IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

Efficient Plane Segmentation in Depth Image Based on Adaptive Patch-wise Region Growing

Lantao Zhang[1], Haochen Niu[1], Peilin Liu[1], Fei Wen[1] and Rendong Ying*[1]

***Abstract*—Plane segmentation algorithms are widely used in robotics, serving key roles in scenarios such as indoor localization, scene understanding, and robotic manipulation. These applications typically require real-time, precise, and robust plane segmentation processing, which presents a significant challenge. Existing methods based on pixel-wise or fix-sized patch-wise operation are redundant, as planar regions in real-world scenes are of diverse sizes. In this paper, we introduce a highly efficient method for plane segmentation, namely Adaptive Patch-wise Region Growing (APRG). APRG begins with data sampling to construct a data pyramid. To avoid redundant planer fitting in large planar regions, we introduce an adaptive patch-wise plane fitting algorithm with the pyramid accessed in a top-down manner. The largest possible planar patches are obtained in this process. Subsequently we introduce a region growing algorithm specially designed for our patch representation. Overall, APRG achieves more than 600 FPS at a 640x480 resolution on a mid-range CPU without using parallel acceleration techniques, which outperforms the state-of-the-art method by a factor of 1.46. Besides, in addition to its speedup in run-time, APRG significantly improves the segmentation quality, especially on real-world data. Code is available at ZhangLanTao/APRG.**

***Index Terms*—RGB-D Perception; Object Detection, Segmentation and Categorization.**

I. Introduction

DEPTH sensing cameras are widely used in robots to acquire high resolution depth image of the scenes, which provide robots with the capability of accurate 3D sensing. Plane segmentation plays an important role for understanding the 3D geometry of a scene from depth images. It is an essential prerequisite for many post-processing functions in robotics, such as geometry-centric visual odometry (VO) [1], [2], scene understanding [3], sensor calibration [4], and pose estimation [5], [6], which commonly rely on essential information of the 3D planes in the scenes. Therefore, plane segmentation has drawn much research interest over the past few years. Though much progress has been made, it still faces challenge in many real-world robotic applications that require real-time and high-precision segmentation [7], especially in the scenarios of edge-device computing in industrial and mobile robots with limited computing resources.

Manuscript received: October, 28, 2024; Revised January, 19, 2025; Accepted March, 19, 2025.

This paper was recommended for publication by Editor Cesar Cadena upon evaluation of the Associate Editor and Reviewers' comments.

This work was supported by National Natural Science Foundation of China under Grant 62276166.

[1]All authors are with the Brain-Inspired Application Technology Center (BATC), School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China. https://bat.sjtu.edu.cn/zh/

Digital Object Identifier (DOI): see top of this page.

Traditional RANSAC based plane segmentation approaches require repetitive random sampling, parameter fitting and exhaustive examination of each pixel, which incurs a high computational cost. Recently, Roychoudhury et al. [8] proposes a plane segmentation method using depth-dependent flood fill, namely DDPFF. It uses region growing algorithm and has achieved improvement in both speedup and accuracy. Nevertheless, pixel-wise region growing still necessitates the examination of each pixel, which is time-consuming. In contrast, methods such as D-KHT [9] and CAPE [10] firstly segment an image into fix-sized patches and then employ patch-wise Hough transform or region growing. These methods substantially reduce the computational load. However, the process of obtaining fixed-sized patches requires multiple plane fittings for a single large planar region, resulting in redundant calculation. Furthermore, this duplication in patch representation introduces redundancy in the subsequent Hough transform or region growing steps.

Recently, deep learning-based methods, such as the segment anything model (SAM) [11], have shown a significant advancement. However, despite their excellent capabilities, these methods are computationally expensive, making them unsuitable for deployment on resource-limited edge devices. Moreover, their reliance on the texture information of color images can result in segmentation outputs with ambiguous geometric interpretation.

To address these issues, we introduce Adaptive Patch-wise Region Growing (APRG) for plane segmentation. Our method adaptively segments the depth image into patches of variable sizes. Subsequently, region growing is performed on these patches. Utilizing the sparsity of large planar regions, our adaptive-patch design reduces the computation in planar patch fitting, and reduces the number of patches in the region growing step, which results in higher computational efficiency. Besides, the top-down segmentation process makes our method more robust to noise.

In summary, the main contributions are as follows.

1) We propose a novel plane segmentation algorithm tailored for depth images, which employs an adaptive-patch representation to exploit the sparsity of planar regions, thereby reduces the computational cost of plane fitting. In this algorithm, we design a pyramid representation of the input data and perform patch extraction in a top-down manner, which not only speedups the processing but also enhances the robustness against noise.
2) We propose a region growing algorithm based on our adaptive-patch representation, which reduces the computational cost compared to traditional fix-sized region growing processes, resulting in higher efficiency.

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

3) Comprehensive experiments demonstrate that our method outperforms state-of-the-art methods in both computational efficiency and accuracy.

II. Related Works

Existing methods can be mainly divided into RANSAC based, Hough Transform (HT) based, clustering based and deep learning-based approaches.

RANSAC [12] is widely used for estimating model parameters by iterative sampling points to fit models and evaluating fitting quality. However, it is computationally inefficient and sensitive to noise in scenes with an unknown number of planes. To improve efficiency, Li et al. [13] apply voxelization and ran RANSAC at the voxel level. Parallel RANSAC [14] use GPUs for speedup. Schnabel et al. [15] introduce a unified framework for detecting various geometric primitives with enhanced inlier criteria.

HT [16] is a model-fitting method originally developed for line extraction in binary images. The HT framework can be used as long as constraints on the parameters can be obtained from local data, thus is widely used to extract other primitives [17]. In HT based plane segmentation methods, parameters such as the normal vector and distance are accumulated through voting, and local maxima in the accumulator space indicate potential planes.

