IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 19 June 2024, accepted 9 July 2024, date of publication 12 July 2024, date of current version 23 July 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3427378

RESEARCH ARTICLE

Model-Based Segmentation-Supported Camera Tracking in Fab's Indoor Environments

JEONGHYEON AHN, JUNGHO HA, JAEMIN SON, AND JUNGHYUN HAN, (Member, IEEE)
Department of Computer Science and Engineering, Korea University, Seoul 02841, South Korea

Corresponding author: Junghyun Han (jhan@korea.ac.kr)

This work was supported in part by the Ministry of Science and ICT, South Korea, through the ICT Creative Consilience Program, under Grant IITP-2024-2020-0-01819; and in part by the Information Technology Research Center (ITRC) Support Program under Grant IITP-2024-2020-0-01460 and Grant 2020-0-00861.

ABSTRACT Currently, it is a norm to design a semiconductor fab using building information models (BIMs), which refer to a digital representation of a building's physical and functional characteristics. The comprehensive data provided by BIMs include 3D geometric models. This paper presents a 3D model-based camera tracking method, which is targeted at navigating a fab's wide indoor environment. The key observation made in designing the method is that there are a number of fixed objects in such an indoor environment. The columns are the representative among them. Our method extracts the columns from the input image and matches them to their BIMs to estimate the camera pose. The estimation accuracy is significantly increased by adopting an instance segmentation network. It is trained with a dataset, which is extracted from the target indoor environment and processed by our own data engine. The test results show that our tracking method is drift-free, accurate and robust. We envision that it can be used in many applications such as AR-based visual inspection.

INDEX TERMS Augmented reality, building information modeling, camera tracking, instance segmentation.

I. INTRODUCTION

A semiconductor fabrication plant, which is commonly abbreviated to a *fab*, refers to a factory where integrated circuits are manufactured. In the semiconductor industry, it is currently a norm to design a fab using *building information models* (BIMs). They refer to a digital representation of a building's physical and functional characteristics, and the comprehensive data provided by BIMs include 3D geometric models.

In many domains of industry, augmented reality (AR) has been explored for *visual inspection*. In the mechanical domain, for example, the video capturing a mechanical part is overlaid with its computer-aided design (CAD) model to see if there exists discrepancy between them.

Throughout the life cycle of a fab, inspection is a crucial activity of checking if all works progress as planned. In the construction phase, for example, inspection is regularly carried out to verify compliance with the design. BIMs visualized via AR help for this purpose [1], [2]. Shown in Fig. 1-(a) is our laboratory, where two pipes are being assembled between columns, and Fig. 1-(b) shows the pipe and column BIMs of the original design in wireframe drawing. In order to see if the pipes under construction are compatible with the design, the pipe BIMs are rendered on top of the captured image, as shown in Fig. 1-(c), revealing that the left pipe is incomplete, the middle one is complete, and the right one is not constructed at all.

In such an AR-based visual inspection task, the key element is *camera tracking*. Unless the camera pose is precisely estimated, the BIMs are not correctly overlaid onto the physical objects. It is especially challenging in the semiconductor industry. As shown in Fig. 2-(a), the size of a fab is huge. A fab is composed of multiple production lines, and each line typically occupies an area ranging from $10,000m^2$ to $20,000m^2$. In such a wide environment, a naive

The associate editor coordinating the review of this manuscript and approving it for publication was Dominik Strzalka.

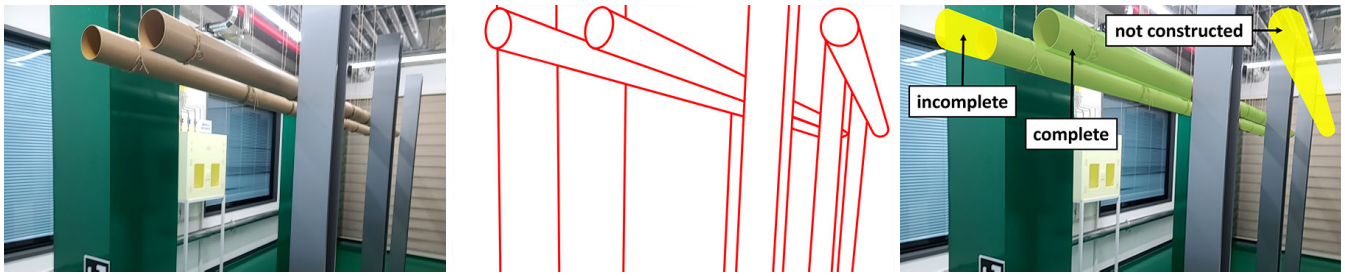

FIGURE 1. Visual inspection for checking the progress of pipe construction: (a) All experiments reported in this paper were made in this lab. Between the columns are pipes under construction. (b) BIMs of columns and pipes. (c) AR-based visual inspection.

FIGURE 2. Fabs and columns: (a) Aerial views of fabs. (b) Columns in the indoor environment of a fab under construction.

camera tracking method is likely to accumulate localization errors over time, which eventually makes it impossible to do visual inspection.

In this paper, we present a *model-based* camera tracking method, which is designed for navigating a wide-area fab environment. Our method extracts a set of "fixed" objects from the video capturing the environment and aligns them with the 3D models provided in their BIMs, so as to keep calibrating the possible drift of the camera trajectory. In the current implementation, our method uses the *columns*, which are everywhere in any fab, as shown in Fig. 2-(b); the 3D columns are extracted from the input RGB-D image and matched to their 3D models to estimate the camera pose.

The performance of our camera tracking method hinges on how accurately the columns are extracted. The required level of accuracy is ensured by an *instance segmentation network*, the training data for which are extracted from the target indoor environment and processed by our own easy-to-use *data engine*. The test results show that our tracking method is drift-free, accurate and robust enough for many critical inspection tasks in fab environments, such as AR-based visual inspection.

Our main contributions are listed as follows:

- We propose a *model-based camera tracking* method, which is best suited for navigating a wide-area fab environment. It is drift-free, accurate and robust, for example, enabling AR-based visual inspection.
- The method has broad applicability not only because it can be used with any "fixed" objects only if their 3D models are available, but also because it works well even in a textureless or poorly-textured environment.
- The tracking accuracy is significantly increased by involving an *instance segmentation network*. Given a novel environment, the network needs to be trained with the data extracted from the environment, but our *data engine* enables the training dataset to be generated easily and quickly.

II. RELATED WORK

Simultaneous localization and mapping (SLAM) methods built upon point features, such as ORB-SLAM [3], are prone to fail in textureless or poorly textured environments. In addition, point features hardly encode geometric information and may not provide effective constraints, often leading to large errors accumulated over time. Incorporating additional depth sensors, previous studies, including ORB-SLAM2 [4], BundleFusion [5], and BAD-SLAM [6], have explored the field of RGB-D SLAM [7]. However, they have also failed to address accumulated errors due to the inaccuracies inherent in depth sensors. As an alternative, lines have attracted attention. Smith et al. [8] proposed a system based on non-structural lines, but the lines may be incompletely detected and partially occluded, making the system unstable. Some other studies, such as PL-SLAM [9], proposed to combine points and lines to make the SLAM system more applicable in challenging scenes. Wei et al. [10] proposed a visual-inertial system based on points and lines to enhance tracking accuracy with IMU sensors. Their system demonstrated improved stability in

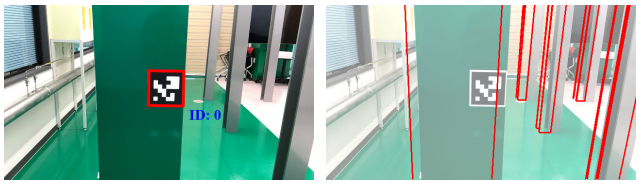

FIGURE 3. Columns and their BIMs: (a) A large column (in the middle) and four small columns (at the right). In the fab environment, a marker is attached to each large column. (b) BIMs projected onto the input image are visualized in wireframes.

low-textured environments but failed to overcome completely the inherent drawback of using non-structural features.

Zhou et al. [11] proposed a system based on structural lines of buildings. Their system can alleviate the accumulating orientation errors, but its robustness is affected when auxiliary information from other sensors is not available, such as wheel odometer. Taking the geometric information of orthogonality, parallelism and co-planarity into consideration, Li et al. [12] proposed an optimization strategy, which helps refine the estimation results made by PL-SLAM [9]. Under the Manhattan world assumption, several works decouple the rotational and translational motion estimation to obtain a drift-free rotation estimation. Li et al. [13] predicts surface normals by a convolutional neural network (CNN), and Liu and Meng [14] proposed a two-stage vanishing points estimation method to improve the localization performance. Pop-up SLAM [15] combined scene understanding with traditional visual SLAM to increase the performance of both state estimation and dense mapping especially in low-textured environments. It generates plane landmark measurements by employing a single image pop-up plane model. Plane-Edge-SLAM [16] introduced a system that combines plane and edge features for use in indoor environments. Contour-SLAM [17] proposed an object-level SLAM approach that aligns contours between the object models and instance masks.

