimate 3D line segments without explicitly matching and triangulation.

In this paper, we develop a novel method to extract 3D line segments from semi-dense point clouds. Those detected line segments present structural relations among points and prove to be a highly efficient form of 3D representation. We propose a novel edge aided 3D line segment fitting algorithm. We first detect edge segments from keyframes using Edge Drawing [18]. Then 3D line segments are incrementally fitted along edge segments by minimizing the fitting error on both the image and depth plane. Our method produces accurate 3D line segments with few outliers.

The rest of this paper is organized as follows. In Section 2, we review some preliminary and related works about 3D line segment reconstruction. In Section 3, we describe our proposed algorithm in details. We present our experimental results in Section 4 and conclude in Section 5.

2. Related Works

Accurate sparse 3D point cloud of a scene can be reconstructed by SFM methods [20, 10] from a sequence of images. Feature-based SFM methods extract features from images and match them across different images. Given the correspondences of these features, their 3D locations and camera poses are computed by bundle adjustment. MVS algorithms [5, 4] are usually employed to densify the sparse point clouds. Although the resulting dense point cloud is more informative and complete, MVS can take hours or even days to reconstruct a relatively large scene on a regular desktop computer.

If fast performance is desired, a monocular SLAM system can be used to produce 3D point cloud of the scene

in real time. Similar to incremental SFM methods, given a stream of video frames taken by a moving camera, SLAM systems can compute the camera poses and the environment map at the same time. Although SLAM systems are usually more concerned with the localization accuracy, their resulting maps are very informative as well. By mapping and maintaining keyframes, SLAM can achieve great performance in terms of speed and memory usage.

Generally, SLAM systems output point clouds as their mapping result. According to the density of point clouds, SLAM systems can be categorized into three classes: sparse, semi-dense and dense. Sparse SLAM systems like PTAM [8] and ORB-SLAM [11] detect features and match them across multiple frames. Dense SLAM systems, e.g. DTAM [14], use the raw pixels instead of extracted features. By estimating depth for all the pixels, it can produce smooth and dense reconstruction of the scene. Semi-dense SLAM systems, e.g. LSD-SLAM [3] and DSO [2] achieve great efficiency by using only part of the pixels. The point clouds produced by these methods are not as dense as those from DTAM but are generally denser than those from feature based SLAM methods.

In this paper, we focus on extracting 3D line segments from keyframe-based semi-dense SLAM systems. It is different to other methods that also attempt to reconstruct 3D line segments, e.g. Line3D++[6]. Given the result from SFM and detected line segments on images, Line3D++ construct 3D line hypotheses using weak epipolar constraint. After filtering out outliers based on mutual support, 3D hypotheses are clustered based on their similarity, producing accurate and consistent result. Line3D++ is highly efficient due to the fact that it relies on simple geometric constraints as well as using efficient graph based segment clustering algorithm. Although it performs well in outdoor urban environment, it tends to produce outliers in less structured environment, such as typical indoor environment. In contrast, our method takes semi-dense point clouds as input and is able to produce accurate 3D line segments even for indoor environment.

Line segments are introduced in some recent SLAM systems to improve the robustness and accuracy. Pumarola *et al*. [16] detect the line segments by LSD [19] and match them across different images using Line Band Descriptor (LBD) [22]. By using both the correspondences from line segments and feature points, they achieve better performance compared to using only feature points. Similarly, Yang and Scherer [21] improve the performance of direct Visual Odometry (VO) by detecting and matching line segments using LSD and LBD. They operate on pixels instead of detected features or lines.The matched line segments are used to regularize the depth of pixels on line segments. Their 3D line regularization process is quite similar to our method as they also transform the 3D line fitting problem into a 2D line fitting problem. The key difference is that we try to incorporate depth information in the line detection process, whereas they use the detected 2D line segments from LSD directly.

