Point cloud change detection in indoor environments

Tomoya Matsubara[1,*], Hideo Saito[1]

[1]Keio University, Yokohama, Japan

Abstract

3D change detection plays a crucial role in a wide range of applications, including disaster management, as well as in robotics for search, rescue, security, and surveillance purposes. Although previous works exist, most of them are limited to detecting a few specific targets or are restricted to 2D images. Additionally, some assume prior knowledge of the object positions of interest. This paper presents a novel change detection algorithm that combines panoptic segmentation and k-NN, enabling the detection of changes without relying on positional information about the objects of interest. Experimental evaluations on indoor point clouds demonstrate the algorithm's capability to detect the removal of densely and closely placed objects, an aspect overlooked by previous approaches due to their inherent limitations. Despite variations in settings and datasets, our algorithm achieves a recall improvement of 0.06 for the *removed* class, surpassing the performance of existing related works.

Keywords

metaverse, point cloud, change detection, panoptic segmentation, k-Nearest Neighbor

1. Introduction

Point clouds, which accurately capture the 3D geometry of scenes, have extensive applications in scene understanding and robotics [1], encompassing tasks such as 3D shape classification [2, 3], 3D object detection [4, 5, 6, 7], and point cloud segmentation [8]. In the realm of the metaverse, point clouds frequently serve as representations of scenes within virtual worlds.

The metaverse is built upon immersive user experiences, necessitating an interaction layer that effectively bridges the physical and virtual worlds [9]. Digital twins [10] serve as a critical component within this layer, facilitating the transmission and synchronization of data and information between the virtual and physical worlds. However, the constant scanning of the entire scene to update the digital twin is impractical due to the vast amount of data involved. Thus, the selective updating of the digital twin in areas where changes have occurred becomes paramount, highlighting the essential role of change detection.

2D change detection techniques, which primarily focus on comparing two input images, have been proposed primarily for remote sensing applications [11, 12]. However, these approaches are constrained by the requirement of image alignment between the two inputs. In contrast, 3D change detection has garnered attention for its ability to overcome this limitation. Although some research has been conducted on 3D change detection in domains such as disaster management [13], security patrols [14], and real-time robotic applications [15], the field is still in its early stages and has limitations concerning the detection targets.

In this study, we present a novel 3D change detection algorithm that leverages panoptic segmentation and k-Nearest Neighbor (k-NN). The main contributions of our work are as follows:

- The proposed algorithm surpasses the limitations of 2D change detection techniques by effectively detecting changes that were previously challenging to address.
- Our approach eliminates the need for prior information on the objects of interest, making it versatile and applicable in various scenarios.
- Notably, our algorithm demonstrates its effectiveness in indoor environments, where objects are densely situated in close proximity. This aspect has often been overlooked in related studies due to the constraints of existing algorithms.

2. Related Work

3D change detection has been studied by various methods; however, these methods often suffer from limitations in their applicability. Some approaches necessitate prior knowledge of the object positions of interest [16], while others are confined to detecting a limited number of targets [17, 7]. Additionally, certain techniques are only suitable for change detection in 2D images [18]. The process of collecting positional information can be labor-intensive, particularly in the case of 3D data, as it involves annotation. The range and diversity of detection targets directly impact the algorithm's applicability.

S. Nikoohemat et al. [17] concentrate on changes in building geometry and propose a change detection algorithm that combines 2D and 3D nearest points. However,

APMAR'23: The 15th Asia-Pacific Workshop on Mixed and Augmented Reality, Aug. 18-19, 2023, Taipei, Taiwan

*Corresponding author.

tomoya.matsubara@hvrl.ics.keio.ac.jp (T. Matsubara); hs@keio.jp (H. Saito)

their algorithm's applicability is limited as it only considers vertical planes as potential objects of interest that may undergo changes. M. Voelse et al. [7] introduce change detection as a means to distinguish between static and temporal objects when creating updated 3D models of environments. The algorithm initiates by segmenting point clouds using region growing, with a set distance threshold of 10 cm. Additionally, segments with a height below 20 cm are excluded (i.e., discarded) to mitigate misclassifications. These defined thresholds make it challenging to apply the algorithm in environments where objects are closely positioned, such as indoor scenarios.