Due to the proliferation and wide adoption of BIMs in the semiconductor industry, BIM-based localization studies have recently been made. Park et al. [18] introduced a method for real-time tracking that leverages Bluetooth, motion sensors, and BIMs. Asadi et al. [19] presented a real-time localization method built upon ORB-SLAM2 [4]. It registers a video sequence to a BIM via perspective detection and matching between video frames and their corresponding BIM views. Acharya et al. [20] introduced a BIM-based indoor localization method, where the BIM edges visible from a mobile device's camera are sampled and projected onto the image plane so that their correspondences with the structural edges in the image are established. Then, the camera pose is estimated by minimizing the distances between the projected points and the image points. Moura et al. [21] introduced a BIM interface that integrates 3D building data into the pose graph of a SLAM system operated by a mobile robot. Chen et al. [22] proposed a method that utilizes a BIM as a reference to rectify and fine-tune coarse camera poses estimated by photogrammetry. It is achieved by aligning a photogrammetric point cloud with a BIM-referenced point cloud. Yin et al. [23] presented a method to localize a mobile 3D LiDAR sensor within a BIM-generated map. The BIM is converted into a point cloud map, and the map points are labeled with the categories from the BIM. Then, the LiDAR 3D points are aligned with the point cloud map. There have been attempts to integrate BIM-based localization methods with deep learning. Acharya et al. [24] proposed a solution to regress the camera poses by fine-tuning a pre-trained CNN using synthetic images generated from a BIM. Chen et al. [25] introduced a method to estimate indoor camera poses using a 3D style-transferred BIM and photogrammetric techniques. Zhao and Cheah [26] introduced a BIM-based construction automation system for initial robot localization combined with real-time object detection using CNN.

Recently, research works have been reported on AR-based visual inspection [27]. Meža et al. [28] demonstrated an implementation of a BIM-based AR system for construction using component-based software engineering methods. Choi et al. [29] proposed an AR platform designed to support workers in manufacturing and inspection of offshore structures. This platform estimates the positions of workers in real-time and accurately determines the poses of their mobile devices by aligning 3D CAD models with actual correspondences. Tsai et al. [30] proposed a method for on-site pipeline inspection and automatic coordination, which combines AR with a path planning algorithm. VisualLive [31] provides state-of-the-art AR-based visual inspection. However, it does not address the drift issue, making it difficult to conduct precise visual inspection in large environments.

III. OVERVIEW

In the semiconductor industry, two types of columns are widely used, which we simply call *large* and *small*. The large column is made of concrete, and the small one is a steel H-beam. Fig. 3-(a) shows a large column (in dark green) and four small ones (in gray). In a real fab, the large columns are far apart from each other, and the fab's floor plan is partitioned into a grid of rectangular cells such that a cell is centered on a large column. As shown in Fig. 3-(a), we attach a marker to each large column. It stores the cell's ID.

The indoor environment is scanned by a mobile device, iPad Pro 6th generation in the current implementation, and its camera pose is tracked every frame according to the steps presented in Fig. 4. Only for the first frame, we detect the marker attached to an arbitrary large column. The grid cell centered on the large column is identified and the BIMs of all columns inhabiting the cell are loaded.

Simultaneously, the camera pose is estimated via homography and is used to transform the loaded BIMs into the *camera space*. Fig. 3-(b) visualizes the BIMs projected onto the input image. The "BIM columns" are not correctly registered with the "scene columns" because the *initial guess* of the

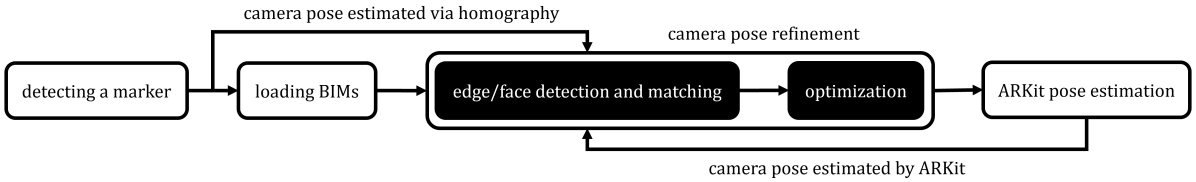

FIGURE 4. Camera tracking using BIMs: Every frame, the initial guess of the camera pose is refined by the "camera pose refinement" step. For the first frame, the initial guess is made via a single marker detection followed by homography. From the second frame, it is provided by ARKit.

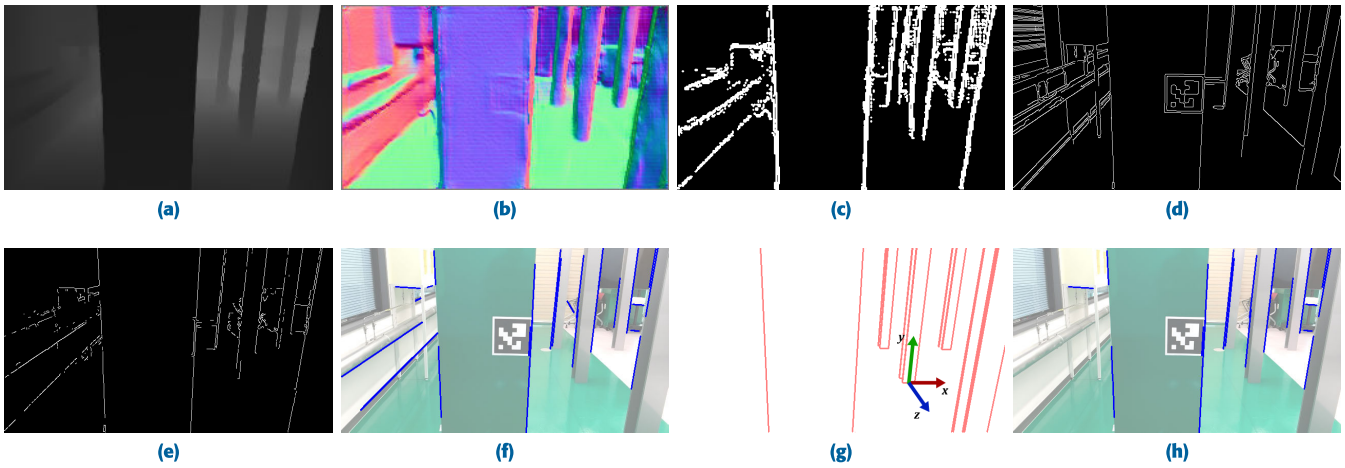

FIGURE 5. Scene edge detection: (a) Depth map. (b) Normal map. (c) Edges detected from the depth and normal maps. (d) Edges detected from the RGB image. (e) Combination of (c) and (d). (f) The edges obtained via probabilistic Hough line transform are visualized in blue. (g) BIM edges and Manhattan axes. (h) Detected scene edges.

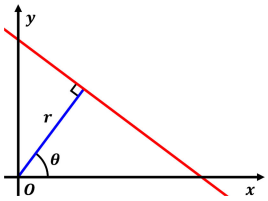

FIGURE 6. Hough line transform.

camera pose is not accurate. It is refined by the "camera pose refinement" step shown in Fig. 4. From the second frame, the initial guess is provided by ARKit (see the last box in Fig. 4) and is then also refined. This loop is repeated.

In our study, we have developed two versions of *model-based tracker*, which we name V1 and V2. They differ in the "camera pose refinement" step, for which V1 uses traditional computer vision techniques whereas V2 integrates *instance segmentation* into the framework of V1. Overall, V2 performs far better than V1, and most of our contributions lie in V2. However, the training dataset for V2's instance segmentation is generated using V1.

IV. V1 = BIM-BASED TRACKING

The "camera pose refinement" step in Fig. 4 is composed of two sub-steps: "edge/face detection and matching" and "optimization." They are presented in the following two subsections.

A. EDGE/FACE DETECTION AND MATCHING

For SLAM in textureless environments, leveraging geometric features such as edges has proven to be effective [15], [32]. Our tracking method relies heavily on the columns' edges. They are detected from the input image and then matched to the corresponding edges in the BIMs. We call the former "scene edges" and the latter "BIM edges."

1) EDGE DETECTION AND MATCHING

Using the LiDAR sensor of iPad Pro, ARKit generates a *depth map* every frame (Fig. 5-(a)). As done in Kinect-Fusion [33], we generate a *normal map* using it, i.e., after bilateral-filtering the depth map, the depth values are back-projected into 3D space to generate a 3D *vertex map*, and for each vertex position, a normal is computed using the cross-product operation with neighboring vertices. Fig. 5-(b) visualizes the normal map.

