Change-Resilient Localization Estimation

Fernando Reyes-Aviles* VRVis GmbH
Philipp Fleck† Graz University of Technology
Dieter Schmalstieg‡ University of Stuttgart
Clemens Arth§ Graz University of Technology

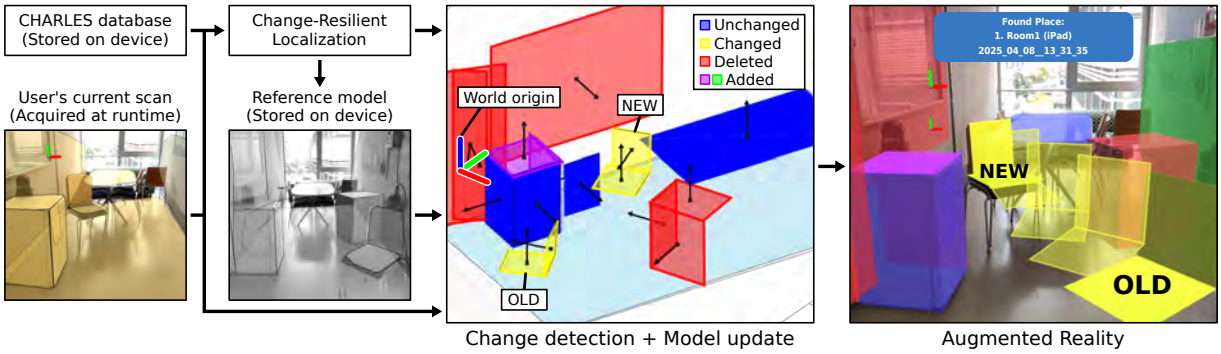

Figure 1: Our change-resilient localization method receives as input a set of primitives (*user's current scan*) captured with an XR mobile device such as an iPad Pro or an Apple Vision Pro. The primitives are combined into geometric descriptors that we compare to the models stored in our database to (a) perform localization (i.e., place recognition and pose estimation), (b) detect primitives whose pose in the current scan and the reference model differ (i.e., primitives whose pose changed between *sessions*), and (c) annotate such changes. The output of our method is the pose of the device in a global reference frame, and an updated set of primitives which preserves the *world origin* of the reference model, while incorporating the changes in the scene. In the *change detection + model update* step of the method, we show unchanged primitives in blue, absent primitives in red, changed primitives in yellow, and new primitives in purple or green.

Abstract

Indoor localization is essential in applications such as augmented reality or robotics. Existing solutions for localization in static scenes work well even for large environments, but localization in environments with movable objects whose pose in the scene change between sessions remains challenging. In this paper, we propose a change-resilient localization method based on a novel geometric descriptor computed only from geometric primitives. Our method is capable of re-identifying primitives that have moved in the scene. We leverage this feature to update a stored reference model (anchor) of the environment to accommodate the changes, which enables localization that is resilient to changes in the scene. We report on a set of experiments demonstrating the robustness and scalability of our method. In addition, we present use cases highlighting the importance of being able to update a reference model.

Index Terms: Localization, Change Detection, World Anchor, 3D Registration, Augmented Reality, Digital Twin.

1 Introduction

Global localization, the process of finding a known world origin in a known environment, is essential in fields such as augmented reality (AR) or robotics. Navigating a building in AR or accessing a device in a known location, such as a smart device in the Internet of Things (IoT), requires knowing one's pose in a global reference frame. Although much research has investigated how to scale localization to wide areas, making localization robust if the scene is not static and objects move between *sessions* remains challenging.

*e-mail: reyesaviles@tugraz.at
†e-mail: philipp.fleck@tugraz.at
‡e-mail: schmaldr@visus.uni-stuttgart.de
§e-mail: clemens.arth@tugraz.at

Existing localization frameworks that assume a static, immutable scene are easily confused if unique objects or landmarks appear in changing locations. The ability to detect such changes and update the reference model is of key importance to further improve the robustness of localization.

Various commercial XR platforms offer global localization with respect to a previously acquired map of the environment. The most prominent examples today include ARKit anchors [2], ARCore anchors [25], Azure spatial anchors [38] and Vuforia area targets [51]. These platforms acquire a 3D map of the environment using on-device sensors and extract a proprietary feature set for relocalization, a so-called *anchor*. With an anchor, instant localization is possible. However, these anchors require extensive storage, and updating anchors to incorporate changes in the environment is usually not possible without repeating the entire anchor generation process.

Recently, we demonstrated that anchors can be constructed from geometric properties of the scene [41, 42]. Compared to the proprietary feature descriptors used by commercial platforms, which typically encode detailed visual properties of individual interest points, geometric descriptors encode scene information on a much higher level of abstraction. This property makes geometric descriptors very lightweight and shareable across platforms. It is enabled by the *scene understanding* capabilities of devices such as the HoloLens 2 [36] or the Apple Vision Pro [3], which acquire a 3D point cloud of the environment and fit polygonal primitives to planar clusters of points.

While anchors composed of geometric descriptors have many advantages over their visual descriptor counterparts, both descriptor flavors have not been designed with updates in mind. This limitation must be considered a missed opportunity, especially for geometric descriptors: Scene understanding can instantly generate geometric primitives from new observations, but existing geometric descriptors are not designed to incorporate these changes.

In this paper, we propose CHARLES (short for "CHAnge-Resilient Localization EStimation") to overcome this limitation. We extend our previous work [42] to make geometric descriptors resilient against changes and use these enhanced descriptors for relocalization in the presence of significant changes in the scene.

Our method takes as input a set of primitives that is taken directly from the scene understanding component of XR devices such as the iPad Pro or the Apple Vision Pro. Typically, they detect only *large* primitives. Consequently, we work with objects in human-made environments that can be decomposed into such primitives, for example, chairs, couches, desks, etc. (see Figure 1). Note that, they are not capable of detecting primitives from small parts of objects such as legs of a desk. **We do not use RGB images, RGB-D data or dense depth maps**. Thus, we do not detect changes in objects such as coffee cups or cellphones. The output of our algorithm is the pose of the device in a global reference frame, and an updated set of primitives that preserves the *world origin* of the *reference model*, while incorporating the primitives changes in the *user's current scan* (see Figure 2). In summary, this paper presents the following core contributions:

- A localization method capable of finding the *world origin* in a known environment, even when the pose of a substantial amount of objects in the scene changed.
- A change detection method capable of finding scene-to-model correspondences of primitives whose pose in the scene changed.
- A method to update a stored reference model by adding, removing and updating primitives, while preserving the original *world origin* of the anchor (reference model).
- A workflow to leverage existing reconstruction tools for change-resilient localization in XR applications, e.g., Leica or Matterport scanners.

Note that, the scene understanding component of XR devices cannot provide updates of the pose of *dynamic* objects at frame rate. Therefore, we designed our change detection method to find and annotate changes between sessions. We do not perform single-session change detection of dynamic objects in the scene. Also, we leverage the simultaneous localization and mapping (SLAM) capabilities of XR devices only to scan a scene and get a real-time updated pose in the camera coordinate system. Starting with PTAM [29], SLAM systems split tracking (full frame rate) from mapping (slower). As already discussed in previous work [49], the global localization process runs at an even slower rate. In the worst case scenario, AR overlays become available several seconds later. Therefore, in this paper we refer as instant localization to the process of determining the position and orientation of a device, in a global reference frame, without significant delay (e.g., within one second).

To demonstrate the practical relevance of our new method, we run it on two ubiquitous XR devices: iPad Pro and Apple Vision Pro. In addition, we analyze the plausibility of creating scalable data structures using open-source tools such as ScanNet [10, 11] and PlaneRCNN [33, 34]. Furthermore, we show that our localization method can be used in conjunction with standard reconstruction tools such as CloudCompare [1] and the Matterport Pro3 camera [35], to enable change-resilient localization in large-scale environments.

2 Related Work

Our work is at the intersection of research areas including change detection, pose estimation in dynamic environments, and SLAM. In the following, we revisit the most important prior work in these research fields.

2.1 Change detection

Change detection is concerned with all applications that require the finding of differences between two geometric models. Examples of applications include the analysis of discrepancies in point clouds [23] and version control in collaborative virtual design [57].

Fehr et al. [17] developed a method for robotics which deals with changes in the environment and adequately represents them in the 3D map. They propose a 3D reconstruction algorithm based on an extended truncated signed distance function (TSDF) to continuously refine the map while simultaneously obtaining 3D reconstructions of dynamic objects in the scene.