T. Ku et al. [16] propose three distinct algorithms, namely PoChaDeHH, HGI-CD, and SiamGCN, for change detection on a street-scene dataset consisting of point clouds. The dataset encompasses various street furniture objects, including road signs, advertisements, statues, and garbage bins, with the positions of each object of interest provided, facilitating the extraction of these objects from the point cloud. PoChaDeHH initially eliminates outliers and noisy objects from the extracted point cloud, then employs clustering techniques to separate the remaining objects. The change is estimated based on the mean distance between points in the registered point clouds. HGI-CD utilizes statistical techniques to remove outliers, constructs color and geometric change graphs using the k-NN algorithm, and estimates change using Siamese graph convolutional networks (GCNs) with Fast Point Feature Histograms (FPFH) [19] as the node features. SiamGCN also employs GCNs with graphs constructed through k-NN, but does not include a cleaning step for the extracted point cloud.

As a change detection algorithm that works without prior information of the positions of objects of interest, K. Sakura et al. [18] propose two deep learning models, namely CSCDNet and SSCDNet. CSCDNet, a Siamese network based on ResNet-18 [20], estimates the probability mask of change from two input images. SSCDNet, on the other hand, is a U-Net-based network that utilizes the input images and the output of CSCDNet to predict semantic change labels for each pixel. While the networks can be trained with semantic labels from non-aligned images, during inference, two aligned images are required. If the input images are not aligned during inference, the models would erroneously detect changes because many pixels in the input images represent different objects that did not undergo change but appear different. Similarly, W. G. C. Bandara and V. M. Patel [11] propose ChangeFormer for 2D change detection, which has demonstrated state-of-the-art performance on LEVIR-CD [21] and DSIFN-CD [22]. However, it faces the same challenge mentioned above, where the alignment of input images during inference is crucial to avoid misinterpretation of changes.

3. Methods

Let P_t and $P_{t'}$ denote a pair of already registered point clouds captured at different times t and t' (where $t \neq t'$), respectively, of the same scene. Given that an object instance exists in P_t but not in $P_{t'}$, we define this scenario as the instance being *removed* when $t < t'$, or *added* when $t > t'$. Therefore, the addition and removal of an object instance can be treated equivalently by interchanging the time values t and t'. Accordingly, we formulate the task of change detection as a binary classification problem, distinguishing between *no change* and *removed*, with a particular focus on identifying the specific object instance that underwent the change in the point cloud P_t.

First, 3D point clouds of the scene are reconstructed from captured images. Subsequently, panoptic segmentation is applied to assign an object instance label to each pixel in the images, thereby associating them with the corresponding points in the point clouds. In the next step, partial point clouds containing the same object instance are extracted from P_t, while their bounding volumes are utilized to extract the corresponding point clouds from $P_{t'}$. Finally, the pair of extracted point clouds undergo classification using the k-NN algorithm, determining whether there is *no change* or the instance has been *removed*. An overview of the proposed change detection algorithm is depicted in Figure 1.

The details are described in this section.

3.1. Point Cloud Reconstruction

The proposed algorithm for change detection relies on the comparison of two point clouds. These point clouds are reconstructed using RGB-D images captured by an iPhone running ARKit, along with the corresponding confidence maps and camera parameters (i.e., intrinsic and extrinsic).

The confidence maps, provided by ARKit, are 2D arrays with the same dimensions as the depth map. They take three values: *low*, *medium*, and *high*, which indicate the accuracy of the depth values. To optimize computational efficiency and enhance performance, pixels with *low* or *medium* confidence values are excluded and not utilized in the reconstruction process.

By utilizing the camera parameters and depth map, the position of each pixel in the 3D world coordinate system can be uniquely computed. The RGB images are employed to extract color information, while the instance labels are obtained through segmentation, as discussed in Subsection 3.2.

3.2. Panoptic Segmentation

To perform panoptic segmentation on the RGB images, we employ Detectron2 [23], a pre-trained deep learn-

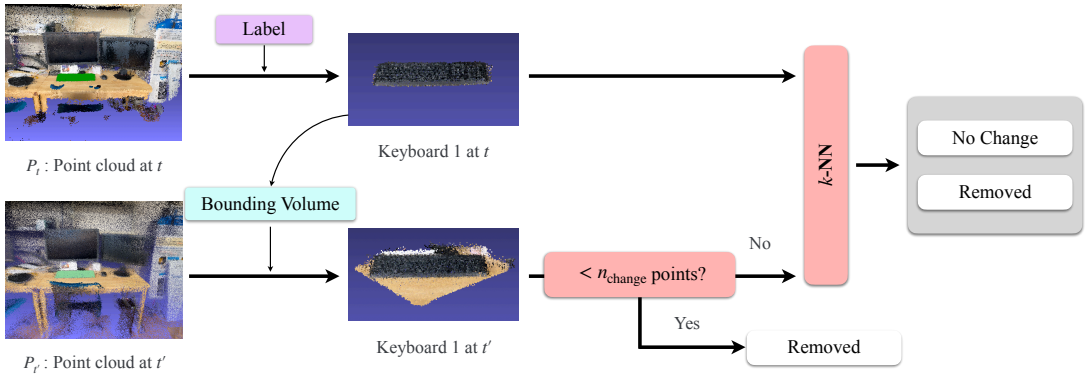


Figure 1: Overview of the change detection algorithm. KEY – n_{change}: threshold for the number of points to predict immediately *removed*.