Both the depth and normal maps are Laplacian-filtered, and the results are combined with the pixelwise OR operation to produce the edges shown in Fig. 5-(c). Simultaneously, Canny edge detector [34] is applied to the input RGB image to produce the edges shown in Fig. 5-(d). The two edge images are combined with the pixelwise AND operation (Fig. 5-(e)),

and the probabilistic Hough line transform [35] is applied to the combined to generate the blue-colored edges in Fig. 5-(f), where an edge is defined by its two end points.

In terms of Hough line transform, each edge is associated with (r, θ), as depicted in Fig. 6, where r is the distance from the origin to the closest point on the line aligned with the edge, and θ is the angle between the x-axis and the line connecting the origin with the closest point.

Among the edges in Fig. 5-(f), we are interested only in those which belong to columns. The others are *outliers*. In order to discard as many outliers as possible, we resort to the *Manhattan world assumption* [36]; each geometric entity is assumed to align to one of the Cartesian coordinate system's axes, named *Manhattan axes*. In the BIMs of a fab, this is almost always the case if we consider only the columns. In Fig. 5-(g), it can be found that all BIM edges are aligned with the Manhattan axes.

Among the BIM edges, consider the vertical ones, i.e., the edges aligned with the y-axis. When a vertical edge is projected into the image plane, its θ is calculated. When all vertical edges are processed, we obtain a θ-range denoted as $[\theta_1^y, \theta_2^y]$. Similarly, $[\theta_1^x, \theta_2^x]$ and $[\theta_1^z, \theta_2^z]$ are obtained by projecting the BIM edges aligned with the x- and z-axes, respectively.

If a blue edge in Fig. 5-(f) belongs to none of the three ranges, it is discarded. The remaining edges are taken as "scene edges" (Fig. 5-(h)). Even though the outliers are not completely removed, their count is significantly reduced, leading to more robust edge matching.

Fig. 7-(a) visualizes the detected "scene edges" in blue and the projected "BIM edges" in red. Given a scene edge, the midpoint between the end points is computed, and then its depth is retrieved from the depth map. For a BIM edge, it is straightforward to compute the midpoint depth. Matching between the scene and BIM edges is made using a k-dimensional tree (k-d tree) [37], for which we use r, θ and the midpoint depth. In Fig. 7-(a), the matched edges are linked using yellow line segments. Note that multiple scene edges can be matched to a single BIM edge.

2) FACE DETECTION AND MATCHING

In order to extract the columns' *faces* and also the fab's *floor*, we scan the environment with ARKit's *plane detection* enabled. Then, a set of rectangular planes is detected, as shown in Fig. 7-(b), where the floor's plane is not visualized in order to avoid cluttering. Each detected plane is associated with the information of surface normal, 3D center position, etc.

Outlier planes are discarded using the Manhattan world assumption. We call the remaining "scene faces." The floor is indispensable for tracking and is included in the scene faces. From the BIMs, the columns' faces and the floor are extracted. They are named "BIM faces." For matching between the scene and BIM faces, we use a k-d tree with the 3D center position.

B. OPTIMIZATION

Given a set of matched pairs, i.e., both edge pairs and face pairs, camera pose estimation is defined as the task of finding the optimal rigid motion, $(\mathbf{r}_{\text{cam}}, \mathbf{t}_{\text{cam}})$, where $\mathbf{r}_{\text{cam}}$ is the 3D vector for the axis-angle representation of rotation, and $\mathbf{t}_{\text{cam}}$ is the 3D translation vector. For optimization, four error terms are defined: E_{edge}, E_{face}, E_{grav} and E_{pose}.

Let the BIM's geometric entities be indexed by i and the scene's by j. The i-th "BIM edge" projected onto the image plane is associated with a 2D line equation, $a_i x + b_i y + c_i = 0$. Let $\mathbf{l}_i$ denote the vector, (a_i, b_i, c_i). On the other hand, the matched "scene edge" is defined by two end points. Let $\dot{\mathbf{p}}_{j,1}$ and $\dot{\mathbf{p}}_{j,2}$ denote their homogeneous coordinates. The edge error, E_{edge}, is defined as follows:

$$E_{\text{edge}} = \sum_{i,j} \left(\left| \mathbf{l}_i \cdot \dot{\mathbf{p}}_{j,1} \right|^2 + \left| \mathbf{l}_i \cdot \dot{\mathbf{p}}_{j,2} \right|^2 \right) \tag{1}$$

Let $\mathbf{c}_i$ denote the center position of the i-th "BIM face." Let $\mathbf{c}_j$ and $\mathbf{n}_j$ respectively denote the center position and normal of the j-th "scene face." The face error, E_{face}, is defined as follows:

$$E_{\text{face}} = \sum_{i,j} \left| \left(\mathbf{c}_i - \mathbf{c}_j \right) \cdot \mathbf{n}_j \right|^2 \tag{2}$$

Let $\mathbf{g}_{\text{world}} = (0, -1, 0)^T$, $\mathbf{R}_{\text{cam}}$ denote the 3×3 matrix representation of $\mathbf{r}_{\text{cam}}$, and $\mathbf{g}_{\text{imu}}$ denote the gravity vector measured by the mobile device's IMU sensor. The gravity error, E_{grav}, is defined as follows:

$$E_{\text{grav}} = \left| 1 - \mathbf{g}_{\text{world}} \cdot \mathbf{R}_{\text{cam}} \mathbf{g}_{\text{imu}} \right|^2 \tag{3}$$

The optimal camera pose, $(\mathbf{r}_{\text{cam}}, \mathbf{t}_{\text{cam}})$, is a 6D vector. Let $(\mathbf{r}_{\text{A}}, \mathbf{t}_{\text{A}})$ denote the 6D vector that represents the camera pose returned by ARKit. Then, the camera pose error, E_{pose}, is defined as the squared L2 norm of their difference:

$$E_{\text{pose}} = \left\| (\mathbf{r}_{\text{cam}}, \mathbf{t}_{\text{cam}}) - (\mathbf{r}_{\text{A}}, \mathbf{t}_{\text{A}}) \right\|_2^2 \tag{4}$$

The objective or total error, E, is defined as the weighted sum of four error terms:

$$E = w_{\text{edge}} E_{\text{edge}} + w_{\text{face}} E_{\text{face}} + w_{\text{grav}} E_{\text{grav}} + w_{\text{pose}} E_{\text{pose}} \tag{5}$$

The optimization problem is solved with the Levenberg-Marquardt method inside a RANSAC (random sampling consensus) loop so as to remove incorrect matching, i.e., mismatched edge pairs and face pairs.

For RANSAC, we randomly select three pairs of matched entities, e.g., "an edge pair and two face pairs" or "three edge pairs." When selecting three pairs, we impose what we call *Manhattan axis constraint*. Fig. 7-(c) shows an example of three edge pairs, which do not satisfy the constraint: Because the edges are all along the y-axis, the y-coordinate of the camera position cannot be determined.

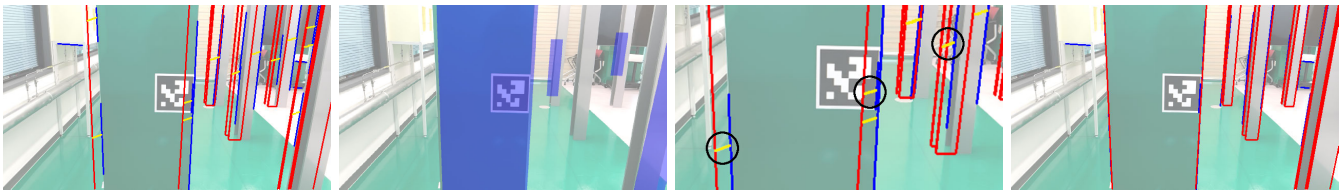

FIGURE 7. Camera pose refinement using edges and faces: (a) Scene edges (in blue) are matched with BIM edges (in red). (b) Planes detected by ARKit. (c) Three edge pairs marked with black circles do not satisfy what we call Manhattan axis constraint. (d) The BIM columns are *registered* to the scene columns due to optimization.

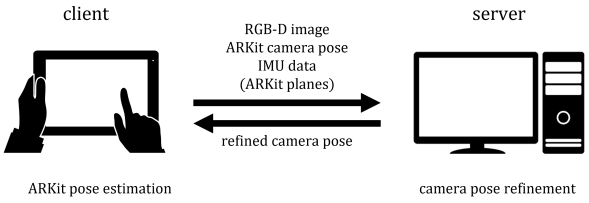

FIGURE 8. The server-client architecture: ARKit planes are used only in V1. In V2, they are not passed to the server.