While robust to noise and missing data, standard HT faces challenges with balancing accuracy and computation cost, as well as non-uniform precision in parameter space. To address these, modifications have been made in various methods. The work [18] combine Randomized Hough Transform (RHT) with angle and distance constraints to filter out non-ground elements. Tian et al. [19] introduce a parallel 3D HT algorithm for precise plane detection in 3D LiDAR point clouds. Vera et al. [9] employ an implicit quadtree to obtain clusters of nearly coplanar points in the 2.5-D scene. PCA and a specified minimum number of valid depth pixels in tree nodes determine how the quadtree subdivides. The maximum tree height is predetermined, resulting in a predictable computational cost.

Both RANSAC and HT methods necessitate supplementary stages for boundary segmentation, typically achieved through clustering or region growing approach. This two-step method incurs higher computational costs, whereas a single clustering-based or region-growing-based method can achieve the same outcome more efficiently.

Weingarten et al. [20] divides the point cloud into voxels and applies RANSAC independently within each voxel. Finally, the planes are merged using region growing. Czerniawski et al. [21] utilize the DBSCAN clustering method within a six-dimensional parameter space to segment planes within point clouds. Points characterized by spatial continuity and nearly identical normal vector directions are grouped into plane clusters. Clustering on the original point cloud data requires evaluating criteria for all points, making it time-consuming. Vo et al. [22] divide the raw point cloud into voxels, and apply region growing to form clusters. Du et al. [23] utilize the inherent data orderliness of spinning multibeam LiDARs, efficiently extract line segments in 2D-space. They then cluster these line primitives into planes. Xu et al. [24] introduce a method with a local phase using an optimal vector field to detect plane intersections and a global phase for smoothing. They improve accuracy by integrating local and global constraints. Roychoudhury et al. [8] and Proença et al. [10] leverage data orderliness of depth images, perform region growing in 2D image, achieve fastest processing speed among other methods. However, the method [8] perform pixel-wise region growing and need to evaluate every pixel, thus is computational expensive and is more sensitive to noise. Proença et al. [10] use patch-wise region growing, they firstly segment the input image into fix-sized patches, then perform region growing among the patches, thus can achieve real-time performance of more than 100 FPS. But the optimal patch size need to be manually set for best performance in certain data. Besides, in most scenes, there are large planar regions, it is duplicated to segment large planar regions into small patches.

Recently, deep learning-based plane segmentation methods have attracted much attention. Liu et al. [25] introduce an end-to-end network for piece-wise planar depthmap reconstruction from a single RGB image, which directly infers plane parameters and segmentation masks. They later developed PlaneRCNN [26], a deep neural architecture that detects and reconstructs planar surfaces using a variant of Mask R-CNN [27]. Learning-based approaches heavily rely on data, necessitating extensive annotations. Furthermore, these methods demand substantial computational resources, making them challenging to employ in real-time, highly time-sensitive robotic applications.

III. Method

The scheme of the proposed method is illustrated in Fig. 1. APRG has four main steps: Firstly, the input depth image is converted to point cloud and then down sampled to construct a data pyramid. Secondly, planar patches of various sizes are adaptively obtained through top-to-bottom data accessing and parameter fitting within the pyramid. Detailed patch information is stored in a status pyramid. Next, region growing is performed, grouping adjacent and coplanar patches into planar regions. Finally, pixel-level labels are assigned by evaluating coplanarity within the boarder of each region. Details are giving in the following sections.

A. Data Structure and Patch Representation

An input depth image, with dimension $w \cdot h$ and one channel, is firstly converted into a point cloud. The point cloud is stored in an image-like format, i.e., with three $w \cdot h$ matrices representing x, y, and z coordinates, respectively. This enables seamless and efficient patch-based access. The point cloud is then down-sampled to create a data pyramid, where the raw point cloud is stored in layer 0, and each successive layer down-samples the x, y, z matrices, halving its width and height. This pyramid enables easy correspondence between pixels across layers. As shown in Fig. 2(a), a pixel at position $[r, c]$ in layer l corresponds to the pixel at position $[2r, 2p]$ in layer $l-1$. Any patch can be represented by pyramid index $i_p = [l, r, c, s]$, where l stands for its layer, (r, c) is

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

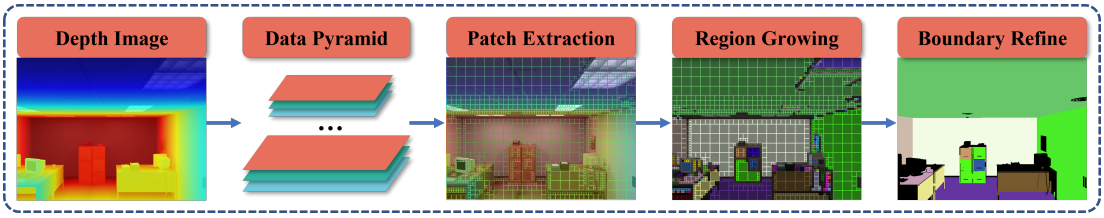
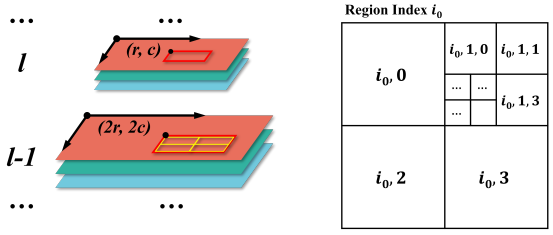


Fig. 1. Scheme of APRG. The input depth image (we show pseudo color image for better visualization) is converted to point cloud, based on which a data pyramid is built. Planar patches of variable sizes are extracted recursively. The patch-wise region growing incorporates the patches to form regions. Finally, boundary pixels are refined to get the final pixel-wise segmentation result.