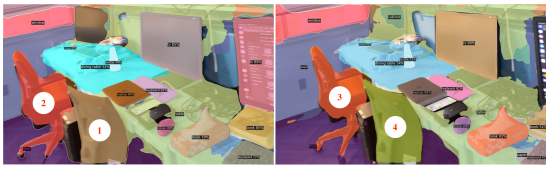


Figure 2: Instance labels obtained from panoptic segmentation. chair_1 (resp. chair_2) of the left image is the same instance as chair_4 (resp. chair_3) in the right image.

ing model that assigns an instance label to each pixel. The instance label takes the form of *object_id* (e.g., keyboard_10)), where *object* is a character string indicating the type of object represented by the pixel, and *id* denotes the instance's unique identifier.

It is important to note that the uniqueness of instance labels extends not only within each image but also across all images used for point cloud reconstruction. Consequently, different instance labels in two separate images may correspond to the same underlying object (as illustrated in Figure 2). Conversely, distinct instance labels within a single image necessarily represent different objects. At this stage, we establish an unpaired set, S, comprising pairs of instance labels that must represent distinct instances.

By projecting pixels with instance labels onto the 3D space, each point within the reconstructed point cloud is assigned an instance label. Consequently, the point cloud is represented as a 7-dimensional vector, comprising the 3D position, RGB color, and instance label.

To consolidate different instance labels corresponding to the same object, we employ a k-NN-based algorithm. Initially, for each point, we compute the k_{merge} nearest points and their corresponding distances. Subsequently, instance labels are merged using the algorithm depicted in Figure 3. This algorithm verifies whether a point p_1 and its neighboring point p_2 possess the same object label and if their distance falls below the predefined threshold d_{merge}. Moreover, it examines the set S to determine if p_1 and p_2 should be considered as the same instance. It is important to note that the condition $(p_1, p_2) \notin S$ alone is insufficient; p_1 may share the same instance label with other points p_o, where $(p_o, p_2) \in S$, and likewise for $(p_1, p_{o'})$, where $p_{o'}$ represents a point within the group sharing the instance label with p_2. If all three conditions are satisfied, the instance labels are merged. The Union-Find data structure is employed in the algorithm to manage and merge instance labels.

3.3. Point Cloud Extraction

Following the reconstruction and panoptic segmentation steps, the point cloud P_t is partitioned into partial point clouds based on their instance labels. This division ensures that each resulting partial point cloud contains precisely one instance label, and no other partial point clouds share the same label. However, due to imperfections in the panoptic segmentation performed by Detectron2, erroneous instance labels may occasionally arise. These incorrectly labeled partial point clouds often consist of only a few points, as they do not correspond to any actual instances in the scene. To address this issue, we introduce a threshold n_{discard} and discard any partial point clouds containing fewer than n_{discard} points.

Let $P_t(i)$ denote the partial point cloud associated with the instance label i obtained from P_t. In contrast to the extraction of partial point clouds from P_t the extraction of corresponding partial point clouds from $P_{t'}$ is based on the bounding volume of $P_t(i)$. Specifically, if $P_t(i)$ resides within the range $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \times [z_{\min}, z_{\max}]$, then the corresponding point cloud $P_{t'}(i)$ consists of

Data: P: Point cloud, I: k_{merge} nearest neighbors, D: k_{merge} nearest neighbors' distances, S: Unpair set, d_{merge}: Distance threshold, UF: Union-Find instance equipped with *union* method that merges instance labels and *members* method that returns the group members of the given point.
Result: Instance labels of the same instance are merged into one single label.

```
for p1 ∈ P do
    i1 ← p1.instance_label;
    o1 ← p1.object_label;
    for p2 ∈ I[p1] do
        i2 ← p2.instance_label;
        o2 ← i2.object_label;
        if o1 ≠ o2 then continue;
        if D[p1][p2] > d_merge then continue;
        mergeable ← True;
        for m1 ∈ UF.members(i1) do
            if (m1, i2) ∈ S then
                mergeable ← False;
                break
            end
        end
        for m2 ∈ UF.members(i2) do
            if (i1, m2) ∈ S then
                mergeable ← False;
                break
            end
        end
        if mergeable then
            UF.union(i1, i2);
        end
    end
end
```

Figure 3: Algorithm to merge instance labels.

points from $P_{t'}$ that also fall within the same range $[x_{\min}, x_{\max}]\times[y_{\min}, y_{\max}]\times[z_{\min}, z_{\max}]$. It is important to note that $P_{t'}(i)$ may contain multiple instance labels, unlike $P_t(i)$ which only has a single instance label associated with it.