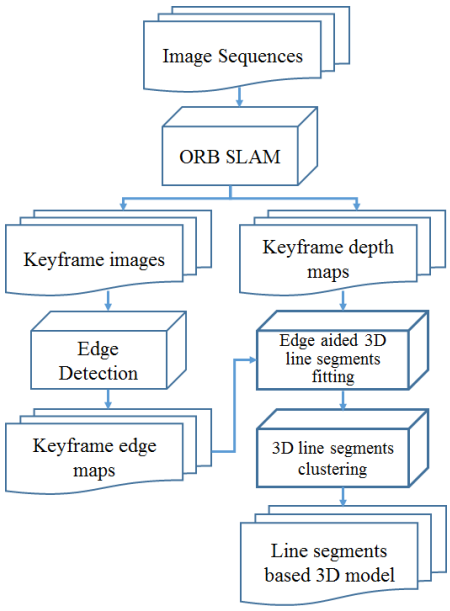

Figure 1. Workflow of our method. The input of our method is a video (image sequence) which is captured by a calibrated moving camera. The output is a line segment based 3D model of the scene.

3. Method

In order to extract 3D line segments from semi-dense point cloud, our method mainly performs the following steps on each new keyframe (shown in Figure 1): (1) Compute semi-dense depth map; (2) Edge aided 3D line segment fitting; (3) 3D line segment clustering and filtering.

3.1. Key frames and depth maps generation

In this paper, our implementation is based on ORB-SLAM [11] with semi-dense module [12]. We focus on extracting 3D line segments from the semi-dense depth map of keyframes. ORB-SLAM is a feature based SLAM system which takes the image sequence from a moving camera and computes the camera poses in real time. By applying advanced keyframe management, powerful feature descriptor, local and global bundle adjustment and loop closure detection, it can robustly track the camera pose and map the environment. Mur-Artal and Tards [12] present a semi-dense module which is able to compute a semi-dense depth map for each keyframe.

Here we briefly describe the process of generating semi-dense depth maps introduced in [12]. By performing epipolar search on a pair of keyframes based on image intensity

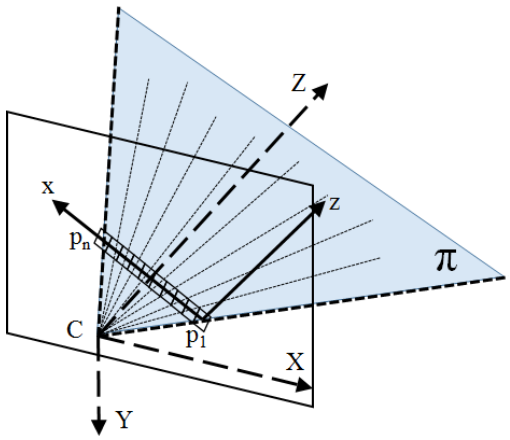

Figure 2. Line segments fitting related coordinates. $\mathbf{C}-\mathbf{XYZ}$ is the camera coordinates. The X-axis and Y-axis are parallel to the image coordinates. The Z-axis is the depth direction. $\{p_1...p_n\}$ represent a detected 2D image line segment. $\mathbf{C}$, p_1 and p_n determine the plane π. Line segment pixels $\{p_1...p_n\}$ and their corresponding real-world points are all located on the same plane π. The x-axis is defined by vector $\overrightarrow{p_1 p_n}$ while z-axis is parallel to the Z-axis.

and gradient, an inverse depth hypothesis is produced for each pixel. An uncertainty of this inverse depth hypothesis is computed and kept throughout the process. Multiple hypotheses are generated for a single pixel by searching on different neighboring keyframes. These hypotheses are fused and the result is kept as the inverse depth of the pixel. The inverse depth is checked and smoothed with respect to the neighboring pixels on the same keyframe, and those from other keyframes. Finally, the inverse depth map is filtered based on the final uncertainty of the pixels. In our experiment, we filter out inverse depth of a pixel if its standard deviation $\sigma > 0.02$. This threshold is empirically chosen to give a balance between accuracy and completeness and it is fixed throughout our experiments.

In principle, our method is general enough to be applied to other keyframe-based semi-dense or dense SLAM systems (e.g. LSD-SLAM [3], DSO [2]).