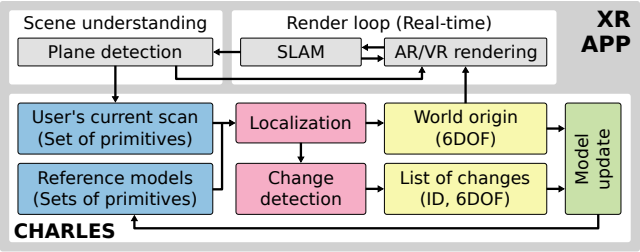

Figure 2: Example of XR app (3D game engine) using CHARLES. We leverage the proprietary SLAM and scene understanding components of XR devices (gray blocks) to detect primitives in the scene and forward them to CHARLES. The user's scan (input) is compared to the stored models at primitive level to determine the world origin and a list of changes in the scene, which we use to update the reference model (output). CHARLES reports the world origin to the rendering component to align the virtual content accordingly.

Langer et al. [17] addressed the problem of detecting novel objects introduced into the scene for robotics applications. The authors combine semantic information with horizontal plane detection to distinguish novel objects from permanent objects, such as furniture.

Qiu et al. [40] proposed a method to describe detailed scene changes, including change types, object attributes, and spatial locations. To robustly assess the changes, they used two registered 3D scans of the scene observed from different routes and viewpoints.

Xu et al. [54] proposed a 3D reconstruction and scene understanding approach for AR applications, based on geometric and semantic information. They take advantage of modern AI tools to enable users to interact with physical spaces and objects in a spatially and semantically meaningful manner. Their 3D reconstruction is based on TSDF fusion [9], followed by 3D semantic segmentation based on COCO [32], which allows them to identify more than 100 object categories. To keep track of object changes between sessions, they build a graph that combines geometric and semantic features of the objects in the scene. Semantic segmentation is a very powerful tool, but it requires dense input data and ample processing power. In contrast, our geometric descriptors are lightweight, fast to compute, and they do not rely on the ability to detect specific object classes which may or may not be present in the target scene.

A class of methods including MeshGit [13], 3DFlow [14], SceneGit [5] and NodeGit [43] provide version control for 3D models. These methods capture changes in meshes, materials, shapes, etc., to create digital content collaboratively. Version control methods such as *merging* enable concurrent work of artists. Changes are detected by dissecting 3D models into their individual parts and tracking deviations from an ancestor model. Establishing a history of changes on the geometric entities is an essential requirement for change detection, which we address in our method as well. However, we focus on models of real environments and on fast relocalization. This requirement is fundamentally different from digital content creation, where a unique global coordinate system is always known, while the main interest lies in reliably detecting changes to individual objects.

A separate area considers correspondences between non-rigid 3D shapes. For example, Donati et al. [15] applied machine learning to computing correspondences between deformable shapes. A key element of their method is a feature extraction network that learns descriptors on pairs of shapes. With these descriptors, dense point-to-point correspondences between a source and target shape can be established. Handling deformations goes beyond the scope of our work. For the objective of relocalization, we can assume that the changes are primarily caused by changing the location of rigid objects between sessions, while deformations are of limited interest.

2.2 Localization and pose estimation

Localization can be challenging in non-static environments as the scene appearance changes between sessions. Cupec et al. [7, 8] presented a method for indoor place recognition based on the matching of planar surfaces and straight edges extracted from RGB-D data. The authors discussed the main cases for failure: repetitive geometric structure, wide spaces and lack of geometric structure. These are well-known problems in the literature.

Fernandez-Moral et al. [19, 20] developed a method for place recognition and pose estimation based on planar regions extracted from range data. They used geometric features such as the area and the normal vector. Their method enables localization in non-static scenes where the main structure of the scene remains unchanged. They achieved high localization success with moderate changes in the scene. PlaneLoc2 [53] is another method that leverages planar regions for 3D global localization. It uses triplets of planar segments to generate a probability distribution, which is employed for pose estimation with respect to a global map of the scene.

More recently, Suo et al. [47] proposed a relocalization method based on planar regions. Similarly to our approach, their method relies solely on textureless input data and does not depend on other information, such as RGB images. These methods showed good localization results; however, they do not elaborate on detecting changes in the scene or updating the reference model.

Dubé et al. [16] proposed an algorithm for place recognition in 3D laser data based on the concept of segment matching. Their approach does not rely on assumptions about the environment being composed of geometric primitives, such as planes, or a rich library of objects. However, it does not address the problem of localizing when the appearance of the scene has changed.

Gomez et al. [24] presented a localization that combines metric and semantic information to increase robustness in non-static environments containing movable objects. Semantic information extracted from RGB images provides a good coarse localization, and metric information from depth images delivers fine estimations. In contrast, we rely only on information extracted from geometric primitives (e.g., the surface area of polygonal primitives).

Li and Harada [31] proposed a method based on machine learning for point cloud registration in rigid and deformable scenes. Their approach lies somewhere in the middle between pose estimation and change detection. While they do not explicitly look for changes in deformable scenes, they are able to discern topological changes. We refer the reader to the survey from Deng et al. [12] for an extensive coverage of non-rigid 3D registration.

2.3 Simultaneous localization and mapping

Dynamic environments remain a challenge in SLAM to this day [18]. Many methods [30, 50, 52, 56] have been proposed to tackle real-world challenges such as changing environment geometries, lighting conditions, and camera or motion artifacts. Solving these problems would enable robust long-term operation and reliable performance in dynamic environments. Methods like DS-SLAM [55], SOF-SLAM [6] or OVD-SLAM [27] combine semantic segmentation with movement detection to exclude, from tracking and mapping, features from objects moving across the camera.

In contrast, our approach operates on a higher abstraction level, where the environment is represented by geometric primitives rather than natural features, point clouds or raw RGB-D data. We use a SLAM front-end (e.g., Apple's ARKit or ORB-SLAM [4]) only to scan a known scene and get a real-time updated pose in the camera coordinate system. Our method provides the absolute 6DOF pose w.r.t. a known world origin (stored reference model). In addition, it detects changes of movable objects, i.e., objects which do not appear in the same pose in the reference model and the user's current scan.

3 Scene Description

Our objective is to support relocalization with geometric descriptors that are resilient to changes in the scene resulting from changing the location of rigid objects between sessions. BOWA [42] presented a geometric descriptor, computed from *pairs of parametric primitives*, which is robust enough to perform relocalization in static environments. Consequently, a simple two-level hierarchy was sufficient: Primitives from a scanned *scene* are grouped into a *reference model*, and the reference models are stored in a search structure.

If we allow the location of movable rigid objects to be changed, each consisting of multiple primitives, the naïve strategy of BOWA is no longer sufficient. Assume that we are trying to match the descriptors in a reference model with the descriptors from the same place after the objects have moved. All primitives that make up a rigid object will move together. Therefore, descriptors that only involve primitives of a single rigid object will remain close to identical. Geometric descriptors involving primitives of multiple objects may or may not remain equal, depending on whether one of the objects has moved. Moreover, descriptors that involve symmetric or rotationally invariant primitives can lead to ambiguous matches after moving.

Overcoming these challenges requires the development of more expressive scene descriptions. We start with a set of primitives delivered by the scene understanding component of an XR platform. A primitive $M = (\mathbf{p}, \mathbf{n}, \mathbf{u}, \mathbf{v}, \mathbf{c})$ is created from a set of coplanar points. We store the centroid $\mathbf{p}$ and the normal $\mathbf{n}$ of the supporting plane. Moreover, we determine a bounding rectangle that includes all points in the supporting plane with minimal area. The bounding rectangle has edge vectors $\mathbf{u}, \mathbf{v}$, such that $\|\mathbf{u}\|_2 \leq \|\mathbf{v}\|_2$. In addition, we store the *primitive category* $\mathbf{c}$. Commercial tools like AR Foundation [48] and Microsoft's Scene Understanding SDK [37] classify the planar surfaces detected in the scene into a variety of categories including floor, wall, ceiling, etc.

Furthermore, we aggregate connected primitives into *clusters* to better handle movable objects. A cluster represents an object or parts of an object, depending on the quality and completeness of the observations. For example, clusters can correspond to items of furniture, and can change if the interior is rearranged. To store clusters, we extend the hierarchical structure to three levels: Stored reference models consist of clusters, and clusters consist of primitives. Our change detection strategy first attempts to find cluster correspondences and resolve any unmatched primitives, which could have disappeared, been newly added, or are incorrectly assigned.

3.1 Aggregating primitives into clusters

To create clusters from primitives, we measure the distance between the primitives. We join a pair of primitives into a cluster if two vertices of each bounding rectangle match. A match is defined by the Euclidean distance between a pair of vertices below a threshold. The result of this procedure is an undirected graph, given as an adjacency matrix, which has primitives as nodes and membership in the same cluster as edges. The clusters are found with the Reverse Cuthill–McKee algorithm [22]. Each cluster is assigned a unique cluster ID that will be retained throughout the lifetime of a (potentially changing) cluster. A detailed overview of the algorithm is available in the supplementary material.