3.4. Change Detection

Our proposed algorithm focuses on detecting changes between two point clouds, $P_t(i)$ and $P_{t'}(i)$. Initially, the algorithm counts the number of points in $P_{t'}(i)$, denoted as m, and promptly identifies the change as *removed* if m is less than a predefined threshold n_{change}. This decision is based on the observation that instances are less likely to be represented by a small number of points, indicating the absence of the instance in $P_{t'}(i)$.

In cases where m is not less than n_{change}, the algorithm proceeds with the change classification process, as outlined in Figure 4. This process involves counting the

Data: P_1, P_2: Point clouds to compare, I: k_{change} nearest neighbors, D: k_{change} nearest neighbors' distances, r_{change}: Ratio threshold.
Result: Detection result: no change or removed.

```
n_matches ← 0;
for p1 ∈ P1 do
    o1 ← p1.object_label;
    for p2 ∈ I[pi] do
        o2 ← p2.object_label;
        if o1 = o2 then
            n_matches ← n_matches + 1
    end
end
if n_matches/(k ∗ P1.length) < r_change then
    return removed
else
    return no change
end
```

Figure 4: Change detection algorithm.

Table 1
Dataset description. Each frame has an RGB image, depth map, confidence map, and camera parameters.

Time	t	t'
Number of frames	1,558	720

number of matches in object labels between each point in $P_t(i)$ and its k_{change} nearest neighbors in $P_{t'}(i)$. The algorithm then classifies the change based on the ratio of successful matches to the total number of comparisons.

4. Experiments

4.1. Dataset

We utilized an iPhone equipped with ARKit to capture a dataset[1] comprising RGB-D images, confidence maps, and camera parameters within a room containing various objects, including chairs, computers, books, cell phones, and keyboards. The dataset consists of a collection of frames captured at two different time instances, t and t'. The specific number of frames captured at each time instance is presented in Table 1. It is important to highlight that individual partial point clouds within the dataset do not necessarily correspond to distinct object instances; some partial point clouds may represent different parts of the same object instance.

[1]Our source code is available on GitHub: https://github.com/Tomoya-Matsubara/RGB-D-Scan-with-ARKit

Table 2
Parameters setting. KEY – n_{sample}: number of pixels sampled per frame, k_{merge}: number of nearest neighbors in the label merge, d_{merge}: distance threshold in the label merge, n_{discard}: minimum number of points to form a partial point cloud, n_{change}: threshold for the number of points to predict immediately *removed*, k_{change}: number of nearest neighbors in the change detection, r_{change}: ratio threshold in the change detection.

Parameter	n_{sample}	k_{merge}	d_{merge}	n_{discard}	n_{change}	k_{change}	r_{change}
Value	5,000	30	0.02 [m]	1,000	5	30	0.1

Table 3
Object instances in the extracted point clouds.

Label	Count
cell phone	1
bottle	2
cup	2
keyboard	18
laptop	18
chair	21
book	26

4.2. Implementation Details

Table 2 provides an overview of the parameter settings employed in our implementation[2]. During the point cloud reconstruction phase, we applied random sampling and selected n_{sample} sample pixels from each frame to optimize processing time for subsequent operations.

Although Detectron2 offers support for various object instances, we focused on extracting point clouds associated with specific labels, namely *book, bottle, cup, chair, keyboard, laptop*, and *cell phone*.

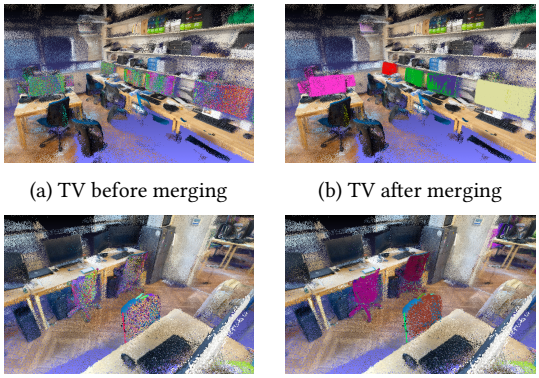
4.3. Annotation

We performed manual annotation to assign labels (i.e., *no change* and *removed*) to the extracted partial point clouds. During the annotation process, we carefully examined the origin of each partial point cloud in P_t, determining the corresponding object instance it belonged to, and verified the presence of the same object instance in $P_{t'}$. The presence of the object instance indicated *no change*, while its absence indicated that the object instance had been *removed*.