3.2. Edge aided 3D line segment fitting

Direct 3D line fitting from point clouds are difficult and time consuming. In this paper, we propose to use 2D image information on keyframes to help 3D line segment fitting from semi-dense point clouds.

We first extract edge segments from keyframes using Edge Drawing (ED) [18]. ED is a linear time edge detector which can produce a set of accurate, contiguous and clean edge segments represented by one-pixel-width pixel chains. Then we project the semi-dense depth map to the corresponding keyframe. Now the detected edge

Algorithm 1 Edge aided 3D line segment fitting

Input: A set of Edge Segments on the k-th keyframe: $\mathrm{ES_k}$=$\{\mathrm{es_1}, \mathrm{es_2}, ..., \mathrm{es_m}\}$, $\mathrm{es_i}$ =$\{\mathrm{p_1^i}, \mathrm{p_2^i}, ..., \mathrm{p_t^i}\}$, p_j^i denotes the j-th pixel of edge segment $\mathrm{es_i}$

Output: Fitted 3D line segments $\mathrm{LS_k}$

for $i = 0; i < n; i$++ **do**
  fit_check = 0, new_line = true, line = null
  for $j = 0; j < m; j$++ **do**
    if new_line **then**
      for $t = 0; t < 5; t$++ **do**
        line.push_back(p_{j+t}^i) // Save temporary line segments
      end for+
      new_line = false
    end if
    l_{im} = LEAST_SQUARE(line.x,line.y)
    l_{depth} = LEAST_SQUARE(line.xn,line.Z)
    if DIST(l_{im},point($\mathrm{p_j^i}$.x, $\mathrm{p_j^i}$.y)) $< \mathrm{e_1}$
    and DIST(l_{depth},point($\mathrm{p_j^i}$.xn, $\mathrm{p_j^i}$.Z)) $< \mathrm{e_2}$ **then**
      line.push_back($\mathrm{p_j^i}$), fit_check=0
    else
      fit_check++
    end if
    if fit_check>5 **then**
      if line.size> 5 **then**
        $\mathrm{LS_k}$.push_back(line)
      end if
      line = null, new_line = true
      fit_check = 0
    end if
  end for
  if line.size> 5 **then**
    $\mathrm{LS_k}$.push_back(line)
  end if
end for
return $\mathrm{LS_k}$

segments of the all keyframes can be expressed as $\mathrm{ES} = \{\mathrm{ES_1}, \mathrm{ES_2}, ..., \mathrm{ES_n}\}$ where $\mathrm{ES_k}$ denotes the edge segment set of the k-th keyframe. $\mathrm{ES_k} = \{\mathrm{es_1}, \mathrm{es_2}, ..., \mathrm{es_m}\}$ where $\mathrm{es_i} = \{\mathrm{p_1}, \mathrm{p_2}, ..., \mathrm{p_t}\}$ is an edge segment formulated as a pixel chain. $\mathrm{p_j}$ represents a pixel which is a vector of (x, y, Z) where x and y are the image coordinates and Z is its corresponding depth. The number of key frames, number of edge segments in a keyframe and number of pixels in an edge segment are denoted by n, m and t respectively. It is worth noting that image pixels with high gradient are more likely to be selected for computing depth value in the SLAM system. Edge segments are detected based on pixel intensity gradients. Thus, most of the detected edge pixels will have depth values projected from the depth map. We assign infinity to those pixels which have no depth values.

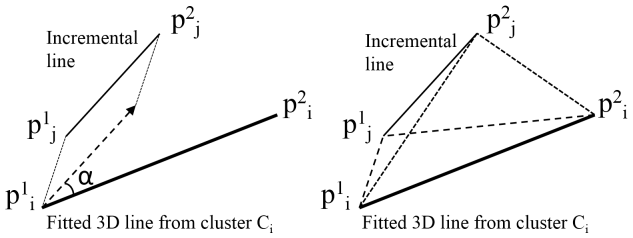

Figure 3. Clustering by angle and distance. $p^1_i p^2_i$ is fitted from an existing 3D line segment cluster, $p^1_j p^2_j$ is an unclustered 3D line segment.