3.2 Geometric descriptor

From *pairs of primitives*, we generate descriptors that extend those proposed by BOWA [42]. Our descriptor $F \in \mathbb{R}^{10}$ is defined as

$$F(M, M') = \left(A, A', R, R', \mathbf{q}, \mathbf{t}\right). \tag{1}$$

It consists of geometric relationships between two primitives M and M', which are organized into two feature groups, characterizing the shapes of the primitives (A, A', R, R') and the transformation between them ($\mathbf{q}$, $\mathbf{t}$).

The first feature group describes the area and aspect ratio of each primitive's bounding rectangle:

$$A = \|\mathbf{u}\|_2 \cdot \|\mathbf{v}\|_2, \quad A' = \|\mathbf{u}'\|_2 \cdot \|\mathbf{v}'\|_2,$$
$$R = \frac{\max(\|\mathbf{u}\|_2, \|\mathbf{v}\|_2)}{\min(\|\mathbf{u}\|_2, \|\mathbf{v}\|_2)}, \quad R' = \frac{\max(\|\mathbf{u}'\|_2, \|\mathbf{v}'\|_2)}{\min(\|\mathbf{u}'\|_2, \|\mathbf{u}'\|_2)}. \tag{2}$$

The second group of features characterizes the transformation between the primitives using a translation $\mathbf{t} = \mathbf{p}' - \mathbf{p}$ and a rotation $\mathbf{q}$ stored as a unit quaternion (see Figure 3). To uniquely determine the rotation, we define a local coordinate system by assuming the origin is at $\mathbf{p}$; the z-axis is aligned with $\mathbf{n}$, and the y-axis is parallel to $\mathbf{v}$. We discard unreliable descriptors resulting from parallel primitives, coplanar primitives, or primitives with equal areas.

3.3 Descriptor matching

We establish correspondences between a reference model and the scene (i.e., the user's current scan) by measuring the similarity of their descriptors. We perform a series of measurements $\Delta = \{\delta_i\}_{i=1}^{7}$ and accept a model descriptor F_m and a scene descriptor F_s only if all δ_i are below predefined thresholds. The first four measurements are concerned with the areas and aspect ratios:

$$\delta_1 = \frac{|A_m - A_s|}{\max(A_m, A_s)}, \qquad \delta_2 = \frac{|A'_m - A'_s|}{\max(A'_m, A'_s)},$$
$$\delta_3 = \frac{|R_m - R_s|}{\max(R_m, R_s)}, \qquad \delta_4 = \frac{|R'_m - R'_s|}{\max(R'_m, R'_s)}. \tag{3}$$

The next two measurements concern the transformation between the pairs of primitives:

$$\delta_5 = |\,\|\mathbf{t}_m\|_2 - \|\mathbf{t}_s\|_2\,|, \qquad \delta_6 = |\,\mathbf{q}_m^\top \mathbf{q}_s\,|. \tag{4}$$

The last measurement involves the normals and the vector $\mathbf{t}$. We extract the normals $\mathbf{n} = [0,0,1]^\top$ and $\mathbf{n}' = \mathbf{q}\mathbf{n}\mathbf{q}^{-1}$ (see Figure 3) from the descriptors and determine an array $\mathbf{a}$:

$$\mathbf{a} = \left[\cos^{-1}(\mathbf{t}^\top \mathbf{n}),\ \cos^{-1}(\mathbf{t}^\top \mathbf{n}'),\ \cos^{-1}(\mathbf{n}^\top \mathbf{n}')\right]^\top. \tag{5}$$

This preparation lets us define the last measurement:

$$\delta_7 = \|\mathbf{a}_m - \mathbf{a}_s\|_2. \tag{6}$$

4 Localization

The first step for localization recognizes the place among the reference models stored in the database and the second performs pose estimation within the model. We use a vocabulary tree [39] for simultaneous model retrieval and correspondence search. The vocabulary tree is a scalable and efficient search structure for finding descriptors, in the database, that resemble those computed from the current set of primitives Ω, delivered by the scene understanding component of the XR platform. It rapidly returns a large number of candidate matches. We employ the verifications described in Section 3.3 to confirm the matches. For the search, we only use descriptors that relate primitives which come from different clusters. This measure allows us to avoid accepting primitives from clusters that moved in the scene. Since primitives that make up a rigid object (cluster) will move together, the descriptors computed from a cluster will remain identical even if the cluster's pose in the scene changes.

The vocabulary tree returns a sorted list of candidate models. The order of the list is determined using a relevance score [39]. For each candidate model, we perform a geometric verification [42], which consists of registering the candidate reference model to the user's current scan using the pose estimation method described in CWA [41]. The first candidate anchor which achieves at least three matched primitives and a low registration error is selected as the final solution, and the search stops.

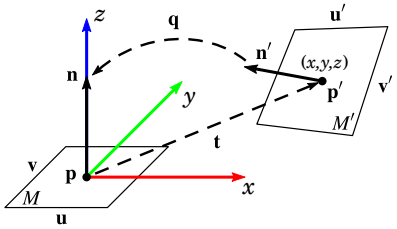

Figure 3: Local coordinate system to compute $\mathbf{q}$ and $\mathbf{t}$ (Equation 1).

Three matches of unchanged primitives is the theoretical minimum to perform a successful localization [41,42]. In the presence of extreme changes in the scene, three or more matches might be available. However, these matches might involve both, changed and unchanged primitives. Those matches will allow the correct anchor to fall into the sorted list of candidate models. Nevertheless, the geometric verification [42] between the reference model and the current set of primitives Ω will fail. Given the right reference model and at least two correspondences (retrieved from the vocabulary tree), we search for more correspondences leveraging the *primitive category* (see Section 3) provided by the XR platform.

To find more correspondences, we use the matched primitives to compute a 6DOF rough registration between the reference model and the scene (Ω). To do so, we use a series of intermediate coordinate systems. First, we must identify the *floor* in both, the reference model and the scene. If available, we rotate the primitives to align the normal of the floor with the z-axis (see Figure 4). This step solves two degrees of freedom (roll and pitch). Then, we translate the primitives so the floor plane lies on the xy-plane. After these two transformations, the plane equation of the floor should be [0 0 1 0]. This step solves another degree of freedom (translation along the z-axis). To solve the three remaining degrees of freedom (yaw and (x,y) translation), we use the xy-coordinates of the centroids of the matched primitives to build a system of linear equations, which we solve using least squares. Finally, we integrate these three transformations into a single matrix $\mathbf{T} = [\mathbf{R}\,|\,\mathbf{t}] \in \mathbf{SE}(3)$.

We use this transformation to roughly align the reference model to the scene, and perform a two-stage search of *vertical primitives*. First, we take the primitives labeled as *walls*, and we measure their normal deviation and plane-to-plane distance as

$$\alpha = \mathbf{n}_m^\top \mathbf{n}_s, \qquad \varepsilon = |\,\mathbf{n}_m^\top (\mathbf{p}_s - \mathbf{p}_m)\,|. \tag{7}$$

We consider that two wall primitives match if $\alpha > \theta$ and $\varepsilon < \tau$. Second, we take all the remaining *vertical primitives* which were not labeled as wall, and measure their normal deviation α and plane-to-plane distance ε.

Although we could also search for additional correspondences from *horizontal primitives*, the *floor* is sufficient to compute the 6DOF registration between the reference model and the user's current scan Ω (scene).

5 Change detection

In this section, we describe the proposed change detection method, which helps us to update the reference model. In summary, the steps to carry out the change detection are as follows:

1. Unchanged primitive correspondences search
2. Changed primitive correspondences search
3. Verification of correspondences

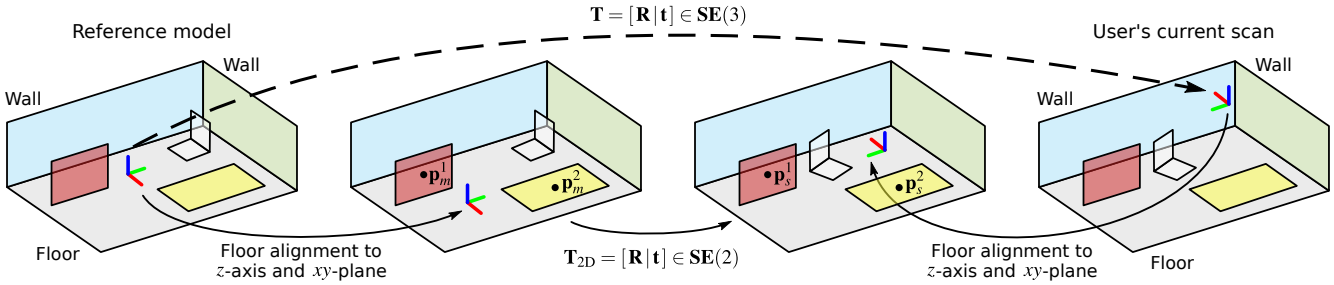

Figure 4: Intermediate coordinate systems to compute the rough registration used to search additional correspondences in the presence of extreme changes in the scene. The colors of the primitives depict correspondences. The *floor* is used to align the primitives with the z-axis and the xy-plane. The centroids $\mathbf{p}$ of the matched primitives (obtained through the place recognition procedure) are used to find the rotation about the z-axis and the (x,y) translation to align the reference model to the scene.