(a) (b)

Fig. 2. (a) An example of patch correspondence between two layers. (b) An example of hierarchical index of region i_0.

the up-left corner pixel coordinate, s is the size of the path. The corresponding area in layer $l-1$ can be represented by $[l-1, 2r, 2c, 2s]$. This area contains four child patches $[l-1, 2r, 2c, s]$, $[l-1, 2r+s, 2c, s]$, $[l-1, 2r, 2c+s, s]$ and $[l-1, 2r+s, 2c+s, s]$.

A status pyramid is used to track the progress of the algorithm. It shares a similar data structure and indexing method with the data pyramid, making it convenient to record information across layers.

We use a hierarchical index $i_h = [i_0, i_1, ..., i_n], (n \leq N)$ to represent the patch in the original image, with index values indicating spatial positions, N is the maximum layer number. Specifically, the $w \cdot h$ matrices are firstly divided into $w_0 \cdot h_0$ square regions. The size of each square region is denoted as $s_{init} = w/w_0 = h/h_0$. i_0 indicates the region in which a patch is located. $[i_1, i_2, ..., i_n]$ indicates the relative position inside the region (with value 0, 1, 2, 3 corresponding to the upper-left, upper-right, lower-left, and lower-right orientations, respectively. An example is shown in Fig. 2(b)). N is determined by s_{init} and the minimum patch size s_{min} as

$$N = \log_2 \left(\frac{s_{init}}{s_{min}} \right). \tag{1}$$

For example, given an input depth image with a resolution of 640×480, an initial 4×3 partitioning results in 12 square regions with size $s_{init} = 160$. When $s_{min} = 5$, $N = 5$.

To reduce redundancy, APRG uses a fixed number of s_{min}^2 points to perform plane fitting within any patch. Specifically, for any patch with the hierarchical index $i_h = [i_0, i_1, ..., i_n]$, the sampled points are directly extracted from the data pyramid with index $i_p = [N-n, r, c, s_{min}]$. The up-left position (r, c) is calculated as

$$r = 2^n s_{min} \left(\lfloor \frac{i_0}{w_0} \rfloor \right) + \sum_{j=1}^{n} 2^{n-j} s_{min} \left(\lfloor \frac{i_j}{2} \rfloor \right), \tag{2}$$

$$c = 2^n s_{min} (i_0 \bmod w_0) + \sum_{j=1}^{n} 2^{n-j} s_{min} (i_j \bmod 2), \tag{3}$$

where mod denotes the modulo operation and $\lfloor \cdot \rfloor$ represents the floor division.

B. Patch Segmentation

We employ a recursive process to hierarchically segment planar patches, as outlined in Algorithm 1. The input data includes the previously mentioned data pyramid, where $\mathbf{D}_i$ refers to the points associated with the hierarchical index i. We utilize a queue structure initialized with $w_0 \cdot h_0$ indexes. Each index in the queue is checked, if the point cloud corresponding to the index is planar, the patch information is recorded in the status pyramid. If the point cloud is not planar, the four child indexes are added to the queue for further processing.

Algorithm 1: Recursive Planar Patch Segmentation

```
input : Data Pyramid D,
        Pyramid Layers N
output: Status Pyramid S
Q ← {[0], [1], ..., [w_0 · h_0 − 1]};
while Q is not empty do
    i = Q.pop();
    flag = ContinuityCheck(D_i);
    ϵ_i = PlanarCheck(D_i);
    if ϵ_i < ϵ_th and flag then
        SaveInfo(S, i) ;
    else
        if length(i) < N then
            Q.push(children indexes of i);
        end
    end
end
```

Beginning with hierarchical indexes of length 1, we conduct a continuity check same as the approach [10]. For continuous patches, plane fitting is performed through principal component analysis (PCA), where the mean squared error (MSE) corresponds to the lowest eigenvalue, and the plane normal vector corresponds to the eigenvector of the lowest eigenvalue. For patches with MSE higher than threshold ϵ_{th}, their four child patches are examined. The same procedure are recursively performed until reaching the maximum hierarchical index length.

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

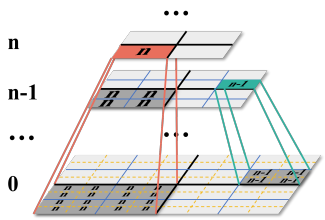


Fig. 3. An example of status pyramid. When a planar patch is found, the patch information is stored in the status pyramid with the same hierarchical index. The layer number is stored in all the child patches across lower layers.

Once a planar patch is identified, the patch information, including plane parameters, layer number, and other details, is stored in the status pyramid with the same hierarchical index. The layer number is also stored in all child patches across lower layers (as shown in Fig. 3) to facilitate subsequent neighboring search.

C. Patch-wise Region Growing

Region growing is applied to merge the co-planar patches. This process starts with seed selection. Rather than calculating a histogram of normals and select the planar cell with most confidence, we simply start from the remaining largest patch. Next, we employ a breadth-first search strategy to expand regions. This large-to-small design is more efficient and less sensitive to noise.

As outlined in Algorithm 2, the input data is status pyramid $\mathbf{S}$. We first select the largest unlabeled and unvisited patch as the seed point and add it to the process queue $\mathcal{Q}$. The set $\mathcal{R}$ is used to store the patch indexes in the current region. Then for each patch $q \in \mathcal{Q}$, the neighboring patch set $\mathcal{N}$ is obtained. $\mathcal{N}$ is sorted by area, ensuring that the region growing process prioritizes larger blocks first. Afterwards, each neighboring patch $n \in \mathcal{N}$ is examined, if n is coplanar with q, it is added to $\mathcal{R}$ and $\mathcal{Q}$. This process continues until $\mathcal{Q}$ is empty. Finally, region labels of the patches in $\mathcal{R}$ are stored in the status pyramid $\mathbf{S}$. The algorithm then proceeds to next region.