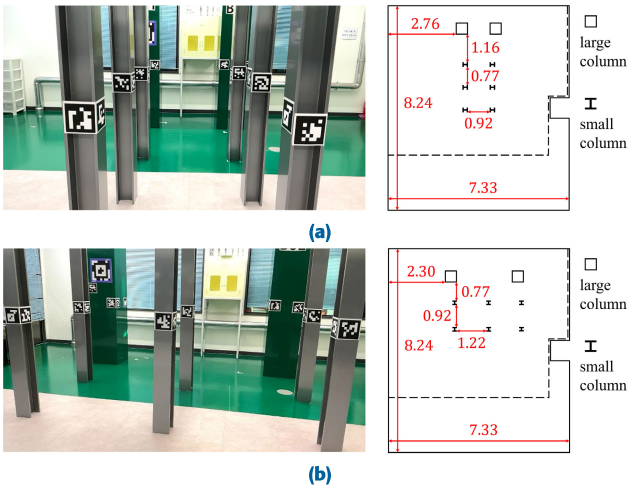

FIGURE 9. Each test scene contains two large columns and six small columns. In the floor plans, the dimensions are in meters. The large column appears to be a square with a side length of 0.45 meters; the bounding box of the small column is a square with a side length of 0.15 meters: (a) Scene 1. (b) Scene 2.

Solving the optimization problem, we obtain the refined camera pose, $(\mathbf{r}_{\text{cam}}, \mathbf{t}_{\text{cam}})$. Fig. 7-(d) shows that the BIM columns are correctly registered with the scene columns by the camera pose.

C. EXPERIMENTS

Our system is implemented in a server-client architecture. The client and the server are respectively in charge of "ARKit pose estimation" and "camera pose refinement" presented in Fig. 4. Fig. 8 shows the data transferred between them.

The server is a PC with Ryzen 7 5800X CPU (8 cores @ 3.8GHz), 32GB RAM and NVIDIA GeForce RTX 3090 GPU. The camera pose refinement algorithms are written in C++ with OpenCV, Eigen (for linear algebra computation) and Ceres Solver (generally for solving non-linear least squares problems; specifically for the Levenberg-Marquardt method).

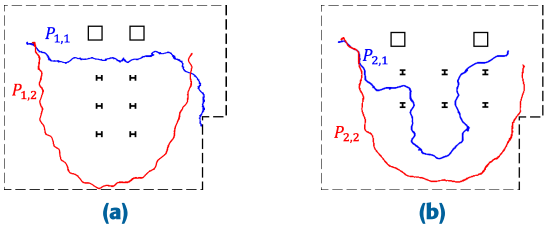

FIGURE 10. Groundtruth trajectories: (a) $P_{1,1}$ and $P_{1,2}$ in Scene 1. (b) $P_{2,1}$ and $P_{2,2}$ in Scene 2. It takes 64.97 seconds for the camera to move along $P_{1,1}$, 89.40 seconds along $P_{1,2}$, 86.17 seconds along $P_{2,1}$, and 73.00 seconds along $P_{2,2}$.

TABLE 1. Camera tracking methods.

	first frame	subsequent frames
ARKit ORB-SLAM3	multiple markers + EPnP	their own methods
V1	a single marker + homography & camera pose refinement	ARKit pose estimation & camera pose refinement
GT	multiple markers + EPnP	multiple markers + EPnP

1) TEST SCENES AND GROUNDTRUTH TRAJECTORIES

Our experiments are made in two scenes presented in Fig. 9-(a) and -(b), which we name Scene 1 and Scene 2, respectively. The area between the columns is made clear in order for the camera to move around freely, but various facilities surrounding them make it difficult to track the camera.

Just to make the *groundtruth* (henceforth, GT), we attach many markers to each column, and the 3D coordinates of all markers' corners are measured beforehand. In such a heavily marked environment, multiple markers can be captured by the camera wherever it is located. Then, the camera pose can be fairly accurately computed using the Efficient Perspective-n-Point (EPnP) algorithm [38]. Such accurately computed poses are taken as GT. Fig. 10 shows two GT trajectories per scene: $P_{1,1}$ and $P_{1,2}$ in Scene 1, and $P_{2,1}$ and $P_{2,2}$ in Scene 2.

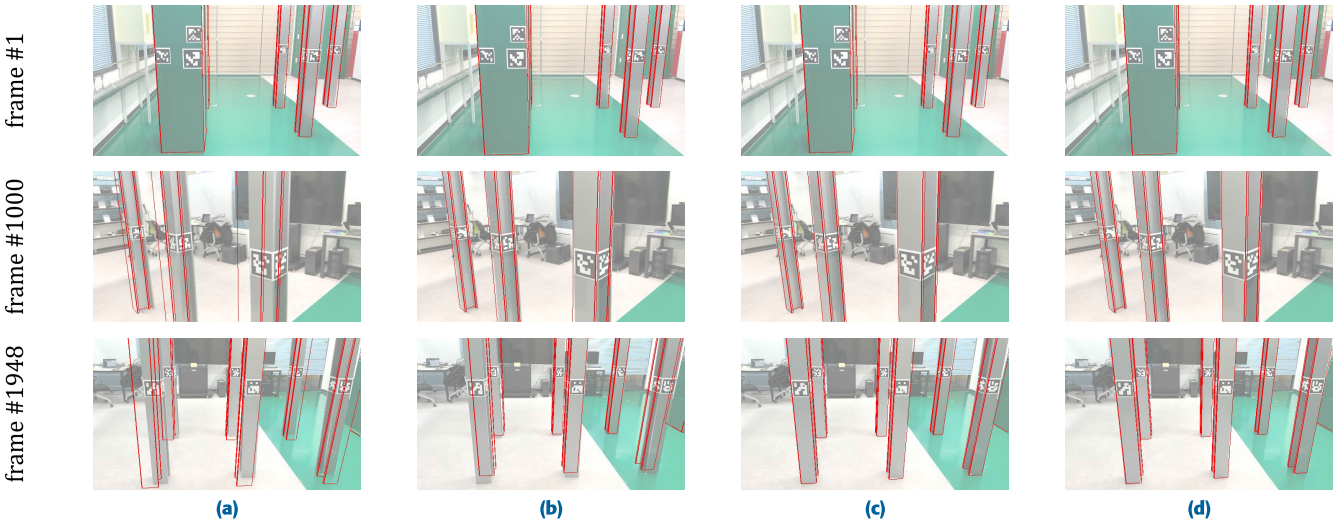

FIGURE 11. Snapshots extracted along $P_{1,1}$: (a) ARKit. (b) ORB-SLAM3. (c) V1. (d) Groundtruth.

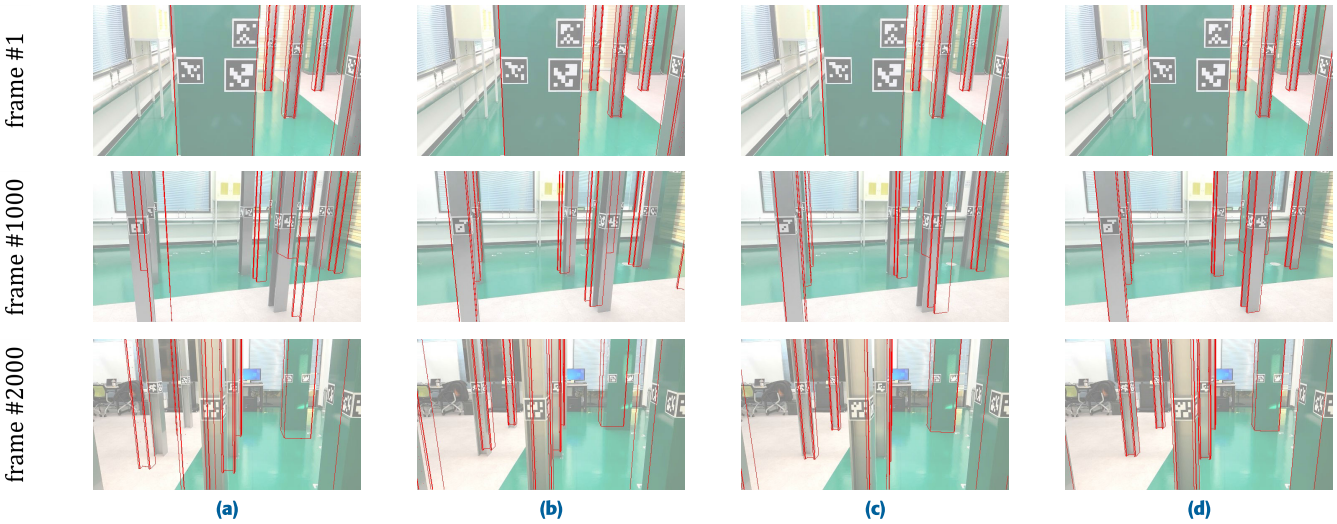

FIGURE 12. Snapshots extracted along $P_{2,2}$: (a) ARKit. (b) ORB-SLAM3. (c) V1. (d) Groundtruth.