5.1 Unchanged primitive correspondences search

Strict thresholds are used for place recognition; thus, the set of matched primitives (obtained from the vocabulary tree) generally does not contain *all the unchanged primitives in the scene*. Therefore, we repeat the search for correspondences to find all unchanged primitives. The second search is limited to only the primitives in the selected reference model, which is a very small number compared to the entire content of the vocabulary tree. For this small number of primitives (typically 20–50), we exhaustively compare the descriptors from the scene and the descriptors in the reference model. To do so, we use the matching method described in Section 3.3, but this time we employ a set of *relaxed* thresholds.

We obtain a superset of matches that contains a mixture of unchanged primitives and borderline cases of primitives that have only moved slightly. To filter out the latter and retain only the unchanged primitives, we create a matrix that stores pairwise relations between the matched primitives. For any pair of primitives in the superset of matches, we compute a simpler geometric descriptor

$$E(M,M') = \Big(\|\mathbf{t}\|_2,\ \mathbf{a},\ |\mathbf{u}^\top \mathbf{u}'| \Big), \tag{8}$$

where $\mathbf{t} = \mathbf{p}' - \mathbf{p}$ (see Section 3.2), $\mathbf{a}$ is computed using Equation 5 and, $\mathbf{u}$ and $\mathbf{u}'$ are the shortest edge vectors of M and M', respectively.

We create a matrix of descriptors E_s for the scene and a matrix of descriptors E_m for the reference model. Then we perform an element-wise comparison. For each pair of elements, we compute the absolute differences between the corresponding features in the two descriptors. If any difference exceeds a feature-specific threshold, the elements are marked as invalid. If a primitive causes more than a certain number of invalid elements, it is removed from the set of matched primitives. We store the resulting unchanged primitives, along with the cluster ID (see Section 3.1) of the reference model, in a container $\mathcal{U}$.

5.2 Changed primitive correspondences search

To find the changed primitives, we reuse the matching method described in Section 3. However, we now apply it to the remaining primitives $\overline{\mathcal{U}} = \Omega \setminus \mathcal{U}$ (see Table 1) that were not successfully matched in the initial matching procedure. We suppress primitives in $\overline{\mathcal{U}}$ with an aspect ratio close to one (i.e., squares), which are easily confused due to their rotational invariance.

Recall that all primitives that make up a rigid object (cluster) will move together. Therefore, descriptors F (see Equation 1) that only involve primitives of a single changed cluster (rigid object) will remain close to identical, independently of the pose of the cluster. Thus, we can iterate over the clusters in $\overline{\mathcal{U}}$ to establish matches for the changed primitives. For all primitives in a cluster, we perform the correspondence verifications described in Section 3.3 and Section 5.1 to find matching changed primitives.

Table 1: List of definitions.

M	Primitive $M = (\mathbf{p}, \mathbf{n}, \mathbf{u}, \mathbf{v}, \mathbf{c})$ created from a set of coplanar points, comprised of its centroid $\mathbf{p}$, normal $\mathbf{n}$, the edge vectors of its bounding rectangle, $\mathbf{u}$ and $\mathbf{v}$, and its *primitive category* $\mathbf{c}$ (see Section 3).
Ω	Set of primitives scanned at runtime, delivered by the scene understanding component of an XR platform.
$\mathcal{U}$	List of *unchanged* primitives, i.e., *matched* primitives that have the same *pose* in both, the reference model (anchor) and the user's current scan (see Section 5.1).
$\overline{\mathcal{U}}$	List of *changed* primitives, i.e., *matched* primitives that have a different *pose* in the reference model (anchor) and the user's current scan (see Section 5.2).
Ω'	Combined list of *unchanged* and *changed* ($\mathcal{U} \cap \overline{\mathcal{U}}$) primitives.

Narrow objects, such as desks, doors, etc., often yield only a single primitive. Such *solo* primitives naturally do not belong to a cluster. Therefore, we run another exhaustive search for these remaining, unmatched solo primitives in $\overline{\mathcal{U}}$. We exhaustively search for matches using only the size features (see Equation 2), and we check only δ_1–δ_4 (see Equation 3). If this check is passed, a primitive is considered a matched primitive. Any remaining unmatched primitives in $\overline{\mathcal{U}}$ are removed. We do not check δ_5–δ_7 because these measurements are used to ensure that the relative pose between a given pair of primitives is the same in both, the reference model and the user's current scan (see Figure 3). In order to pass the check of δ_5–δ_7, a pair of solo primitives would need to be moved as a single rigid body, which is very unlikely to happen in real-world scenarios.

5.3 Verification of correspondences

After identifying the matching primitives, we perform an additional verification. We merge the results obtained so far into $\Omega' = \mathcal{U} \cap \overline{\mathcal{U}}$. Then we regenerate the clusters in Ω'. We run the clustering twice, once according to the primitives in the reference model and once according to the primitives in the scene. We search for solo primitives in the scene that appear in clusters in the reference model, and vice versa. These primitives must be incorrect matches, because the same primitive cannot appear solo and as part of a cluster. This criterion is true for both changed and unchanged clusters. Unchanged clusters additionally do not allow a changed solo primitive as a member, because changed and unchanged primitives cannot be mixed in the same cluster. We remove these offending primitives from Ω'. When using the strictest settings, we delete the entire offending cluster to ensure that no inconsistencies remain. An example of this verification step is available in the supplementary material.

6 Model Update

To perform the model update, we need to know the scene-to-model correspondences and the 6DOF transformation that aligns the reference model to the current scan of the scene. The correspondences

are contained in Ω' (see Table 1), and the transformation is determined as described in Section 4. With these preparations, the steps to update the reference model are as follows:

1. Remove missing primitives from the reference model
2. Update changed primitives in the reference model
3. Find new primitives in the scene
4. Serialize model updates

Remove missing primitives. We start the update of the model by removing the missing primitives from the reference model. We iteratively compare the clusters in the reference model with the primitives in Ω'. If a primitive of a given cluster cannot be found in the scene (it is not in Ω'), we mark it as *removed*. If all primitives of a given cluster are absent from the scanned scene, we mark the cluster as *deleted*.

Updating changed primitives. If correspondences of the changed primitives exist in Ω', we remove the old version of the primitives from the reference model and add the new version to the model. These changed primitives are organized into two subcategories: *removed* and *added*. The old version of the primitive is marked as *removed*, while the new one is *added*.

Finding new primitives in the scene. In the third step of the model update, we *attach* new primitives to existing clusters, and we *add* new clusters to the reference model. To perform this update, we need the transformation $\mathbf{T}$ that aligns the reference model to the current scan of the scene. Since we are looking to update the reference model, we need to apply the inverse transformation $\mathbf{T}^{-1}$ to the new scene primitives to register them with the reference model.

We create clusters from the new primitives and compare them with the clusters already present in the reference model. If a new cluster and an old cluster have primitives in common, we add only the new primitives to the reference model, inserting them into existing clusters as *attached* primitives. Otherwise, if an entirely new cluster is added to the reference model, its primitives are marked as *added*.

Serialization of model updates. We serialize all recorded operations concerning *removed*, *deleted*, *attached* and *added* primitives. Each record contains a unique primitive ID. This serialized representation can be kept as a record of the scene evolution, similar to version control systems.

7 Experimental Results

In the following, we evaluate the localization and change detection performance. We collected two model versions, A and B, of 15 different rooms, i.e., 30 models in total, with an iPad Pro. Version A of the scans was used as the reference model. We added several cardboard boxes as easy-to-move additional furniture and, otherwise, scanned the room as it presented itself. Version B of the scans was used to evaluate change detection and localization. In version B, we moved some of the clusters in the room (chairs, desks, etc.), including some of the boxes.

All core components of our algorithm (i.e., anchor creation, place recognition, pose estimation, change detection and anchor update) were implemented in C/C++. To fetch detected primitives on XR devices, we created an application using Unity software and the platform specific scene understanding component, for example, AR Foundation for iPad Pro and Apple Vision Pro.

7.1 Localization

We evaluated the model retrieval performance of a vocabulary tree created with primitives extracted from 3D reconstructions of real rooms in the ScanNet dataset (v2) [10, 11]. This dataset contains 3D reconstructions of 707 different rooms. Each 3D reconstruction is annotated. Therefore, it is rather straightforward to extract polygonal primitives from these scans (see Figure 5).