3D line segments of a keyframe are extracted from those detected edge segments by Algorithm 1. The main idea of this algorithm is to reduce a 3D line fitting problem to two 2D line fitting problems. For each edge segment, the algorithm initially takes its first five pixels to fit two 2D lines (l_{im} and l_{depth}) in image coordinate frame and the $\mathbf{p_1}$-$\mathbf{xz}$ coordinate frame using least square method. The coordinate frames are defined in Figure. 2. The line l_{im} is fitted based on the pixels' (x, y) values while l_{depth} is fitted based on (xn, Z). Z is the pixel's depth and xn is the distance from p_1 to the pixel's projection on axis x. Given the next pixel in the pixel chain, we compute its distances to l_{im} and l_{depth} in their corresponding coordinate frames. It is worth noting that xn and Z have different units. To have the same unit with xn, Z is multiplied by the focal length f before distance computation. If both distances are smaller than certain thresholds (both are 1.0 in all our experiments), we will add the pixel to the fitted pixel set to extend the line. Otherwise the pixel will be considered as an outlier. If five consecutive pixels are outliers, we stop the current line segment searching and start a new line segment searching procedure. After traversing all of the edge segments of the keyframes, we can achieve one 3D line segment set $\mathrm{LS_k}$ for each keyframe.

3.3. 3D line segment clustering and filtering

To obtain a consistent reconstruction of the environment, 3D line segments $\mathrm{LS} = \{\mathrm{LS_1}, \mathrm{LS_2}, ..., \mathrm{LS_n}\}$ extracted from different keyframes are first registered to the same world coordinate system. The registered 3D line segments are denoted as $\mathrm{ls} = \{\mathrm{ls_1}, \mathrm{ls_2}, ..., \mathrm{ls_w}\}$. w denotes the total number of 3D line segments on all keyframes. Directly registering all of 3D line segments of each keyframe will produce redundant and slightly misaligned 3D line segments. We address this problem by proposing a simple incremental merging method.

The main idea of our merging method is clustering closely located 3D line segments and fitting those cluster sets with new 3D line segments. As illustrated in Figure 3, the angle and distance measures are used for clustering. The angle measure α is defined as:

$$\alpha = acos\left(\frac{\overrightarrow{p^1_j p^2_j} \cdot \overrightarrow{p^1_i p^2_i}}{|\overrightarrow{p^1_j p^2_j}||\overrightarrow{p^1_i p^2_i}|}\right) \tag{1}$$

The distance measure d is computed as:

$$d = min(d_1, d_2) \tag{2}$$

$$d_1 = |\overrightarrow{p^1_j p^1_i}| + |\overrightarrow{p^1_j p^2_i}| - |\overrightarrow{p^1_i p^2_i}| \tag{3}$$

$$d_2 = |\overrightarrow{p^2_j p^1_i}| + |\overrightarrow{p^2_j p^2_i}| - |\overrightarrow{p^1_i p^2_i}| \tag{4}$$

Specifically, we take the first 3D line segment $\mathrm{ls_1}$ as the initial cluster $\mathrm{C_1}$. Then, we compute the angle and distance measure between the initial cluster (single line segment) and the next 3D line segment $\mathrm{ls_2}$. If the angle α and distance d are smaller than certain thresholds (e.g. 10.0 and 0.02 respectively in all our experiments), we add $\mathrm{ls_2}$ to the cluster $\mathrm{C_1}$. Otherwise, we create a new cluster $\mathrm{C_2}$. For each cluster, if it contains more than one 3D line segments, we will fit a new 3D line segment to represent the cluster. The direction of the new line segment is determined by performing Singular Value Decomposition (SVD) on the matrix consisting of all the end points $\mathrm{X_{ep}}$ of line segments in the cluster. A new 3D infinite line is then determined by the direction together with the centroid of $\mathrm{X_{ep}}$. Our objective is to obtain a 3D line segment from this infinite line. We project end points $\mathrm{X_{ep}}$ onto the newly generated infinite 3D line and compute the furthest projections with respect to the centroid in both directions. The 3D line segment between these two furthest projection points is taken as the fitted line segment of the cluster. This process is repeated until all the line segments in ls are clustered. Clusters with small size ($\mathrm{num(C_i)} < 3$) are filtered out in the end. In this way, we can merge a large number of line segments into fewer clusters and generate new 3D line segments with higher quality.