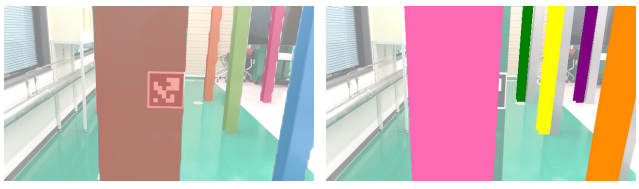

FIGURE 13. Column matching: (a) In this example, YOLACT++ outputs five "column masks," i.e., one for the large column and four for the small columns. (b) BIM columns are rendered to make "BIM masks." Due to the inaccurate camera pose, each "BIM mask" does not coincide with the corresponding "column mask," but they can be matched using the depth information.

2) TEST RESULTS

V1 is compared with two RGB-D VI-SLAM (visual inertial SLAM) methods, ARKit and ORB-SLAM3 [39]. For both of them, the camera pose at the first frame is computed by detecting "multiple markers." From the second frame, however, the cameras are tracked by their own methods. In contrast, our methods (both V1 and V2) detect a single marker attached to a large column. Further, it is done only for the first frame, and no markers are detected for the subsequent frames. Table 1 summarizes the camera tracking methods.

Along each of four trajectories given in Fig. 10, an RGB-D video was captured. Then, we ran ARKit, ORB-SLAM3, V1 and GT on a PC. Fig. 11 shows the sequences of snapshots generated along $P_{1,1}$. The BIM columns are projected using the estimated camera poses. In both ARKit and ORB-SLAM3, camera tracking accumulates drift over time, making the BIM columns registered incorrectly. In contrast, the tracking capability of V1 is comparable to that of GT. With $P_{1,2}$ and $P_{2,1}$, similar results are obtained(The results of quantitative analysis will be presented later).

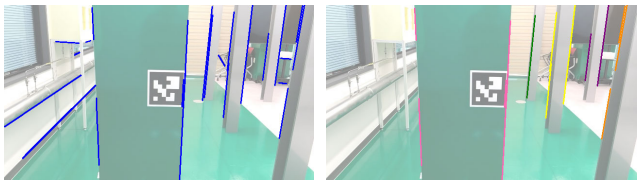

FIGURE 14. Detection of "scene edges" using column masks: (a) There exist many outliers. (b) The outliers are removed to generate the scene edges.

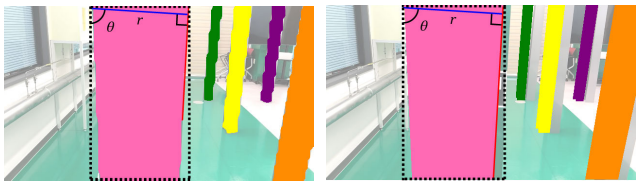

FIGURE 15. Edge matching: (a) Hough line transform of "a scene edge" with respect to the local space of a column mask. (b) Hough line transform of "a BIM edge" with respect to the local space of a BIM mask.

Fig. 12 shows the results for $P_{2,2}$. Not only ARKit and ORB-SLAM3 but also V1 suffers from drift error. Our analysis reveals two major problems of V1: (1) The scene edges and faces often include a number of *outliers*. (2) It is not guaranteed that the scene edges and faces are correctly paired with the corresponding entities of the BIM. Motivated by the analysis results, we have developed V2, which integrates *instance segmentation* into V1.

V. V2 = V1 + COLUMN SEGMENTATION

V2 uses an instance segmentation network to segment the input RGB image into the columns' *masks*, as shown in Fig. 13-(a), where the "column masks" are alpha blended with the input image for visualization purposes. For the sake of real-time performance, we adopt YOLACT++ [40] among many existing instance segmentation networks [41] due to its superiority in the processing speed.

A. COLUMN MATCHING

We assign each column mask the ID of the corresponding BIM column. For this, the depth pixels located within the column mask are back-projected into 3D space, and the mean of their 3D coordinates is taken as the *centroid*. On the other hand, the BIM columns are rendered with the *initial guess* of the camera pose, as shown in Fig. 13-(b). We call the rendered "BIM masks." The 3D centroid of each BIM mask is computed in the same manner as above. Then, searching for the closest centroids between the column and BIM masks, we assign the ID of a BIM column to each column mask in the RGB image.

The initial guess, which is made via homography for the first frame and by ARKit for the subsequent frames, is inaccurate in general. This can be observed in Fig. 13-(b),

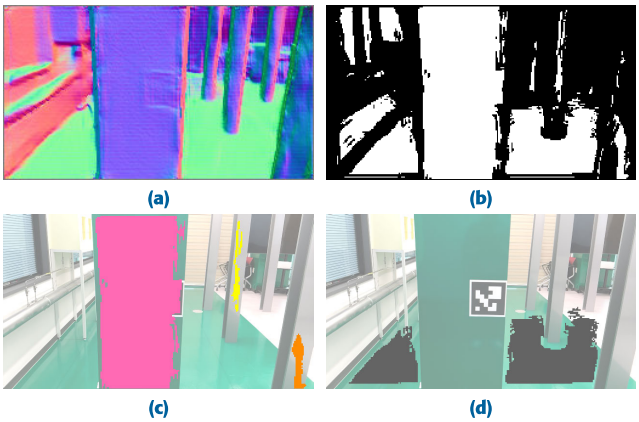

FIGURE 16. Scene face detection: (a) Normal map. (b) Colored in white are the pixels that are judged to belong to planes. (c) Scene faces that belong to columns. (d) Scene faces that belong to the floor.

where part of the large column is visible although most of it is hidden by its BIM mask. Consequently, the BIM masks' centroids do not coincide with the corresponding column masks'. Nonetheless, matching with centroids works robustly because the columns are spaced several meters apart in the 3D space.

B. EDGE DETECTION AND MATCHING

Fig. 14-(a) is the copy of Fig. 5-(f), which shows many *outlying* edges that do not belong to columns. In V2, we have the column masks, and it is straightforward to remove such outliers; if an edge belongs to no column masks, it is removed. Fig. 14-(b) shows the remaining. They are taken as "scene edges."

V2 does not use the Manhattan world assumption, but the result shown in Fig. 14-(b) is much better than that of V1 given in Fig. 5-(h). Also note that the scene edges in V2 are associated with specific BIM columns, thanks to column matching presented in Section V-A. To visualize such association, in Fig. 14-(b), the scene edges belonging to a column are drawn in the same color.

Recall that in V1, r and θ for each scene edge's Hough transform are defined in the *global space* of the input image. In contrast, as each scene edge in V2 belongs to a specific column mask, r and θ can be defined in the *local space* of the mask. For each column mask, its bounding box is computed, as depicted as a dotted box in Fig. 15-(a), and the upper-left corner is taken as the origin of the local space. In Fig. 15-(a), r and θ are defined for the right edge of the column. For all scene edges that belong to a column, their Hough transforms are defined in the same local space.

In Fig. 15-(b), which shows the BIM masks generated with the initial guess of the camera pose, each BIM edge is also defined in a BIM mask's local space. Then, thanks to column matching presented in Section V-A, edge matching can be made *per column*, and the scene and BIM edges are paired far more accurately than in V1.

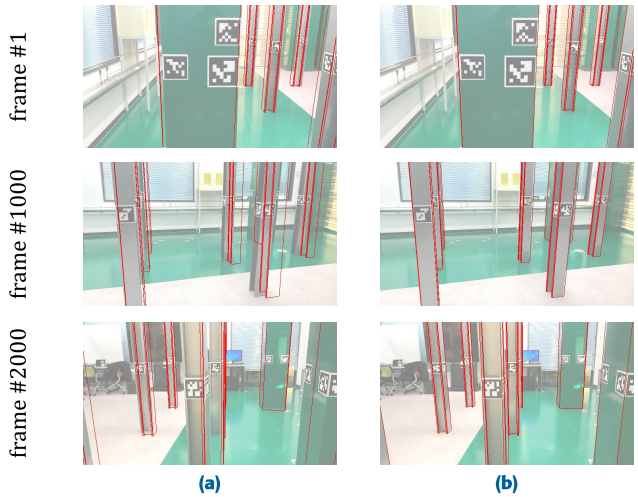

FIGURE 17. Snapshots extracted along $P_{2,2}$: (a) V1's results copied from Fig. 12-(c). (b) V2's results.

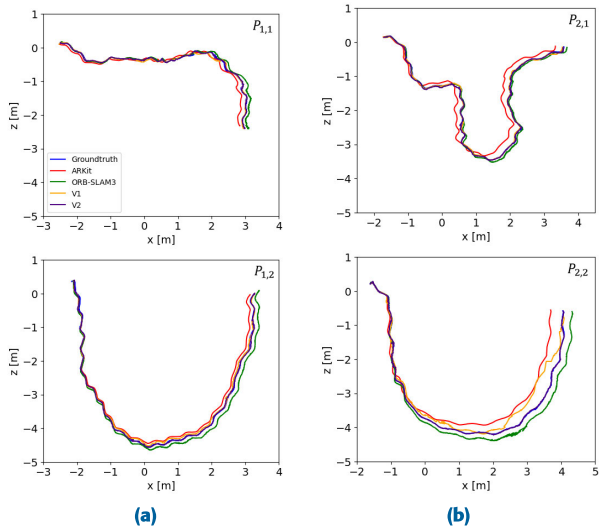

FIGURE 18. Camera trajectories: (a) $P_{1,1}$ and $P_{1,2}$ in Scene 1. (b) $P_{2,1}$ and $P_{2,2}$ in Scene 2.