Algorithm 2: Patch-wise Region Growing

```
input : Status Pyramid S
seed = ChooseSeed(S);
while seed ≠ ∅ do
    R ← ∅;
    Q ← {seed};
    while Q is not empty do
        q = Q.pop();
        N = NeighborOf(q);
        N = Sort(N);
        for n ∈ N do
            if n is coplanar with q then
                Q.push(n);
                R.push(n);
                SetVisited(S, n);
            end
        end
    end
    WriteLabel(S, R);
    seed = ChooseSeed(S);
end
```

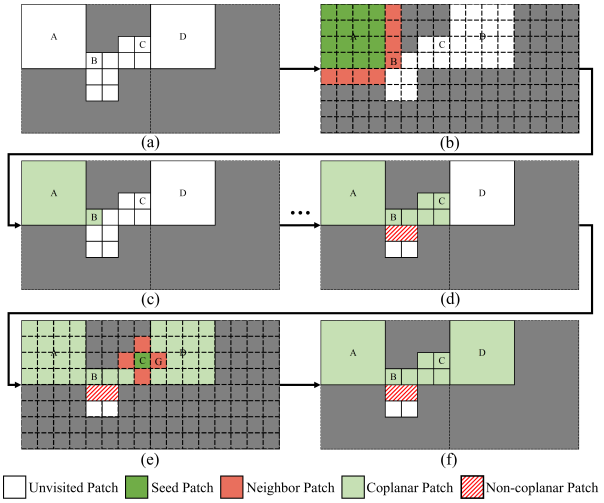


Fig. 4. An example of region growing process. (a) The patch segmentation result. Planar patches are in white, non-planar area are in gray. (b) Neighbor search from seed patch A. The seed patch is in green, the virtual minimal patches are in red. (c) Patch A and B are marked visited, the region expand in case A and B are coplanar. (d) Current status after several region growing operations. The non-coplanar patches are not added to the processing queue, thus their neighbor (two patches at the bottom) is not visited. (e) Patch C is the current seed. (f) Patch D is visited, the process of one region ends. Region label is stored in the status pyramid.

An example is shown in Fig. 4. In our approach, since the patch sizes are variable, typical neighbour search method can not be applied. We introduce a method specifically designed for our segmentation data structure. Given a seed patch with hierarchical index $i_h = [i_0, i_1, ..., i_n]$, the up-left corner point p_1 in layer $N - n$ can be obtained using (2) and (3), the up-right, bottom-left and bottom-right coordinates p_2 to p_4 are computed as

$$p_2 = p_1 + \begin{bmatrix} 1 \\ 0 \end{bmatrix} s_{min}, \tag{4}$$

$$p_3 = p_1 + \begin{bmatrix} 0 \\ 1 \end{bmatrix} s_{min}, \tag{5}$$

$$p_4 = p_1 + \begin{bmatrix} 1 \\ 1 \end{bmatrix} s_{min}, \tag{6}$$

where s_{min} is the patch size. The corresponding four points in layer 0 are computed as

$$p_i' = 2^{N-n} p_i. \tag{7}$$

Fig. 4 (a), (b) and (c) show the process of region growing from large seed patch to the minimum patch (from A to B). The adjacent minimum patch indexes can be obtained by adding a row or column bias b to four edges. For top and left side edges, $b = -s_{min}$, and for right and bottom side edges, $b = 0$. The patch information can be directly accessed in $\mathbf{S}$ by the index.

Fig. 4 (d), (e) and (f) show the process of region growing from small patch to large patch (from C to D). We firstly obtain the minimum patch indexes (i.e. patch G), termed virtual minimum patches, as these patches are not guaranteed to align with the actual patch segmentation status (i.e. patch D). Afterwards, we check the layer number l as indicated in Fig. 3. The actual neighbor index is obtained by extracting

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

only the first l elements from the hierarchical index of virtual minimum patch.

D. Pixel-wise Boundary Refinement

The patch-wise segmentation result is coarse, and in certain scenarios, precise edge detection is necessary. To address this, we adopt the method from [10], which employs cell-based morphological operations to detect boundary cells and subsequently refine the pixels. The segmentation result is firstly converted into region masks, with each element corresponding to a minimum patch. Next, each mask $\mathbf{M}$ undergoes erosion and dilation using a 3×3 kernel. Regions that are entirely eroded are discarded. The boundary mask $\mathbf{M}_B$ is defined as the difference between the eroded and dilated masks:

$$\mathbf{M}_B = dialate(\mathbf{M}) - erode(\mathbf{M}). \tag{8}$$

Subsequently, the point-to-plane distances are computed within the mask $\mathbf{M}_B$. Pixels are assigned to the region if their distance is less than the MSE multiplied by a constant ratio, e.g., $k = 9$ is used in this paper. In case where a cell is shared by multiple regions, point-to-plane distances are calculated for each region, and the pixel is assigned to the region with the minimum distance.

IV. Experimental Evaluation

A. Datasets and Metrics

We evaluate our method on the EVOPS dataset [7], which provides labeled sequences for four TUM [28] real sequences and two ICL-NUIM [29] synthetic sequences. The TUM RGB-D dataset contains challenging indoor scenes with complex motion and varying lighting conditions, providing a benchmark for real-world applications. The dataset is captured using the Kinect for Xbox 360 camera, which provides synchronized RGB and depth images at a resolution of 640×480. However, the Kinect depth sensor exhibits significant noise characteristics, with the depth noise variance following a second-order polynomial relationship with distance. According to the experiments reported in [30], the depth noise can be approximated by the empirical formula:

$$\sigma_Z = \left(9.0 \times Z^2 - 26.5 \times Z + 20.237\right) \times 10^{-3}, \tag{9}$$

where Z is the depth value in meters. Within the default working range of 0.8m to 4m, the depth noise variance ranges from 4.8mm to 58.2mm, indicating a relatively high noise level. The ICL-NUIM synthetic dataset, on the other hand, offers controlled environments that allow for precise performance evaluation without noise.