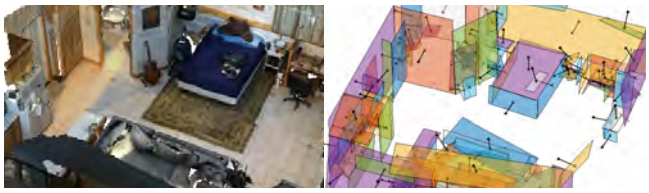

Figure 5: Example of a 3D reconstruction from the ScanNet dataset (v2) [10, 11] (left), and the geometric primitives we extracted from this reconstruction (right) using the *data preparation code* from PlaneRCNN [33, 34].

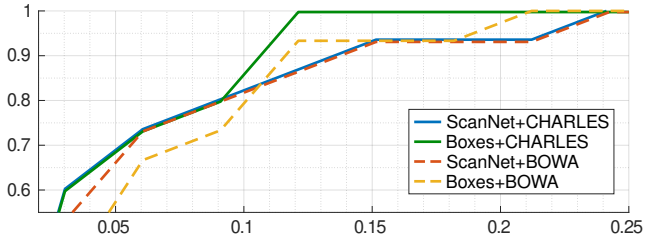

Figure 6: Anchor retrieval performance. The receiver operating characteristic (ROC) curves show the percentage (y-axis) of anchors that make it into the top x percent (x-axis) of all anchors stored in the tree. It is crucial that the corresponding correct anchor falls at the top of the result, as verification (see Section 4) can only be done for a fraction of the database when the database grows large. Hence, we are mainly interested in where the curves meet the y-axis.

We used the *data preparation code* from PlaneRCNN [33] to extract the primitives. PlaneRCNN is a method designed to detect planar surfaces from RGB images. Liu et al. [34] used a RANSAC [21] loop to fit primitives to the 3D reconstructions in the ScanNet dataset, which they used in turn to train their CNN model. Their *data preparation code* (RANSAC loop) provides the parametric equation of the primitives. To determine the bounding rectangle that includes all points in the supporting plane with minimal area, we used the annotations provided with the ScanNet dataset. Then, we used our system to create a vocabulary tree using these primitives, and populated the tree with the descriptors of the reference model of the 15 rooms of real environments that we captured with the iPad Pro.

For comparison, we created a vocabulary tree from purely synthetic room-sized models with an arbitrary number of box-shaped structures, and populated the inverted index of the tree with the same descriptors of the reference model of the 15 rooms.

According to Figure 6, the synthetic vocabulary tree can compete with the one using real data. The performance of the tree created with ScanNet+PlaneRCNN models is comparable to the performance of the tree created with artificial models. However, the synthetic set of models contains 1000 models, which produce 7.7 million descriptors, while the 707 ScanNet+PlaneRCNN models produce only 1.5 million descriptors.

Figure 6 also shows a comparison of the actual model retrieval performance using vocabulary trees created with BOWA descriptors and CHARLES. The tree created with CHARLES descriptors provides slightly better or on-par performance.

7.2 Change detection

To show the superiority of our new descriptor, we compare CHARLES with BOWA [42] while performing change detection in the 15 different room models. Figure 7 shows the results of this experiment with 15 groups of three bars each. The first bar shows the results obtained with BOWA, the second one shows the results obtained with CHARLES, and the third bar shows the ground truth.

Figure 7 reveals that both BOWA and CHARLES are able to find all unchanged primitives. However, BOWA marks some changed primitives as unchanged. Figure 8 shows the detailed correspondences in room number 1. The cyan, orange and yellow primitives

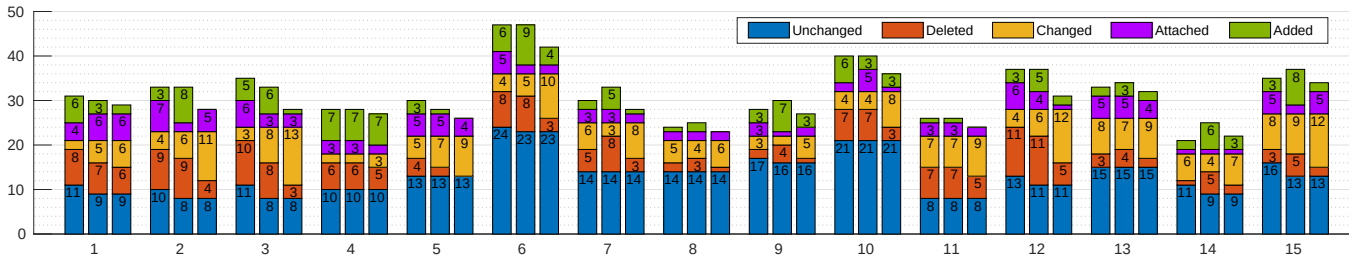

Figure 7: Evaluation of the change detection performance using BOWA [42] and CHARLES for 15 different scenes (x-axis). For each scene, we show a group of three bars. The first bar shows the results obtained with BOWA. The second bar shows the results obtained with CHARLES. The third one shows the ground truth. The height of the blue part of the bar shows the number of unchanged primitives; red shows the deleted primitives; yellow shows the changed primitives; purple shows the attached primitives; green shows the added primitives. The sum of unchanged, deleted and changed primitives is the same for each group of bars, because we mark a primitive as deleted (and added), if we are not able to detect a change. Note that, with CHARLES, we always found the correct amount of unchanged primitives, while BOWA marks some changed primitives as unchanged (e.g., compare the first and second bars in groups 1 and 15). This shows the superior localization capabilities of CHARLES over BOWA.

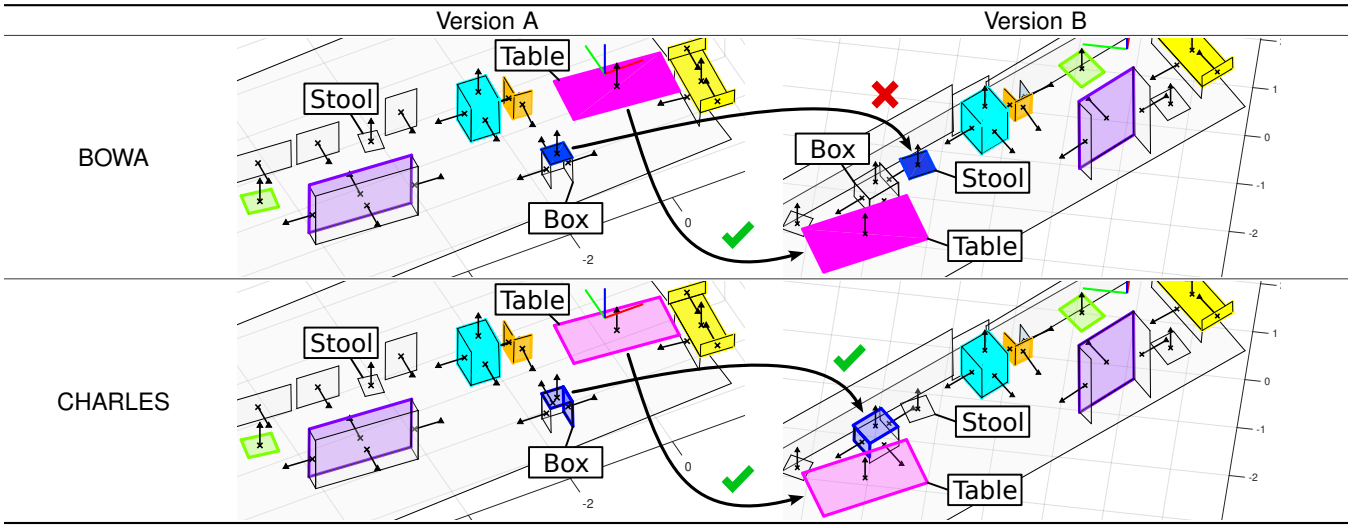

Figure 8: Example of change detection. The colors denote correspondences. The cyan, orange and yellow primitives are detected as unchanged, and the green and purple as changed when using both BOWA and CHARLES. With BOWA, the pink and deep blue primitives are detected as unchanged. The table correspondence is right while the deep blue correspondence is not. With CHARLES, we correctly assign the correspondences and detect the changes in the scene.

did not move between versions A and B, but the green, purple, blue and pink primitives did move. Note that the position and orientation between the *table* and the top of the *box* in version A are the same as the pose between the *table* and the *stool* in version B. In addition, the size of the top part of the *box* and the *stool* are comparable. In the first row of Figure 8, the table is considered unchanged when using BOWA, and the top part of the box is matched to the stool. The second row of Figure 8 shows that it is possible to find the actual corresponding primitives using CHARLES.