C. FACE DETECTION AND MATCHING

ARKit's plane detection is not reliable. When a plane is detected from a column's face, for example, its surface normal is often significantly different from the face normal. Therefore, V2 directly detects "scene faces" using column masks. As presented in Section IV-A1, the 3D vertex and normal maps are generated from the input depth map. Fig. 16-(a) shows the normal map. We test if each pixel belongs to a plane by computing the point-to-plane distance and the normal discrepancy with its neighboring pixels. The test produces a binary image, shown in Fig. 16-(b), where the planar areas are colored in white. It is partitioned into a set of *connected components* [42]. The components that overlap with column masks (Fig. 16-(c)) and those whose surface normals are vertical (Fig. 16-(d)) make up the "scene faces."

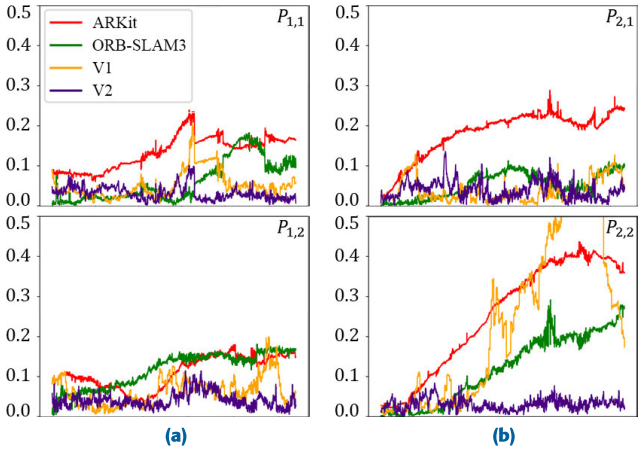

FIGURE 19. ATE for each trajectory: (a) $P_{1,1}$ and $P_{1,2}$ in Scene 1. (b) $P_{2,1}$ and $P_{2,2}$ in Scene 2.

TABLE 2. ATE RMSE (in meters).

		ARKit	ORB-SLAM3	V1	V2
Scene 1	$P_{1,1}$	0.141	0.079	0.061	**0.036**
	$P_{1,2}$	0.118	0.125	0.072	**0.045**
Scene 2	$P_{2,1}$	0.191	0.060	0.047	**0.045**
	$P_{2,2}$	0.288	0.147	0.305	**0.033**

For each detected scene face, the mean of its 3D vertex coordinates is taken as its centroid, and that of its vertex normals is taken as the face normal. The same information is obtained from each BIM face. Then, face matching is made using a k-d tree with the centroid and face normal. Similar to edge matching, face matching is made *per column* or *between floors*, and the accuracy is significantly increased.

D. OPTIMIZATION

Recall that in V1, RANSAC is used to remove incorrect matching. In V2, however, we have few mismatched pairs, thanks to column masks. Therefore, instead of RANSAC, a two-stage optimization is performed. Initially, we run the Levenberg-Marquardt method with all pairs. Then, with the computed camera pose, the error of each pair is computed so as to remove all mismatched pairs. With the remaining, i.e., with the inliers only, we run the Levenberg-Marquardt method to compute the final camera pose.

E. EXPERIMENTS AND EVALUATION

Fig. 17 shows that V2 computes the camera poses accurately for the trajectory, $P_{2,2}$, for which V1 does not. In numerous experiments we made, V2 remains robust and accurate. Fig. 18 depicts the camera trajectories estimated by ARKit, ORB-SLAM3, V1, V2 and GT. Observe that the camera trajectories estimated by V2 are almost identical to those of GT. For each trajectory, the absolute trajectory error (ATE), i.e., the average deviation from the GT trajectory per frame,

TABLE 3. RPE RMSE (translation in meters and rotation in degrees).

			ARKit	ORB-SLAM3	V1	V2
Scene 1	$P_{1,1}$	t	0.040	0.063	0.062	**0.038**
		r	0.829	0.897	1.115	**0.774**
	$P_{1,2}$	t	0.047	0.043	0.073	**0.030**
		r	**0.341**	0.478	1.129	0.832
Scene 2	$P_{2,1}$	t	0.054	0.041	0.062	**0.034**
		r	0.732	0.799	1.542	**0.587**
	$P_{2,2}$	t	0.113	0.069	0.229	**0.035**
		r	**0.227**	0.660	2.802	0.695

TABLE 4. Running time of V2 (in milliseconds) where std denotes standard deviation.

steps	median	mean	std
preprocessing	4.64	4.74	0.54
column mask generation	42.00	42.34	3.52
column matching	2.11	2.13	0.13
scene edge detection	6.47	6.59	0.99
edge matching	3.48	3.44	0.67
scene face detection	1.33	1.33	0.12
face matching	2.10	2.11	0.28
optimization	3.07	3.11	0.40
total	65.60	65.80	4.49

is computed. Fig. 19 depicts the ATE graphs, and Table 2 shows the root mean square error (RMSE) of ATE for each trajectory. Both V1 and V2 are better than ARKit and ORB-SLAM3, and V2 shows significantly improved performances over V1. Additionally, the relative pose error (RPE) is computed every 300 frames for each trajectory. Table 3 shows that V2 demonstrates significant improvements over V1 and it is generally better than ARKit and ORB-SLAM3.

As presented in this section, VII is composed of the following steps: (1) Preprocessing, where not only the vertex and normal maps are computed, but the BIM masks (as shown in Fig. 13-(b)) and the BIM edges/faces are also generated. (2) Column mask generation and column matching. (3) Scene edge detection and edge matching. (4) Scene face detection and face matching. (5) Optimization. Table 4 enumerates the statistics of the time consumed in each step for four trajectories.

VI. DATA GENERATION FOR COLUMN SEGMENTATION

The dataset for training YOLACT++ is composed of the RGB images captured from various locations in the environment and their "column masks." The column masks are first generated automatically and are manually post-processed if incompletely segmented.

A. AUTOMATIC SEGMENTATION

We use Segment Anything Model (SAM) [43] to segment the input RGB image (Fig. 20-(a)) into what we call "SAM masks" (Fig. 20-(b)). Observe that the small column in the middle of Fig. 20-(a) is segmented into two SAM masks in Fig. 20-(b).

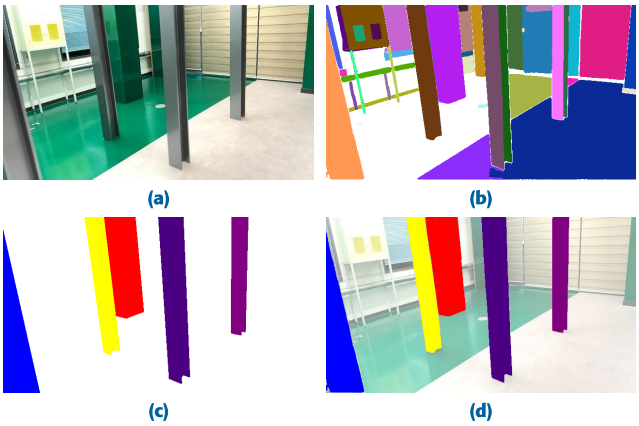

FIGURE 20. Column mask generation - Example 1: (a) Input RGB image. (b) SAM masks. (c) BIM masks. (d) The SAM and BIM masks are compared to generate the column masks.

Fig. 20-(c) shows the "BIM masks." Let M_{SAM} and M_{BIM} denote a SAM mask and a BIM mask, respectively. M_{SAM} is judged to belong to M_{BIM} if the following condition is satisfied:

$$\frac{M_{\text{SAM}} \cap M_{\text{BIM}}}{M_{\text{SAM}}} > \alpha \tag{6}$$

where α is a predetermined threshold (0.6 in the current implementation). Collecting all SAM masks that belong to a BIM mask, a "column mask" is generated. The two SAM masks in the middle of Fig. 20-(b) are combined into a column mask in Fig. 20-(d).

The camera pose used to generate the BIM masks (e.g., in Fig. 20-(c)) should be as accurate as possible. Otherwise, the BIM masks would not be sufficiently overlapped with the corresponding SAM masks, making it hard to create the column masks. Initially, we used ARKit for generating the BIM masks but suffered from its unreliable tracking capability. Replacing ARKit by V1, we found that automatic segmentation stage works fine in general.