4. Experimental results

In this section, we test our 3D line segment extraction method on four sequences from the TUM RGB-D dataset [17] and estimate its performance in terms of compactness and completeness.

4.1. Implementation

The experiments in this section are performed on a desktop computer with an 8-core Intel i7-6700k CPU. We use the open source ORB-SLAM2 [13] as our base system. We implement the semi-dense module in C++ as described in [12]. The parameters in ORB-SLAM are kept as default, and the parameters for semi-dense module are set as presented in [12].

4.2. Decoupled 3D line segment fitting

To further demonstrate the capability of our method, we compare it to a decoupled 3D line segment fitting method using 2D line segments given by EDLines [1]. Given detected line segments and the depth information on some of the pixels along the line segments, we can easily estimate the 3D line segment position by performing a single 2D line fitting. In this case, we have a fixed number of pixels on the line segment since we do not need to iteratively search along pixel chains and extend line segments. Therefore we can efficiently perform RANSAC to remove outliers in one shot before the line fitting process. With the fitted line, we can compute the 3D location of the endpoints and reconstruct the 3D line segment.

4.3. Comparison

The results of our 3D line segment extraction method on the test sequences are illustrated in the last two rows of Figure 5. Our results accurately fit the semi-dense point clouds shown in the second row of Figure 5. They still capture the major structures of the scene while reducing the number of 3D elements greatly.

4.3.1 Accuracy and Completeness

First, we compare our results with those from Line3D++. The results of Line3D++ [6] on the test sequences is shown in the third row of Figure 5. In our experiments, Line3D++ uses line segments detected by EDLines together with the keyframe images and camera poses output by ORB-SLAM to construct 3D line segments. However, it is fragile in some cases, such as complex indoor environment or scenes without long, straight line segments. Therefore, Line3D++ tends to produce a large number of outliers due to the ambiguity of geometric line matching in such cases. On the contrary, our method utilizes the accurate semi-dense depth map. Since the depth maps are checked multiple times and filtered to produce confident points, the results of our method have fewer outliers.

In contrast to Line3D++, semi-dense points can cover regions with large image gradient, such as boundaries, contours, where straight lines may be absent. Since our method takes both intensity and depth information into consideration, it is robust to outliers caused by intensity noise so that it can extract shorter line segments than EDLines. Thus, our results fit curves better and captures finer details than Line3D++. As shown in Figure 5, the completeness of our method is also better than Line3D++.

The results of decoupled line segment fitting are presented in the forth row in Figure 5. Compared to the edge aided 3D line fitting which tries to utilize pixel position and depth simultaneously, the decoupled fitting essentially fits lines in image plane and depth plane in two steps. The error from line fitting in the image plane will be propagated to the error of 3D line segment position, which result in an inaccurate reconstruction compared to our method, as shown in Figure 4. Worth mentioning is that the decoupled fitting tends to generate longer segments since only the pixel position is considered in the image plane line fitting process. Longer segments will make the error propagation even worse because the total error of line segments in image space might be larger. Another source of error is that the EDLines may detect a long line segment which is not a continuous line in 3D space. Trying to fit a single 3D line segment onto the 2D segment in this case will result in a large error of the estimated 3D line segment. On the other hand, in our method, if any of the two errors of line fitting grows higher than the threshold, we stop the line fitting and start a new line fitting process. In this way, the error accumulated from image plane and depth are bounded, therefore prevent the line segment to being far away from the points.

4.3.2 Compactness

As shown in Table 1, the point clouds are greatly simplified with our edge aided 3D line fitting algorithm. The results are simplified further to present a clean structure of the scene using our 3D line segments clustering process. Note that although Line3D++ produces the fewest number of vertices in the reconstruction, the completeness of reconstruction is generally worse than our method as shown in Figure 5. For example, in the third sequence, Line3D++ completely missed the structure in the background while our method captures the environment in a compact manner. In conclusion, our method achieves a better balance between completeness and compactness.