We adopt the same evaluation method as EVOPS. An output region mask is considered correct if it has a sufficiently large overlapping area with the ground truth (GT) mask. However, for incorrect outputs, segmentation tasks involve multiple types of errors, necessitating a more detailed categorization for accurate evaluation. Specifically, the outputs can be classified into one correct status and four error statuses, as follows:

- **Correct**: Given $\mathbf{M}_i \in \mathbb{M}_r$, $\exists \mathbf{M}_j \in \mathbb{M}_{gt}$ such that $a_{i,j}/a_i \geq 0.8$ and $a_{i,j}/a_j \geq 0.8$.
- **Over-segmented**: Given $\mathbf{M}_i \in \mathbb{M}_{gt}$, $\exists \mathbb{M}_s \subseteq \mathbb{M}_r$ such that $\forall \mathbf{M}_j \in \mathbb{M}_s$, $a_{i,j} > 0$ and $|\mathbb{M}_s| \geq 2$.
- **Missed**: Given $\mathbf{M}_i \in \mathbb{M}_{gt}$, $\forall \mathbf{M}_j \in \mathbb{M}_r$, $a_{i,j}/a_i < 0.8$ or $a_{i,j}/a_j < 0.8$.
- **Under-segmented**: Given $\mathbf{M}_i \in \mathbb{M}_r$, $\exists \mathbb{M}_s \subseteq \mathbb{M}_{gt}$ such that $\forall \mathbf{M}_j \in \mathbb{M}_s$, $a_{i,j} > 0$ and $|\mathbb{M}_s| \geq 2$.
- **Noise**: Given $\mathbf{M}_i \in \mathbb{M}_r$, $\forall \mathbf{M}_j \in \mathbb{M}_{gt}$, $a_{i,j}/a_i < 0.8$ or $a_{i,j}/a_j < 0.8$.

Here, $\mathbb{M}_r, \mathbb{M}_{gt}$ denote the sets of output and GT masks. a_i, a_j denote the area of mask $\mathbf{M}_i$ and $\mathbf{M}_j$. $a_{i,j}$ denotes the overlap area between $\mathbf{M}_i$ and $\mathbf{M}_j$.

The metrics for a single frame are calculated based on the statistics of these status categories and specifically include the following indicators:

- **Precision**: The ratio of correct regions to $\mathbb{M}_r$.
- **Recall**: The ratio of correct regions to $\mathbb{M}_{gt}$.
- **Intersection over Union (IoU)**: The average IoU of correct regions in a frame.
- **Dice Coefficient (DICE)**: The average DICE score of correct regions in a frame.
- **Over-segmentation Rate (OSR)**: The ratio of over-segmented regions to $\mathbb{M}_{gt}$.
- **Missed**: The ratio of missed regions to $\mathbb{M}_{gt}$.
- **Under-segmentation Rate (USR)**: The ratio of under-segmented regions to $\mathbb{M}_r$.
- **Noise**: The ratio of noise regions to $\mathbb{M}_r$.

B. Overall Experimental Results

We compare our method with CAPE[10] and DDPFF[8] because they specifically target depth image processing, which aligns perfectly with our approach. In contrast, the other methods [20], [21], [23], and [24] focus on general 3D point cloud processing. Additionally, for learning-based methods, we include comparisons with PlaneRecNet[31] and SAM-based[11] approaches. All the methods are evaluated on an Intel i7-12700K CPU with 32GB RAM, and the implementations are single-threaded. PlaneRecNet and SAM are evaluated on a Nvidia RTX 4080S. The text prompts used in GroundingDINO[32] inclue “wall”, “ground”, “desk”, “sofa”, “television”, “book”, “ceiling”.

Additionally, the reported runtime for all methods excludes the time required for reading. Thus the evaluation focuses mainly on the algorithmic processing time. Memory usage (RAM for trational methods and VRAM for learning-based methods) is tested on a subset of TUM dataset (660 frames). For CAPE and DDPFF, the optimal parameters are used based on the recommendation in [7]. For CAPE and our method, the minimal patch size is set to 10.

Table I shows a comprehensive comparison of different methods across various performance metrics, with example outputs shown in Fig. 5. Our method, APRG, demonstrates superior performance across several key metrics. Specifically, APRG achieves the highest precision, recall, IoU and DICE scores on both synthetic and real-world datasets.

Notably, SAM, without any prior knowledge, achieves similar segmentation performance as DDPFF. When combined