B. MANUAL POST-PROCESSING

In the RGB image shown in Fig. 21-(a), consider the large column at the right. Fig. 21-(b) shows its SAM masks. Observe that some area of the column is left blank, i.e., unsegmented. It is because the large column's surface is highly reflective. Given the BIM masks shown in Fig. 21-(c), the large column's mask is computed yet to be incomplete and disconnected, as shown in Fig. 21-(d).

To resolve such a problem, we have developed an easy-to-use *data engine*, which includes features for generating new masks with SAM's *point prompt* input. Fig. 22 shows the interface of the engine, where the automatically-generated column masks (shown in Fig. 21-(d)) are overlaid on top of the input RGB image (shown in Fig. 21-(a)). The color array under the title, Index, represents the indices of the automatically-generated column masks. Having selected the red color, the human annotator clicks the unsegmented

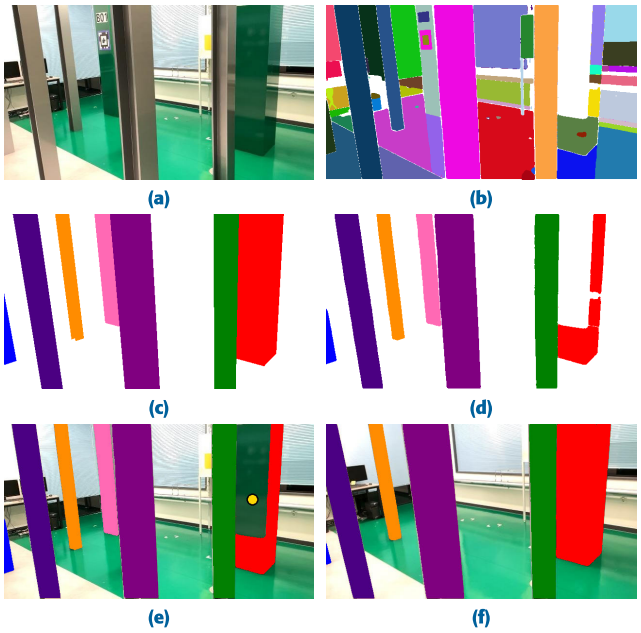

FIGURE 21. Column mask generation - Example 2: (a) Input RGB image. (b) SAM masks. (c) BIM masks. (d) The large column's mask is incomplete. (e) The two red masks shown in (d) are connected by the point prompting presented in Fig. 22. To complete the large column's mask, another point prompting (the yellow dot) is made. (f) The large column's mask is completed.

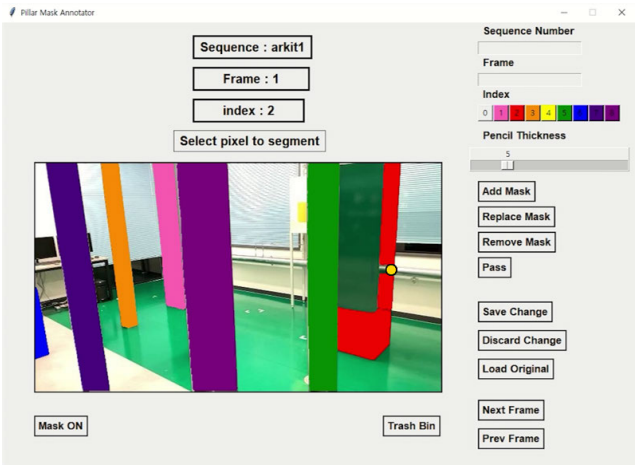

FIGURE 22. The human annotator has selected the red color under Index and is point-prompting the unsegmented area. See the yellow dot. Then, two red masks of the large column will be connected, as shown in Fig. 21-(e).

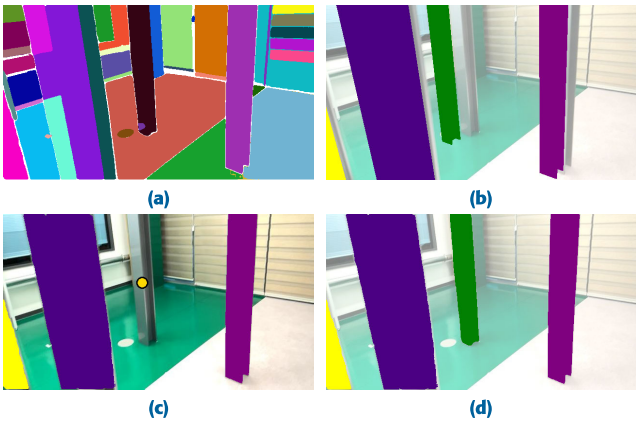

FIGURE 23. Column mask generation - Example 3: (a) SAM masks. (b) BIM masks. (c) Point prompting. (d) Completed column masks.

area of the large column. See the yellow dot. It is point prompting, which is aimed at filling the blank area with the red color. Then, by pressing Add Mask button, the blank area is filled(Watch the accompanying video). The result is shown in Fig. 21-(e), where an additional step of point prompting (visualized also as a yellow dot) is being made. Finally, the large column's mask is completed, as shown in Fig. 21-(f). The data engine provides more functionalities such as replacing an existing mask with a new one (via Replace Mask button) and removing existing masks (via Remove Mask button).

With this two-stage (automatic and manual) process, a set of over 50,000 training data is created. Training YOLACT++ with the dataset consumes about 90 hours with an NVIDIA GeForce RTX 3090 Ti(It can be trained much faster with multiple latest GPUs). Suppose that our camera tracker is ported to a novel environment, for example, where its columns' shapes are different from those of the rectangular parallelepiped columns and the H-beam columns. Then, YOLACT++ needs to be trained with the data extracted from the environment, but our data engine enables the training dataset to be generated easily and quickly.

VII. DISCUSSION

As presented in Section VI-A, VI is used for generating the BIM masks. In a certain frame, however, the camera pose returned by V1 may not be sufficiently accurate. Fig. 23-(a) shows the SAM masks of an input RGB image, and Fig. 23-(b) overlays the V1-generated BIM masks on top of the RGB image. Observe that the green BIM mask is not sufficiently overlapped with the small column in the middle. Consequently, no column mask is generated automatically for the column.

The point prompting provided in our data engine resolves this problem. In Fig. 23-(c), the yellow dot implies point-prompting the unsegmented middle column. Fig. 23-(d) shows that a complete mask is generated for the middle column(Watch the accompanying video).

To accurately determine the camera pose in our model-based camera tracking method, the Manhattan axis constraint presented in Section IV-B must be satisfied. For example, suppose that neither the floor nor the ceiling is captured by the camera even though a number of vertical edges are detected. Then, our tracker may drift vertically. Fortunately, such a case rarely happens in a wide indoor environment, at which our method is targeted, because the floor and ceiling are too wide to be missed in the input image.

On the other hand, if no columns are captured by the camera, we have to rely entirely on the camera pose estimated by ARKit. After the camera starts to capture the columns in the scene, we return to the normal iterations presented in Fig. 4.

VIII. CONCLUSION

In this paper, we presented a camera tracking method designed for an indoor environment with fixed objects. It employs a model-based approach that matches the fixed objects extracted from the input image with their BIMs to estimate the camera pose. Our experiments demonstrate accuracy and robustness of the model-based camera tracking method.

In the current implementation, the columns that are indispensable in a fab are taken as the fixed objects. Our method can be extended to work in a column-less indoor environment, for example, if there exist a sufficient number of window edges or edges formed at the junctions of walls. The algorithms presented in this paper will be extended along the direction.