4.3.3 Running Time

We present the average running time of the edge aided 3D line segment fitting in Table 2. Our line segment fitting method runs fast and efficiently while utilizing large amount of depth information. Compared to the running time of edge aided 3D fitting, decoupled 3D fitting need additional computation time for performing RANSAC. Because the segments are usually much longer in decoupled 3D line segment fitting, RANSAC is necessary in order to obtain a good fit for the larger pixel set on the line segments.

5. Conclusions and discussions

In this paper, we present a 3D line segment based method to simplify semi-dense point cloud output by keyframe-base SLAM system. The main contribution lies in the novel edge aided 3D line segment extraction algorithm which solely relies on the image and the semi-dense depth map for individ-

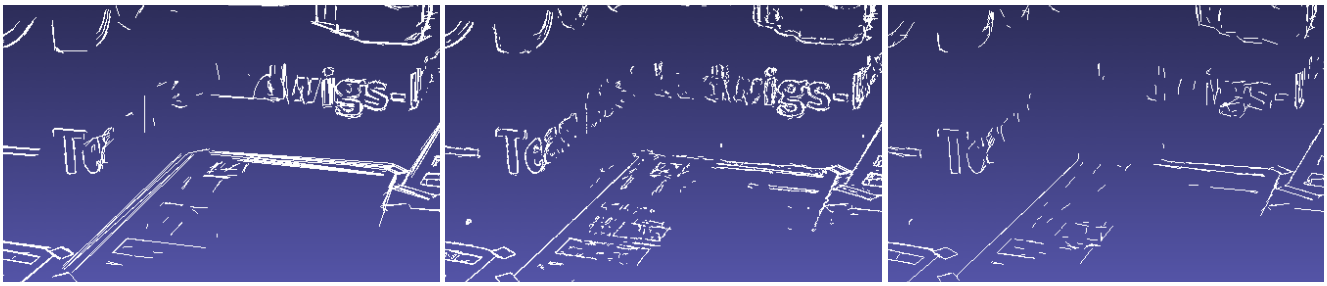

Figure 4. Comparison of detail in sequence fr3-structure-texture-near sequence. Left to right: decoupled 3D line segment fitting, edge aided 3D fitting without (w/o) clustering, edge aided 3D fitting with (w/) clustering

Method	fr3-nostructure-texture-near	fr3-structure-texture-near	fr3-large-cabinet	fr1-room
Semi-dense point cloud	437629	351882	263637	1044752
Line3D++	1346	506	124	330
Decoupled 3D fitting	3710	3686	1106	9966
Edge aided w/o clustering	31772	18674	15760	42624
Edge aided w/ clustering	2296	1546	1304	3334

Table 1. Number of vertices in the reconstruction

Method	Average Time (ms)
Decoupled 3D fitting	17.63
Edge aided 3D fitting	9.42

Table 2. Running time per keyframe

ual keyframes. Our method tries to minimize the line fitting error on both image plane and depth plane simultaneously as the line segment grows. By incrementally clustering the line segments detected on each keyframe, we can obtain a compact and complete 3D line segment reconstruction for the scene. Compared to using the line segments produced by line segment detectors and minimizing the line fitting error on the depth plane afterwards, our method achieves better accuracy with respect to the point cloud.

There are several limitations of our method: 1. Our method relies on the accuracy and completeness of the semi-dense point cloud output by SLAM. Although the line segment fitting method is robust to individual outliers in the depth map, the reconstructed line segments will be affected if the point cloud is not well reconstructed. 2. The current implementation is not real-time due to the clustering process. We will improve it in the future. 3. ORB-SLAM is a feature based SLAM system and originally designed to produce sparse feature point cloud. Although the semi-dense module outputs accurate point cloud, the semi-dense depth map is generated with a delay of a few keyframes. Using a direct method which can generate semi-dense depth map without delay will enable our method to detect 3D line segments in real-time.