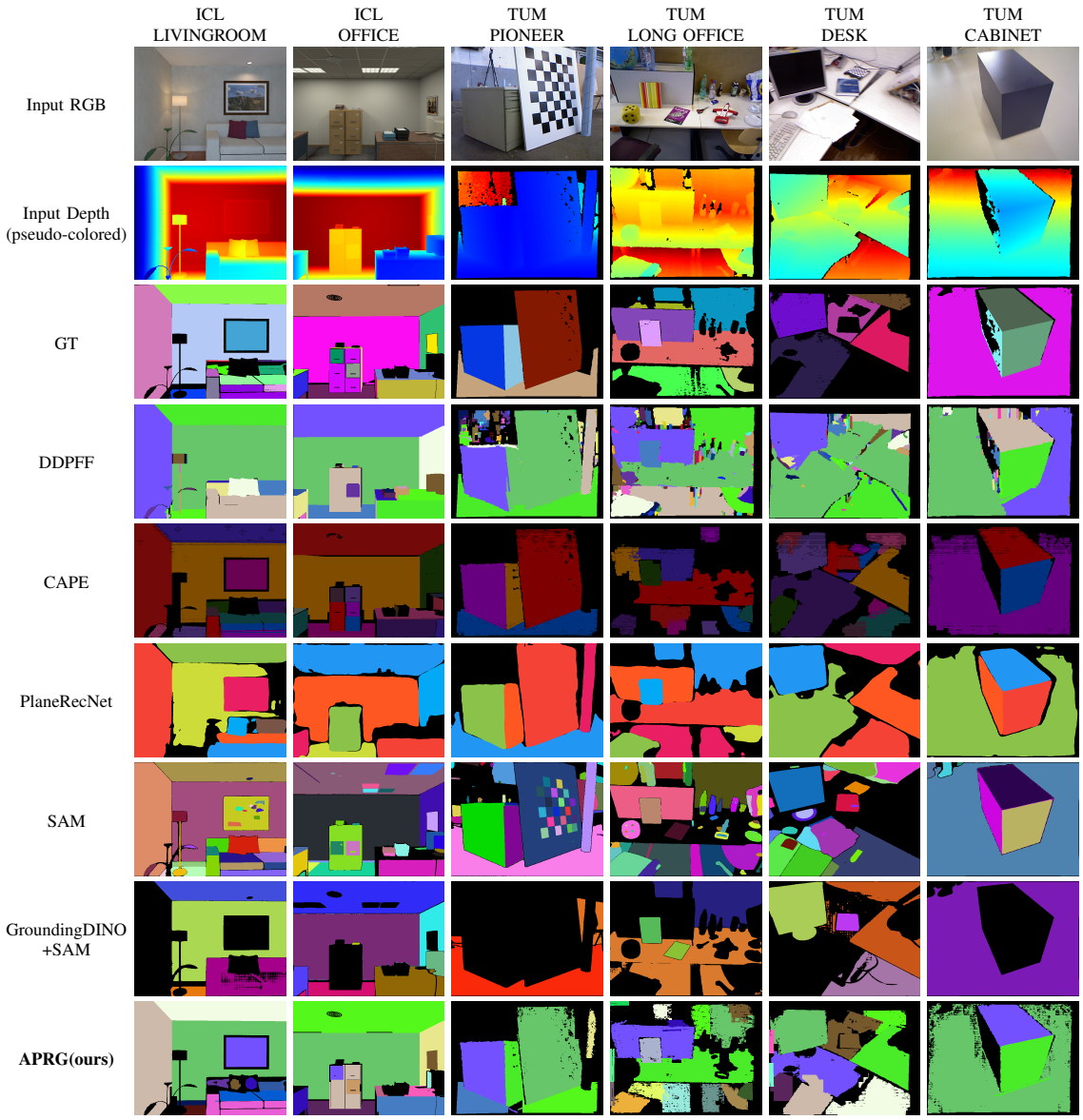


Fig. 5. Example results of evaluated methods. Each row represents one method. Each column represents one data sequence.

with GroundingDINO, both segmentation accuracy and computational efficiency see significant improvement.

In terms of computational efficiency, APRG stands out with a significantly higher processing speed, achieving more than 600 frames per second on both dateset, which is substantially faster than the other methods. From the comparison with three region-growing-based methods, APRG is at least 1.46x faster than the fix-sized region growing method CAPE and at least 14x faster than the pixel-level region growing method DDPFF. This highlights the advantages of our variable-sized region growing algorithm.

Even with GPU acceleration, learning-based methods are significantly slower than traditional methods running on CPU. This emphasizes the efficiency of traditional approaches, which, while less versatile than deep learning models, offer substantial advantages in terms of computational speed.

Overall, the results indicate that APRG not only excels in accuracy metrics but also achieves exceptional processing speed, making it a highly efficient and effective solution for plane segmentation tasks compared to other methods

C. Detailed Comparison with CAPE

Our method is based on variable-sized patch-wise region growing, which is quite similar to CAPE that uses fixed-sized patch-wise region growing. The minimum size of the patches directly determines the number of plane patches involved in plane patch segmentation and region growing, thereby affecting the runtime. To analyze the difference between the two methods, we set the minimum patch size to various sizes ranging from 4 to 15 (padding is used when necessary). We record the time taken for each step of both methods and plot the relationship between time, segmentation performance, and minimum patch size. The results are shown in Fig. 6, where T_1 represents the time taken for data rearrangement or building the data pyramid in both methods, T_2 represents the time of the plane patch segmentation step, T_3 represents the time of

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

TABLE I
OVERALL COMPARISON OF DIFFERENT METHODS

	Method	Precision	Recall	USR	OSR	Missed	Noise	IoU	DICE	Time(fps)	Mem(MB)
ICL	DDPFF	0.27	0.39	0.07	0.01	0.46	0.65	0.88	0.92	36.19	-
	CAPE	0.59	0.49	0.08	0.03	0.39	0.26	0.93	0.96	370.51	
	PlaneRecNet	**0.66**	0.44	0.07	0.01	0.49	0.26	0.91	0.95	38.25	
	SAM	0.16	0.43	**0.04**	0.07	0.35	0.76	0.92	0.96	0.68	
	GroundingDINO+SAM	0.60	0.35	0.06	**0.00**	0.60	0.34	**0.95**	0.97	2.30	
	APRG(ours)	**0.66**	**0.59**	0.06	0.02	**0.30**	**0.25**	**0.95**	**0.98**	**620.05**	
TUM	DDPFF	0.03	0.51	**0.00**	0.08	0.38	0.96	0.71	0.77	48.97	**32**
	CAPE	0.28	0.30	**0.00**	0.06	0.64	0.63	0.54	0.58	495.98	119
	PlaneRecNet	0.40	0.48	0.01	0.01	0.50	0.58	0.66	0.72	29.04	4183
	SAM	0.09	0.43	0.01	0.07	0.47	0.89	0.64	0.69	0.71	6595
	GroundingDINO+SAM	0.30	0.36	0.06	**0.00**	0.57	0.63	0.58	0.63	1.79	15207
	APRG(ours)	**0.54**	**0.61**	**0.00**	0.07	**0.31**	**0.38**	**0.85**	**0.90**	**724.69**	62

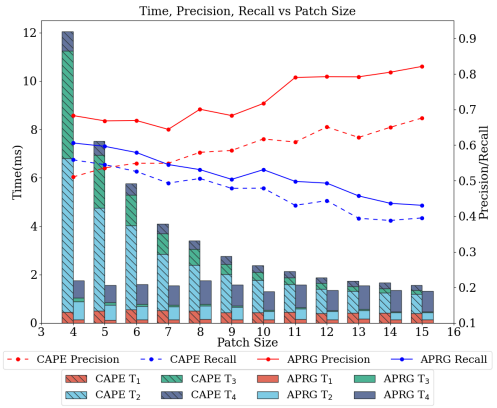


Fig. 6. Performance comparison between CAPE and APRG for different patch size in ICL dataset.