7.3 Long-term localization

The ability to detect changes in the scene and to update the reference model is of key importance to enable long-term applications. We conducted a simulation using the model (version B) of the 15 rooms we captured with the iPad Pro. We divided each model into two sets of primitives, changed and unchanged (see Table 2). The former consisted of all lightweight objects whose pose is very likely to change between sessions, like chairs, doors, boxes, etc. While the latter included big heavy objects like desks or bookshelves. For each room, we simulated a 50–day period of time; in other words, 50 independent AR sessions. Each day, we applied a random transformation to all the primitives in the reference model in order to create an artificial user's current scan (scene). And finally, we added Gaussian noise to all the primitives in the scene, changed and unchanged, to simulate sensor noise. We computed the 6DOF registration between the reference model and the artificial scene, and we measured the position and rotation errors as

$$e_R = \text{acos}\left(\frac{1}{2}(\text{Tr}(\mathbf{R}^\top \mathbf{R}_{\text{GT}}) - 1)\right), \qquad e_t = \|\mathbf{t} - \mathbf{t}_{\text{GT}}\|. \quad (9)$$

After each day (session), the artificial scene became the new reference model to incorporate the changed primitives. This procedure somewhat reflects a situation of updating a reference model of a room with slight random changes on a daily basis for 50 days.

Figure 9 shows the average of the errors e_R and e_t on the 15 rooms, for three different levels of noise. We measured the error w.r.t. the original reference model (absolute), and the error w.r.t. the reference model from the immediate previous trial (relative). While the former increased slowly, the latter remained very small during the 50–day period of time. The continuously increasing absolute error can be attributed to the noise we applied to the primitives in the scene, which was stored in the new model. Details about the rooms and additional results are available in the supplementary material.

Table 2 shows that, in average, 25% of the primitives in the scene remained unchanged. In rooms 9 and 10, we could successfully relocalize even when only 15% of the primitives remained unchanged. In general, three matches of unchanged primitives are required to

Table 2: Characteristics of the test rooms used for the long-term localization simulations.

Room	Size [m^2] (w[m]×ℓ[m])	No. primitives	Change percentage
1	20.35 (9.0×2.3)	23	69.57% (16/23)
2	14.95 (4.5×3.4)	29	82.76% (24/29)
3	26.85 (8.5×3.1)	25	72.00% (18/25)
4	41.61 (7.0×6.0)	27	59.26% (16/27)
5	16.48 (4.8×3.4)	29	62.07% (18/29)
6	21.05 (4.7×4.5)	45	68.89% (31/45)
7	39.84 (8.0×5.0)	32	78.12% (25/32)
8	75.76 (11.5×6.6)	35	68.57% (24/35)
9	28.06 (5.6×5.0)	47	85.11% (40/47)
10	27.86 (6.1×4.6)	34	85.29% (29/34)
11	15.72 (4.6×3.4)	21	76.19% (16/21)
12	19.89 (6.1×3.3)	30	86.67% (26/30)
13	23.03 (5.6×4.1)	29	79.31% (23/29)
14	15.34 (4.5×3.4)	20	65.00% (13/20)
15	25.94 (6.0×4.3)	31	80.65% (25/31)
AVG.	27.52 (6.4×4.2)	23	74.63%

perform relocalization [41,42]. However, one can think about scenarios of failure. For example, if most of the clusters in the scene move symmetrically, the place recognition would work but the pose estimation would fail.

7.4 On-device runtime

We measured the runtime of the proposed method on two popular XR devices: iPad Pro and Apple Vision Pro. For brevity, we abstain from discussing further results for other platforms such as Microsoft's HoloLens 2 or Magic Leap 2. However, the algorithm is fully cross-platform, and runtimes for those platforms are similar, as previously shown in BOWA [42].

Table 3 shows the average runtime for the same 15 rooms we used for the other evaluations. The total file size of the 15 anchors is 2.1 MB, and the approximate size of the vocabulary tree is 89 MB. Loading the vocabulary tree takes a significant amount of time (∼40 ms), however, this is a one-time operation upon start-up. The change detection runtime is substantial (∼190 ms) as well due to the exhaustive correspondence search (see Section 5.1). Overall, the method is fast enough to provide a seamless XR user experience. Details on the 15 rooms are given in the supplementary material.

7.5 Application examples

Besides the situated use case, where data is explicitly recorded in a given location to be used later on for relocalization, our method is suitable for integration with existing spatial databases. For example, building information models require detailed digital representations of buildings. In addition, sectors such as real estate transitioned to virtual tours of properties based on 3D scans, reconstructions, and localized photo-spheres. Structure-from-motion and AI-based methods allow accurate 3D reconstruction based on spherical images [28]. Such existing reconstructions can be used to enable XR applications such as situated room tours with virtual avatars.

We propose a workflow to enable relocalization using 3D representations (mesh, point cloud, or similar) as the reference model. Since our localization method uses geometric primitives as input, we propose the use of ubiquitous software tools to extract primitives from such 3D data. Using CloudCompare [1], we load a 3D reconstruction and resample it to obtain a uniform point distribution. Resampling helps to remove outliers and sensor-induced artifacts. Additionally, other 3D manipulations can be applied, for example, repositioning, clipping or filtering. To extract geometric primitives, we use the RANSAC shape detection plugin of CloudCompare [44]. Since our main focus lies on localization on mobile devices, we extract only planar primitives. This procedure can be scripted in

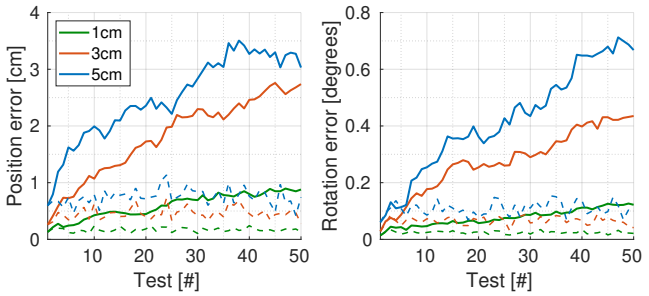

Figure 9: Results of the long-term simulations on 15 different rooms, for three levels of noise. We measured the *world origin* estimation error over 50 consecutive trials. The continuous lines show the *absolute* error, i.e., the error w.r.t. the original reference model, while the dashed lines show the *relative* error, i.e., the error w.r.t. the reference model from the immediate previous trial.

Table 3: Mean and median runtimes, in *ms*, of the proposed method on an iPad Pro and an Apple Vision Pro.

	iPad Pro		Apple Vision Pro	
Step	mean	median	mean	median
1. Vocabulary tree loading	41.41	41.38	37.31	37.24
2. Localization	38.14	37.24	30.51	30.19
3. Change detection	238.67	187.73	135.05	105.75
4. Model update	1.48	1.04	1.43	0.88

CloudCompare to automate the extraction of primitives from 3D representations. Finally, we label the primitives as floor, wall, etc. This step can be done with modern scene understanding methods such as ConceptGraphs [26].

Figure 10 shows a 3D reconstruction captured with a Matterport Pro3 camera [35]. This reconstruction covers the entire ground floor of a building. We extracted primitives from five different rooms using the described workflow. We created an anchor for each room, and saved them on an iPad Pro and a Apple Vision Pro. We were able to successfully relocalize in all rooms with CHARLES on both devices (see Figure 11), despite significant changes to the environment from the time of data collection (2023) to the time of relocalization (August, 2025).

7.6 Discussion

Pose refinement. Currently, we limit the implementation of our method to a single-time localization at the start of the application. In the reported experiments, we used scans of the entire test scenes. A continuous re-estimation of the device's pose throughout the XR session could lead to a higher accuracy result. However, an evaluation on the effects of gradual discovery of new primitives in the scene is beyond the scope of our work.

Extreme changes in the scene. Many clusters of primitives whose pose change in the scene between sessions, or clusters that move symmetrically might lead to wrong place recognition and pose estimation. As proposed by Cupec et al. [7], an approach to solve this problem would be to use features extracted from RGB images, for example, to detect and read signs in the scene.

Multi-instance changes. It is worth noting that our scene representation constitutes a rather high layer of abstraction; therefore, our change detection method is not capable of discerning between multi-instance changes. However, the end-user should not notice this change, since the change detection and anchor update should be transparent to the user. In case the reference model relates digital content to geometric primitives that can change (e.g., in a BIM representation), it would be necessary to adapt our method to be able to correctly update the digital content as well.