One possible application of our method is to improve the efficiency of surface extraction [9, 7] by simplifying semi-dense point cloud with structural geometric information. This will make real-time surface extraction possible when using semi-dense point cloud. In the future, we plan to improve our algorithm in order to reconstruct a consistent and compact line segment based surface model in real-time.

References

[1] C. Akinlar and C. Topal. Edlines: A real-time line segment detector with a false detection control. *Pattern Recognition Letters*, 32(13):1633–1642, 2011. 5

[2] J. Engel, V. Koltun, and D. Cremers. Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017. 2, 3

[3] J. Engel, T. Schöps, and D. Cremers. Lsd-slam: Large-scale direct monocular slam. In *European Conference on Computer Vision*, pages 834–849. Springer, 2014. 2, 3

[4] Y. Furukawa and J. Ponce. Accurate, dense, and robust multiview stereopsis. *IEEE transactions on pattern analysis and machine intelligence*, 32(8):1362–1376, 2010. 1

[5] M. Goesele, N. Snavely, B. Curless, H. Hoppe, and S. M. Seitz. Multi-view stereo for community photo collections. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007. 1

[6] M. Hofer, M. Maurer, and H. Bischof. Efficient 3d scene abstraction using line segments. *Computer Vision and Image Understanding*, 157:167–178, 2017. 1, 2, 5

[7] C. Hoppe, M. Klopschitz, M. Donoser, and H. Bischof. Incremental surface extraction from sparse structure-from-motion point clouds. In *BMVC*, pages 94–1, 2013. 6