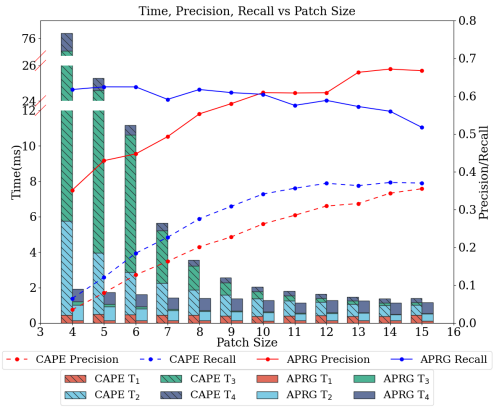


Fig. 7. Performance comparison between CAPE and APRG for different patch size on the TUM dataset.

the patch-wise region growing step, and T_4 represents the time of the boundary refinement step.

Fig. 6 shows the results on the ICL dataset. It can be seen that, as the patch size decreases, the amount of patches grows, thus T_2 and T_3 for CAPE exhibit substantial growth. The total time consumption increases six times as the patch size decreases from 15 to 4. In contrast, as APRG can abort further division in large planar regions when parent patch is extracted, the time for APRG remains almost constant. Regarding segmentation performance, as the patch size decreases, the precision and recall of both methods tend to balance out, with only minor difference in their values.

On the TUM dataset, as shown in Fig. 7, APRG maintains a significant time advantage over CAPE. However, there is a significant difference in segmentation performance between the two methods. When the patch size is less than 10, CAPE's precision and recall drop significantly, falling to around 0.1. Although APRG also experiences a decrease, its precision and recall remain between 0.3 and 0.6. This discrepancy is attributed to noise interference in real-world data. In CAPE, smaller patches make it more susceptible to noise, making it difficult to accurately fit planar patches. Conversely, in APRG, the plane patches are segmented from large to small, thereby avoiding the need for finer segmentation that is more prone to noise.

These results highlight the robust performance and efficiency of our variable-sized patch-wise region growing method, particularly in handling noisy real data, and demonstrate its advantage over fixed-sized patch-wise region growing methods.

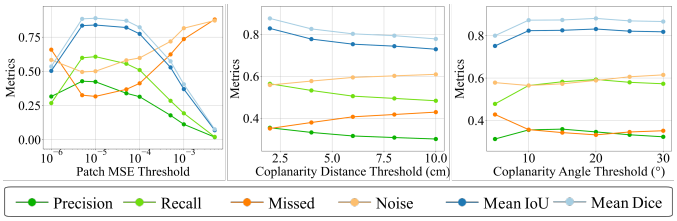


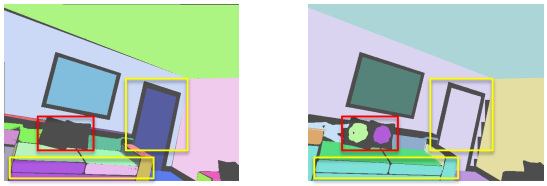
Fig. 8. Effect of three key parameters on algorithm performance.

D. Parameter Sensitivity Analysis

We use a subset (660 frames) of the TUM dataset to perform sensitivity analysis on the key parameters of our method. Specifically, we examine the plane patch MSE threshold, patch coplanarity distance threshold, and patch coplanarity angle threshold to evaluate their impact on segmentation performance.

The results presented in Fig. 8 show that the plane patch MSE threshold plays a crucial role in determining segmentation performance. This threshold should be carefully set based on the noise characteristics of the sensor. Once the plane patches are identified, the coplanarity angle and distance thresholds have only a minimal impact on the overall segmentation performance.

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.



(a) GT (b) Ours

Fig. 9. The red box highlights adjacent planar regions that are assigned different categories in the ground truth (GT), but are grouped into the same category in our output. The yellow box indicates bulging surfaces that are not classified as planes in the GT but are identified as planes in our output.

E. Edge Cases and Limitations

Our method is based purely on geometric information, which leads to certain limitations in specific cases. When two adjacent regions are co-planar, it may fail to differentiate them (yellow box in Fig. 9). Additionally, for approximately planar curved surfaces, our method misclassifies them as planes due to a lack of consideration for global object properties (red box in Fig. 9). These cases highlight the challenges of relying solely on geometric cues for segmentation. If higher segmentation quality is required for subsequent tasks, incorporating RGB image information could help improve the results.

V. Conclusion

This paper presents an efficient plane segmentation method for depth images. Our approach leverages the inherent structure of depth images and performs region growing with variable-sized patches. This design reduces redundancy by avoiding exhaustive traversal of small patches in large planar areas. Additionally, our top-down segmentation prevents unnecessary subdivisions, enhancing robustness against noise. Experimental results show that our method outperforms state-of-the-art approaches in both efficiency and segmentation quality. In future work, we aim to integrate our method into robotic manipulation tasks and explore its benefits for improving 6D pose estimation.

References

[1] R. F. Salas-Moreno, B. Glocken, P. H. Kelly, and A. J. Davison, "Dense planar slam," in *IEEE Int. Symp. Mixed Augmented Reality*, 2014, pp. 157–164.

[2] S. Kannapiran, J. van Baar, and S. Berman, "A visual inertial odometry framework for 3d points, lines and planes," in *IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 9206–9211.

[3] T. T. Pham, M. Eich, I. Reid, and G. Wyeth, "Geometrically consistent plane extraction for dense indoor 3d maps segmentation," in *IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2016, pp. 4199–4204.