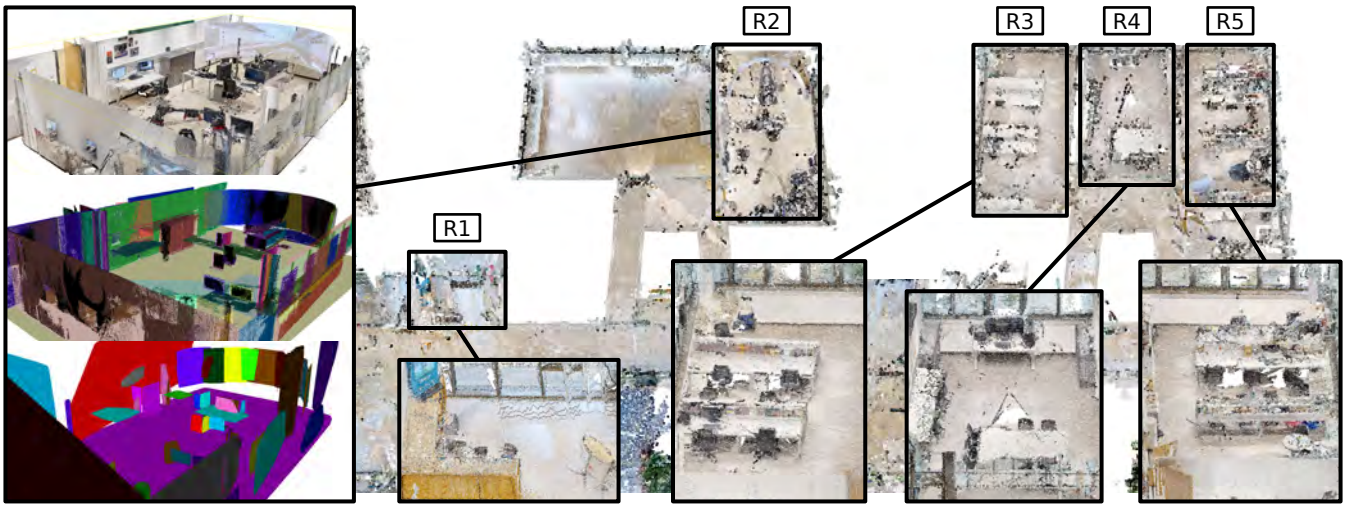

Figure 10: Reconstruction captured with a Matterport Pro3 camera [35]. Rooms R1–R5 show the scenes we used to test the workflow we propose in Section 7.5. In the left of the figure, we show a close-up of the 3D reconstruction of room R2 (top), the primitives extracted with CloudCompare [1, 44] (middle), and the primitives captured with ARKit on an iPad Pro (bottom).

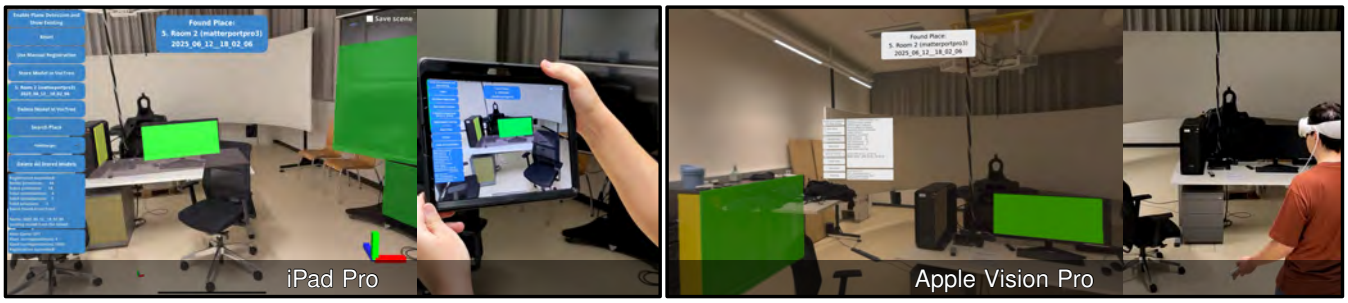

Figure 11: Example of relocalization running on an iPad Pro and an Apple Vision Pro, in room R2 (see Figure 10). Note that, we employed the workflow described in Section 7.5 to generate the anchors from the 3D reconstruction in Figure 10.

Input data. We do not use RGB images, RGB-D data or dense depth maps to compute our compact scene representation. Commercial tools like AR Foundation or Microsoft's Scene Understanding SDK might use RGB images and on-device motion sensors to perform plane detection. However, our method is capable to work on top of approaches that estimate primitives from depth data [45, 46], as previously shown in CWA [41]. Since we use the primitive category to perform localization in the presence of extreme changes in the scene, certain modifications would be needed to compensate the lack of this feature.

Primitive types. XR frameworks currently only provide planar primitives. Therefore, we focused on this type in this paper. However, previous work [45, 46] demonstrated that it is possible to detect primitives such as spheres or cylinders as well. Our approach can incorporate such geometric primitives, following certain modifications, as previously shown in BOWA [42].

Comparison with existing anchor systems. Existing anchor tools like ARKit [2] or Azure spatial anchors [38], do not exactly reveal the data sources they use to perform global localization. Very likely, the use the Wi-Fi connection, visual features from images, motion sensors, etc. Besides, the pose estimation algorithms are unknown as well. Therefore, it would be rather difficult to perform a systematic comparison of world origin deviations between our method and these tools. On the one hand, our method would outperform existing anchor systems on textureless scenarios. On the other hand, the latter would outperform our method in scenes where everything has changed and unique landmarks can be identified on the walls, the floor or the ceiling.

8 Conclusion

In this work, we presented an approach to perform change-resilient localization based on geometric primitives. Our method allows us to successfully detect changes in the scene using the geometric relationships between clusters of primitives. It can contribute to solving the localization problem in changing environments for commercial XR platforms. The experimental results show its suitability for the creation and maintenance of spatial anchors in small and medium sized indoor environments.

Future work will focus on the use of CHARLES for large-scale localization in industrial environments, primarily using existing 3D reconstructions built from, for example, Matterport scans. The widely used Azure spatial anchors for the HoloLens are being discontinued by Microsoft, and limited alternatives for cross-platform, reliable localization exist. Therefore, we believe that CHARLES is important for further research and development. Although the results presented are promising, additional efforts are required to integrate productive XR systems with existing scans through CHARLES as the underlying localization engine.

Acknowledgments

This work was enabled by the Competence Centre VRVis, the Alexander von Humboldt Foundation and the European Union under Grant Agreement No. 101092861. The VRVis GmbH is funded by BMIMI, BMWET, Tyrol, Vorarlberg and Vienna Business Agency in the scope of COMET – Competence Centers for Excellent Technologies (911654) which is managed by FFG. The Alexander von Humboldt Foundation is funded by the German Federal Ministry of Research, Technology and Space, the German Research Foundation *DFG* (grants 528364066, 495135767, 390831618) and the Austrian Science Fund *FWF* (grant I6663).

REFERENCES

[1] CloudCompare - Open Source Project. https://www.danielgm.net/cc/. Accessed: 2025-07-12. 2, 8, 9

[2] Apple. ARKit – ARAnchor. https://developer.apple.com/documentation/arkit/aranchor. Accessed: 2025-07-12. 1, 9

[3] Apple. ARKit – Plane Detection. https://developer.apple.com/documentation/arkit/tracking-and-visualizing-planes. Accessed: 2025-07-12. 1

[4] 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. on Robotics*, 37(6), 2021. 3

[5] E. Carra and F. Pellacini. SceneGit: A practical system for diffing and merging 3D environments. *ACM Trans. Graph.*, 38(6), 2019. 2

[6] L. Cui and C. Ma. SOF-SLAM: A semantic visual SLAM for dynamic environments. *IEEE Access*, 7, 2019. 3

[7] R. Cupec, D. Filko, and E. K. Nyarko. Place recognition based on planar surfaces using multiple RGB-D images taken from the same position. In *European Conf. on Mobile Robots (ECMR)*, 2019. 3, 8

[8] R. Cupec, E. K. Nyarko, D. Filko, A. Kitanov, and I. Petrović. Place recognition based on matching of planar surfaces and line segments. *The Int. Journal of Robotics Research*, 34(4-5), 2015. 3

[9] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, 2023. 2

[10] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner. ScanNet. https://github.com/ScanNet/ScanNet. Accessed: 2025-07-12. 2, 6

[11] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner. ScanNet: Richly-annotated 3D reconstructions of indoor scenes. In *IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2, 6

[12] B. Deng, Y. Yao, R. M. Dyke, and J. Zhang. A survey of non-rigid 3D registration. *Computer Graphics Forum*, 41(2), 2022. 3

[13] J. D. Denning and F. Pellacini. MeshGit: Diffing and merging meshes for polygonal modeling. *ACM Trans. Graph.*, 32(4), 2013. 2

[14] J. D. Denning, V. Tibaldo, and F. Pellacini. 3DFlow: Continuous summarization of mesh editing workflows. *ACM Trans. Graph.*, 34(4), 2015. 2