[8] G. Klein and D. Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007.*

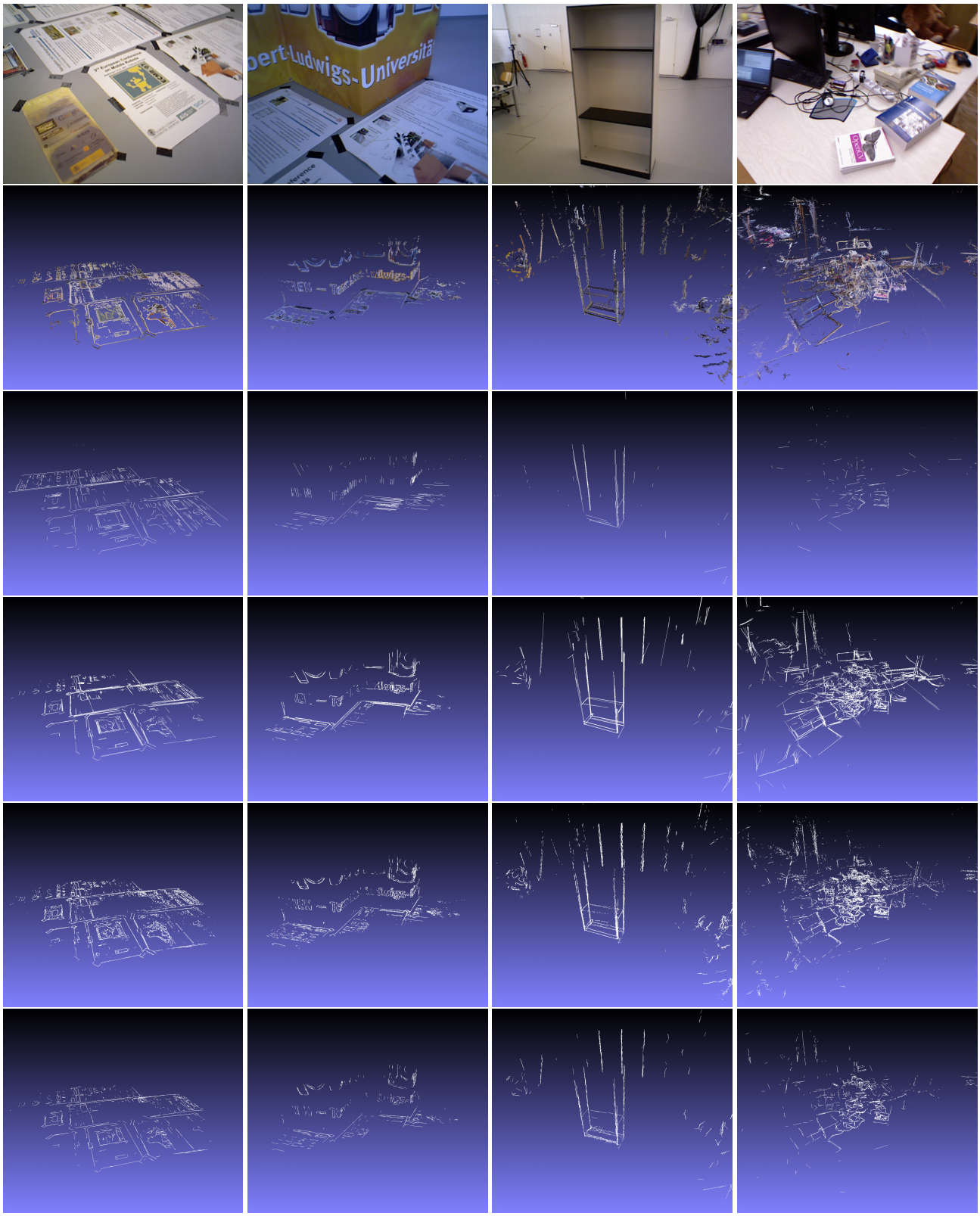

Figure 5. Experimental results. Top to bottom: sample image from the sequence, semi-dense point cloud, results of Line3d++, results of decoupled 3D fitting using EDLines, results of our edge-aided 3D fitting without clustering, results of our edge-aided 3D fitting with clustering. Left to right (four sequences): fr3-nostructure-textrue-near, fr3-structure-texture-near, fr3-large-cabinet, fr1-room.

ISMAR 2007. 6th IEEE and ACM International Symposium on, pages 225–234. IEEE, 2007. 2

[9] D. I. Lovi. Incremental free-space carving for real-time 3d reconstruction. 2011. 6

[10] P. Moulon, P. Monasse, R. Marlet, and Others. Openmvg. an open multiple view geometry library. https://github.com/openMVG/openMVG. 1

[11] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015. 2

[12] R. Mur-Artal and J. D. Tardós. Probabilistic semi-dense mapping from highly accurate feature-based monocular slam. 2, 4

[13] 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*, 2017. 4

[14] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison. Dtam: Dense tracking and mapping in real-time. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2320–2327. IEEE, 2011. 2

[15] A. Pumarola, A. Vakhitov, A. Agudo, A. Sanfeliu, and F. Moreno-Noguer. PL-SLAM: Real-Time Monocular Visual SLAM with Points and Lines. In *International Conference in Robotics and Automation*, 2017. 1

[16] A. Pumarola, A. Vakhitov, A. Agudo, A. Sanfeliu, and F. Moreno-Noguer. Pl-slam: Real-time monocular visual slam with points and lines. In *Proc. International Conference on Robotics and Automation (ICRA), IEEE*, 2017. 2

[17] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012. 4

[18] C. Topal and C. Akinlar. Edge drawing: A combined real-time edge and segment detector. *J. Visual Communication and Image Representation*, 23(6):862–872, 2012. 1, 3

[19] R. G. Von Gioi, J. Jakubowicz, J.-M. Morel, and G. Randall. Lsd: A fast line segment detector with a false detection control. *IEEE transactions on pattern analysis and machine intelligence*, 32(4):722–732, 2010. 2

[20] C. Wu et al. Visualsfm: A visual structure from motion system. 2011. 1

[21] S. Yang and S. Scherer. Direct monocular odometry using points and lines. *arXiv preprint arXiv:1703.06380*, 2017. 2

[22] L. Zhang and R. Koch. An efficient and robust line segment matching approach based on lbd descriptor and pairwise geometric consistency. *Journal of Visual Communication and Image Representation*, 24(7):794–805, 2013. 2