[4] C. Yuan, X. Liu, X. Hong, and F. Zhang, "Pixel-level extrinsic self calibration of high resolution lidar and camera in targetless environments," *IEEE Robot. Automat. Lett.*, vol. 6, no. 4, pp. 7517–7524, 2021.

[5] W. Forstner and K. Khoshelham, "Efficient and accurate registration of point clouds with plane to plane correspondences," in *Proc. IEEE Int. Conf. Comp. Vis. Workshops*, 2017, pp. 2165–2173.

[6] Y. Shi, J. Huang, X. Xu, Y. Zhang, and K. Xu, "Stablepose: Learning 6d object poses from geometrically stable patches," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, June 2021, pp. 15 222–15 231.

[7] A. Kornilova, D. Iarosh, D. Kukushkin, N. Goncharov, P. Mokeev, and A. Saliou, "Evops benchmark: evaluation of plane segmentation from rgbd and lidar data," in *IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2022, pp. 13 074–13 080.

[8] A. Roychoudhury, M. Missura, and M. Bennewitz, "Plane segmentation in organized point clouds using flood fill," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2021, pp. 13 532–13 538.

[9] E. Vera, D. Lucio, L. A. Fernandes, and L. Velho, "Hough transform for real-time plane detection in depth images," *Pattern Recognit. Letters*, vol. 103, pp. 8–15, 2018.

[10] P. F. Proença and Y. Gao, "Fast cylinder and plane extraction from depth cameras for visual odometry," in *IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 6813–6820.

[11] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson *et al.*, "Segment anything," in *Proc. IEEE Int. Conf. Comp. Vis.*, 2023, pp. 4015–4026.

[12] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Commun. of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[13] L. Li, F. Yang, H. Zhu, D. Li, Y. Li, and L. Tang, "An improved ransac for 3d point cloud plane segmentation based on normal distribution transformation cells," *Remote Sens.*, vol. 9, no. 5, p. 433, 2017.

[14] M. Alehdaghi, M. A. Esfahani, and A. Harati, "Parallel ransac: Speeding up plane extraction in rgbd image sequences using gpu," in *Int. Conf. Comput. Knowl. Eng.*, 2015, pp. 295–300.

[15] R. Schnabel, R. Wahl, and R. Klein, "Efficient ransac for point-cloud shape detection," in *Comput. Graph. Forum*, vol. 26, 2007, pp. 214–226.

[16] P. V. Hough, "Method and means for recognizing complex patterns," Dec. 18 1962, uS Patent 3,069,654.

[17] P. Mukhopadhyay and B. B. Chaudhuri, "A survey of hough transform," *Pattern Recognit.*, vol. 48, no. 3, pp. 993–1010, 2015.

[18] H. D. Khanh, V. S. Nguyen, A. Do, and N. T. Dzung, "An effective randomized hough transform method to extract ground plane from kinect point cloud," in *Int. Conf. Inf. Commun. Technol. Convergence*, 2019, pp. 1053–1058.

[19] Y. Tian, W. Song, L. Chen, Y. Sung, J. Kwak, and S. Sun, "Fast planar detection system using a gpu-based 3d hough transform for lidar point clouds," *Appl. Sciences*, vol. 10, no. 5, p. 1744, 2020.

[20] J. Weingarten, G. Gruener, and R. Siegwart, "A fast and robust 3d feature extraction algorithm for structured environment reconstruction," in *Proc. 11th Int. Conf. Adv. Robot.*, vol. 3, 2003, pp. 1142–1147.

[21] T. Czerniawski, B. Sankaran, M. Nahangi, C. Haas, and F. Leite, "6d dbscan-based segmentation of building point clouds for planar object classification," *Automat. Construction*, vol. 88, pp. 44–58, 2018.

[22] A.-V. Vo, L. Truong-Hong, D. F. Laefer, and M. Bertolotto, "Octree-based region growing for point cloud segmentation," *ISPRS J. Photogrammetry Remote Sens.*, vol. 104, pp. 88–100, 2015.

[23] X. Du, Y. Lu, and Q. Chen, "A fast multiplane segmentation algorithm for sparse 3-d lidar point clouds by line segment grouping," *IEEE Trans. Instrum. Meas.*, vol. 72, pp. 1–15, 2023.

[24] S. Xu, R. Wang, H. Wang, and R. Yang, "Plane segmentation based on the optimal-vector-field in lidar point clouds," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 11, pp. 3991–4007, 2021.

[25] C. Liu, J. Yang, D. Ceylan, E. Yumer, and Y. Furukawa, "Planenet: Piece-wise planar reconstruction from a single rgb image," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2018, pp. 2579–2588.

[26] C. Liu, K. Kim, J. Gu, Y. Furukawa, and J. Kautz, "Planercnn: 3d plane detection and reconstruction from a single image," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019, pp. 4450–4459.

[27] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proc. IEEE Int. Conf. Comp. Vis.*, 2017, pp. 2961–2969.

[28] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2012, pp. 573–580.

[29] A. Handa, T. Whelan, J. McDonald, and A. J. Davison, "A benchmark for rgb-d visual odometry, 3d reconstruction and slam," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2014, pp. 1524–1531.

[30] T. Mallick, P. P. Das, and A. K. Majumdar, "Characterizations of noise in kinect depth images: A review," *IEEE Sensors Journal*, vol. 14, no. 6, pp. 1731–1740, 2014.

[31] Y. Xie, F. Shu, J. Rambach, A. Pagani, and D. Stricker, "Planerecnet: Multi-task learning with cross-task consistency for piece-wise plane detection and reconstruction from a single rgb image," 2021.

[32] S. Liu, Z. Zeng, T. Ren, F. Li, H. Zhang, J. Yang *et al.*, "Grounding dino: Marrying dino with grounded pre-training for open-set object detection," in *European Conference on Computer Vision*. Springer, 2024, pp. 38–55.