[15] N. Donati, A. Sharma, and M. Ovsjanikov. Deep geometric functional maps: Robust feature learning for shape correspondence. In *IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020. 2

[16] R. Dubé, D. Dugas, E. Stumm, J. Nieto, R. Siegwart, and C. Cadena. SegMatch: Segment based place recognition in 3d point clouds. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2017. 3

[17] M. Fehr, F. Furrer, I. Dryanovski, J. Sturm, I. Gilitschenski, R. Siegwart, and C. Cadena. TSDF-based change detection for consistent long-term dense reconstruction and dynamic object discovery. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2017. 2

[18] T. Feigl, A. Porada, S. Steiner, C. Loeffler, C. Mutschler, and M. Philippsen. Localization limitations of ARCore, ARKit, and Hololens in dynamic large-scale industry environments. In *Int. Joint Conf. on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP)*, 2020. 3

[19] E. Fernandez-Moral, W. Mayol-Cuevas, V. Arevalo, and J. Gonzalez-Jimenez. Fast place recognition with plane-based maps. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2013. 3

[20] E. Fernandez-Moral, P. Rives, V. Arevalo, and J. Gonzalez-Jimenez. Scene structure registration for localization and mapping. *Robotics and Autonomous Systems*, 75, 2016. 3

[21] M. A. Fischler and R. C. Bolles. Random Sample Consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6), 1981. 6

[22] A. George and J. W. Liu. *Computer Solution of Large Sparse Positive Definite*. Prentice Hall Professional Technical Reference, 1981. 3

[23] D. Girardeau-Montaut, M. Roux, R. Marc, and G. Thibault. Change detection on points cloud data acquired with a ground laser scanner. *ISPRS Archives*, 36(3), 2005. 2

[24] C. Gomez, A. C. Hernandez, R. Barber, and C. Stachniss. Localization exploiting semantic and metric information in non-static indoor environments. *Journal of Intelligent & Robotic Systems (JIRS)*, 109(4), 2023. 3

[25] Google. ARCore – Cloud Anchors. https://developers.google.com/ar/develop/anchors. Accessed: 2025-07-12. 1

[26] Q. Gu, A. Kuwajerwala, S. Morin, K. M. Jatavallabhula, B. Sen, A. Agarwal, C. Rivera, W. Paul, K. Ellis, R. Chellappa, C. Gan, C. M. de Melo, J. B. Tenenbaum, A. Torralba, F. Shkurti, and L. Paull. ConceptGraphs: Open-vocabulary 3D scene graphs for perception and planning. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2024. 8

[27] J. He, M. Li, Y. Wang, and H. Wang. OVD-SLAM: An online visual SLAM for dynamic environments. *IEEE Sensors Journal*, 23(12), 2023. 3

[28] S. Jiang, K. You, Y. Li, D. Weng, and W. Chen. 3D reconstruction of spherical images: A review of techniques, applications, and prospects. *Geo-spatial Information Science*, 27(6), 2024. 8

[29] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *IEEE Int. Symposium on Mixed and Augmented Reality (ISMAR)*, 2007. 2

[30] J. Li, X. Pan, G. Huang, Z. Zhang, N. Wang, H. Bao, and G. Zhang. RD-VIO: Robust visual-inertial odometry for mobile augmented reality in dynamic environments. *IEEE Trans. on Vis. and Computer Graphics (TVCG)*, 30(10), 2024. 3

[31] Y. Li and T. Harada. Lepard: Learning partial point cloud matching in rigid and deformable scenes. In *IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022. 3

[32] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common objects in context. In *European Conf. on Computer Vision*, 2014. 2

[33] C. Liu, K. Kim, J. Gu, Y. Furukawa, and J. Kautz. PlaneRCNN. https://github.com/NVlabs/planercnn. Accessed: 2025-07-12. 2, 6

[34] C. Liu, K. Kim, J. Gu, Y. Furukawa, and J. Kautz. PlaneRCNN: 3D plane detection and reconstruction from a single image. In *IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 4445–4454, 2019. 2, 6

[35] Matterport. Pro3 Camera. https://matterport.com/pro3. Accessed: 2025-07-12. 2, 8, 9

[36] Microsoft. Scene Understanding. https://learn.microsoft.com/en-us/windows/mixed-reality/design/scene-understanding. Accessed: 2025-07-12. 1

[37] Microsoft. Scene Understanding – SceneObjectKind. https://learn.microsoft.com/en-us/windows/mixed-reality/develop/unity/scene-understanding-sdk#sceneobjects. Accessed: 2025-07-12. 3

[38] Microsoft. Spatial Anchors. https://learn.microsoft.com/en-us/windows/mixed-reality/design/spatial-anchors. Accessed: 2025-07-12. 1, 9

[39] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, 2006. 4

[40] Y. Qiu, S. Yamamoto, R. Yamada, R. Suzuki, H. Kataoka, K. Iwata, and Y. Satoh. 3D change localization and captioning from dynamic scans of indoor scenes. In *IEEE Winter Conf. on Applications of Computer Vision*, 2023. 2

[41] F. Reyes-Aviles, P. Fleck, D. Schmalstieg, and C. Arth. Compact World Anchors: Registration using parametric primitives as scene description. *IEEE Trans. on Vis. and Computer Graphics (TVCG)*, 29(10), 2022. 1, 4, 8, 9

[42] F. Reyes-Aviles, P. Fleck, D. Schmalstieg, and C. Arth. Bag of world anchors for instant large-scale localization. *IEEE Trans. on Vis. and Computer Graphics (TVCG)*, 29(11), 2023. 1, 3, 4, 6, 7, 8, 9

[43] E. Rinaldi, D. Sforza, and F. Pellacini. NodeGit: Diffing and merging node graphs. *ACM Trans. Graph.*, 42(6), 2023. 2

[44] R. Schnabel and R. Wahl. RANSAC Shape Detection (plugin). https://www.cloudcompare.org/doc/wiki/index.php/RANSAC_Shape_Detection_(plugin). Accessed: 2025-07-12. 8, 9

[45] C. Sommer, Y. Sun, E. Bylow, and D. Cremers. PrimiTect: Fast continuous hough voting for primitive detection. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2020. 9

[46] A. Stanescu, P. Fleck, C. Arth, and D. Schmalstieg. Semantic segmentation of geometric primitives in dense 3D point clouds. In *IEEE Int. Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, 2018. 9

[47] L. Suo, B. Wang, L. Huang, X. Yang, Q. Zhang, and Y. Ma. VoxelPlane-Reloc: An indoor scene voxel plane relocalization algorithm. *Complex & Intelligent Systems*, 10(3), 2024. 3

[48] Unity. AR Foundation – PlaneClassification. https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@6.2/api/UnityEngine.XR.ARSubsystems.PlaneClassification.html. Accessed: 2025-07-12. 3

[49] J. Ventura, C. Arth, G. Reitmayr, and D. Schmalstieg. Global localization from monocular SLAM on a mobile phone. *IEEE Trans. on Vis. and Computer Graphics (TVCG)*, 20(4), 2014. 2

[50] J. Vincent, M. Labbé, J.-S. Lauzon, F. Grondin, P.-M. Comtois-Rivet, and F. Michaud. Dynamic object tracking and masking for visual SLAM. In *Int. Conf. on Intelligent Robots and Systems*, 2020. 3

[51] Vuforia. Area Targets. https://developer.vuforia.com/library/vuforia-engine/environments/area-targets/area-targets/. Accessed: 2025-07-12. 1

[52] Y. Wang, Y. Zhang, L. Hu, W. Wang, G. Ge, and S. Tan. A semantic topology graph to detect re-localization and loop closure of the visual simultaneous localization and mapping system in a dynamic environment. *Sensors*, 23(20), 2023. 3

[53] J. Wietrzykowski. PlaneLoc2: Indoor global localization using planar segments and passive stereo camera. *IEEE Access*, 10, 2022. 3

[54] C. Xu, R. Kumaran, N. Stier, K. Yu, and T. Höllerer. Multimodal 3D fusion and in-situ learning for spatially aware AI. In *IEEE Int. Symposium on Mixed and Augmented Reality (ISMAR)*, 2024. 2

[55] C. Yu, Z. Liu, X.-J. Liu, F. Xie, Y. Yang, Q. Wei, and Q. Fei. DS-SLAM: A semantic visual SLAM towards dynamic environments. In *Int. Conf. on Intelligent Robots and Systems*, 2018. 3

[56] S. Yue, Z. Wang, and X. Zhang. DSOMF: A dynamic environment simultaneous localization and mapping technique based on machine learning. *Sensors*, 24(10), 2024. 3

[57] L. Zhang, A. Agrawal, S. Oney, and A. Guo. VRGit: A version control system for collaborative content creation in virtual reality. In *ACM Conf. on Human Factors in Computing Systems (CHI)*, 2023. 2