REFERENCES

[1] K. Amin, G. Mills, and D. Wilson, "Key functions in BIM-based AR platforms," *Autom. Construction*, vol. 150, Jun. 2023, Art. no. 104816.
[2] K. W. May, C. Kc, J. J. Ochoa, N. Gu, J. Walsh, R. T. Smith, and B. H. Thomas, "The identification, development, and evaluation of BIM-ARDM: A BIM-based AR defect management system for construction inspections," *Buildings*, vol. 12, no. 2, p. 140, Jan. 2022.
[3] 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, Oct. 2015.
[4] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras," *IEEE Trans. Robot.*, vol. 33, no. 5, pp. 1255–1262, Oct. 2017.
[5] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt, "Bundle-Fusion: Real-time globally consistent 3D reconstruction using on-the-fly surface reintegration," *ACM Trans. Graph.*, vol. 36, no. 4, p. 1, Aug. 2017.
[6] T. Schöps, T. Sattler, and M. Pollefeys, "BAD SLAM: Bundle adjusted direct RGB-D SLAM," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 134–144.
[7] S. Zhang, L. Zheng, and W. Tao, "Survey and evaluation of RGB-D SLAM," *IEEE Access*, vol. 9, pp. 21367–21387, 2021.
[8] P. Smith, I. D. Reid, and A. J. Davison, "Real-time monocular SLAM with straight lines," in *Proc. Brit. Mach. Vis. Conf.*, 2006, pp. 17–26.
[9] 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. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2017, pp. 4503–4508.
[10] H. Wei, F. Tang, Z. Xu, C. Zhang, and Y. Wu, "A point-line VIO system with novel feature hybrids and with novel line predicting-matching," *IEEE Robot. Autom. Lett.*, vol. 6, no. 4, pp. 8681–8688, Oct. 2021.
[11] H. Zhou, D. Zou, L. Pei, R. Ying, P. Liu, and W. Yu, "StructSLAM: Visual SLAM with building structure lines," *IEEE Trans. Veh. Technol.*, vol. 64, no. 4, pp. 1364–1375, Apr. 2015.
[12] H. Li, J. Yao, J.-C. Bazin, X. Lu, Y. Xing, and K. Liu, "A monocular SLAM system leveraging structural regularity in Manhattan world," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 2518–2525.
[13] Y. Li, N. Brasch, Y. Wang, N. Navab, and F. Tombari, "Structure-SLAM: low-drift monocular SLAM in indoor environments," *IEEE Robot. Autom. Lett.*, vol. 5, no. 4, pp. 6583–6590, Oct. 2020.
[14] J. Liu and Z. Meng, "Visual SLAM with drift-free rotation estimation in Manhattan world," *IEEE Robot. Autom. Lett.*, vol. 5, no. 4, pp. 6512–6519, Oct. 2020.
[15] S. Yang, Y. Song, M. Kaess, and S. Scherer, "Pop-up SLAM: Semantic monocular plane SLAM for low-texture environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2016, pp. 1222–1229.
[16] Q. Sun, J. Yuan, X. Zhang, and F. Duan, "Plane-edge-SLAM: Seamless fusion of planes and edges for SLAM in indoor environments," *IEEE Trans. Autom. Sci. Eng.*, vol. 18, no. 4, pp. 2061–2075, Oct. 2021.
[17] S. Lin, J. Wang, M. Xu, H. Zhao, and Z. Chen, "Contour-SLAM: A robust object-level SLAM based on contour alignment," *IEEE Trans. Instrum. Meas.*, vol. 72, pp. 1–12, 2023.
[18] J. Park, J. Chen, and Y. K. Cho, "Self-corrective knowledge-based hybrid tracking system using BIM and multimodal sensors," *Adv. Eng. Informat.*, vol. 32, pp. 126–138, Apr. 2017.
[19] K. Asadi, H. Ramshankar, M. Noghabaei, and K. Han, "Real-time image localization and registration with BIM using perspective alignment for indoor monitoring of construction," *J. Comput. Civil Eng.*, vol. 33, no. 5, Sep. 2019, Art. no. 04019031.
[20] D. Acharya, M. Ramezani, K. Khoshelham, and S. Winter, "BIM-tracker: A model-based visual tracking approach for indoor localisation using a 3D building model," *ISPRS J. Photogramm. Remote Sens.*, vol. 150, pp. 157–171, Apr. 2019.
[21] M. S. Moura, C. Rizzo, and D. Serrano, "BIM-based localization and mapping for mobile robots in construction," in *Proc. IEEE Int. Conf. Auto. Robot Syst. Competitions (ICARSC)*, Apr. 2021, pp. 12–18.
[22] J. Chen, S. Li, and W. Lu, "Align to locate: Registering photogrammetric point clouds to BIM for robust indoor localization," *Building Environ.*, vol. 209, Feb. 2022, Art. no. 108675.
[23] H. Yin, Z. Lin, and J. K. W. Yeoh, "Semantic localization on BIM-generated maps using a 3D LiDAR sensor," *Autom. Construction*, vol. 146, Feb. 2023, Art. no. 104641.
[24] D. Acharya, K. Khoshelham, and S. Winter, "BIM-PoseNet: Indoor camera localisation using a 3D indoor model and deep learning from synthetic images," *ISPRS J. Photogramm. Remote Sens.*, vol. 150, pp. 245–258, Apr. 2019.
[25] J. Chen, S. Li, D. Liu, and W. Lu, "Indoor camera pose estimation via style-transfer 3D models," *Comput.-Aided Civil Infrastruct. Eng.*, vol. 37, no. 3, pp. 335–353, Mar. 2022.
[26] X. Zhao and C. C. Cheah, "BIM-based indoor mobile robot initialization for construction automation using object detection," *Autom. Construction*, vol. 146, Feb. 2023, Art. no. 104647.
[27] A. Z. Kolaei, E. Hedayati, M. Khanzadi, and G. G. Amiri, "Challenges and opportunities of augmented reality during the construction phase," *Autom. Construction*, vol. 143, Nov. 2022, Art. no. 104586.
[28] S. Meža, Ž. Turk, and M. Dolenc, "Component based engineering of a mobile BIM-based augmented reality system," *Autom. Construction*, vol. 42, pp. 1–12, Jun. 2014.
[29] J. Choi, M. G. Son, Y. Y. Lee, K. H. Lee, J. P. Park, C. H. Yeo, J. Park, S. Choi, W. D. Kim, T. W. Kang, and K. Ko, "Position-based augmented reality platform for aiding construction and inspection of offshore plants," *Vis. Comput.*, vol. 36, nos. 10–12, pp. 2039–2049, Oct. 2020.
[30] L.-T. Tsai, H.-L. Chi, T.-H. Wu, and S.-C. Kang, "AR-based automatic pipeline planning coordination for on-site mechanical, electrical and plumbing system conflict resolution," *Autom. Construction*, vol. 141, Sep. 2022, Art. no. 104400.
[31] *Visuallive*. Accessed: Apr. 21, 2024. [Online]. Available: https://www.unity.com/products/visuallive
[32] R. Gomez-Ojeda, F.-A. Moreno, D. Zuñiga-Noël, D. Scaramuzza, and J. Gonzalez-Jimenez, "PL-SLAM: A stereo SLAM system through the combination of points and line segments," *IEEE Trans. Robot.*, vol. 35, no. 3, pp. 734–746, Jun. 2019.
[33] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "KinectFusion: Real-time dense surface mapping and tracking," in *Proc. 10th IEEE Int. Symp. Mixed Augmented Reality*, Oct. 2011, pp. 127–136.
[34] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-8, no. 6, pp. 679–698, Nov. 1986.
[35] N. Kiryati, Y. Eldar, and A. M. Bruckstein, "A probabilistic Hough transform," *Pattern Recognit.*, vol. 24, no. 4, pp. 303–316, Jan. 1991.
[36] J. M. Coughlan and A. L. Yuille, "Manhattan world: Compass direction from a single image by Bayesian inference," in *Proc. 7th IEEE Int. Conf. Comput. Vis.*, 1999, pp. 941–947.
[37] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975.
[38] V. Lepetit, F. Moreno-Noguer, and P. Fua, "EP*n*P: An accurate O(n) solution to the P*n*P problem," *Int. J. Comput. Vis.*, vol. 81, pp. 155–166, Jul. 2009.

[39] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM3: An accurate open-source library for visual, visual–inertial, and multimap SLAM," *IEEE Trans. Robot.*, vol. 37, no. 6, pp. 1874–1890, Dec. 2021.
[40] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee, "YOLACT++ better real-time instance segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 2, pp. 1108–1121, Feb. 2022.
[41] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos, "Image segmentation using deep learning: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 7, pp. 3523–3542, Jul. 2022.
[42] F. Bolelli, S. Allegretti, L. Baraldi, and C. Grana, "Spaghetti labeling: Directed acyclic graphs for block-based connected components labeling," *IEEE Trans. Image Process.*, vol. 29, pp. 1999–2012, 2020.
[43] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick, "Segment anything," 2023, *arXiv:2304.02643*.

JEONGHYEON AHN received the B.S. degree from the Department of Computer Science and Engineering, Korea University, where he is currently pursuing the Ph.D. degree. His research interests include geometric computer vision, 3D graphics, and deep learning.

JUNGHO HA received the B.S. degree from the Department of Computer Science and Engineering, Korea University, where he is currently pursuing the M.S. degree. His research interests include 3D graphics and deep learning.

JAEMIN SON received the B.S. degree from the Department of Computer Science and Engineering, Korea University, where he is currently pursuing the M.S. degree. His research interests include extended reality, 3D graphics, and deep learning.

JUNGHYUN HAN (Member, IEEE) received the B.S. degree in computer engineering from Seoul National University, the M.S. degree in computer science from the University of Cincinnati, and the Ph.D. degree in computer science from the University of Southern California. He is currently a Professor with the Department of Computer Science and Engineering, Korea University, where he directs the Superintelligence Research Center. Prior to joining Korea University, he was with the School of Information and Communications Engineering, Sungkyunkwan University, South Korea, and the Manufacturing Systems Integration Division, U.S. Department of Commerce National Institute of Standards and Technology.