5. Results

The proposed algorithm extracted 88 partial point clouds from the dataset. Table 3 shows the detail of the extracted point clouds.

[2]Our implementation is available on GitHub: https://github.com/Tomoya-Matsubara/point-cloud-change-detection



(a) TV before merging (b) TV after merging

(c) Chair before merging (d) Chair after merging

Figure 5: Instance labels of *tv* and *chair* before and after merging. Each instance label has is colored uniquely. In the left images (a) and (c), each instance (*tv* and *chair*) has many colors, which shows many labels are assigned to the same instance before merging. In the right images (b) and (d), in contrast, each of them has only a few colors, which indicates those labels are merged correctly.

5.1. Label Merge

Figure 5 demonstrates the successful merging of instance labels belonging to the object categories *tv* and *chair*. It can be observed that not only the labels of planar *tv* instances but also those of *chair* instances with more complex shapes were effectively merged. This can be attributed to the fact that the merge operation solely relied on distance information, without making any assumptions about the shapes of the instances.

Although some instances still retained multiple labels, the overall number of instance labels was significantly reduced by approximately 40% (from 4,534 to 2,729). After discarding partial point clouds with a point count below the threshold n_{discard}, the remaining labels were further reduced to a final count of 88.

5.2. Change Detection

The result of the change detection is shown in Figure 6. As the false negative (bottom left corner of the matrix) indicates, the proposed algorithm detected changes perfectly.

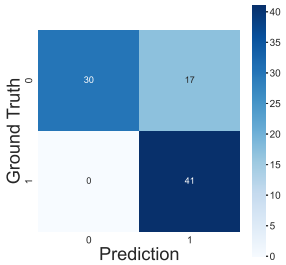
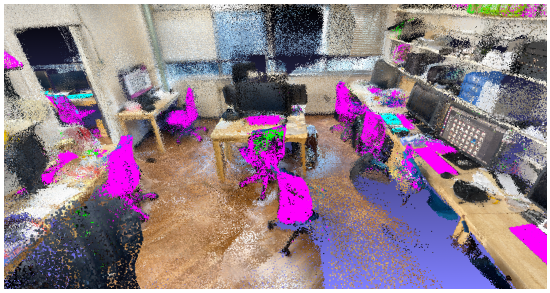


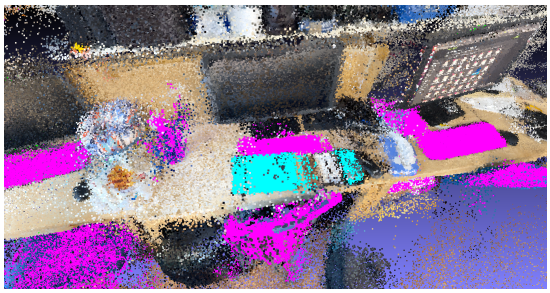
Figure 6: Confusion matrix in the change detection. Label 0 and 1 represent *no change* and *removed*, respectively.



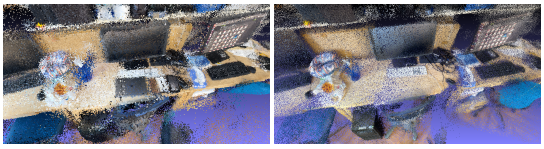
(a) Change detection result

(b) P_t: Point cloud at t (c) $P_{t'}$: Point cloud at t'

Figure 7: Visualization of the change detection result (a) and the original point clouds P_t (b) and $P_{t'}$ (c), all captured from the same angle. COLOR – magenta: true negative, green: false positive, red: false negative, cyan: true positive.



(a) Change detection result of densely close objects



(b) P_t: Point cloud at t (c) $P_{t'}$: Point cloud at t'

Figure 8: Visualization of the change detection result (a) and the original point clouds P_t (b) and $P_{t'}$ (c), all captured from the same angle, different from Figure 7. COLOR – magenta: true negative, cyan: true positive.

However, there were 17 cases of false positives, where the algorithm incorrectly predicted that an object instance was removed when it was actually present. This can be attributed to the limited number of partial point clouds in $P_{t'}$. Figure 7 provides a visual representation of the change detection results, with P_t and $P_{t'}$ captured from the same angle. False positive cases are highlighted in green, particularly noticeable in the central chair in Figure 7 (a). Upon closer examination of the same chair in Figure 7 (b) and (c), it becomes evident that P_t contains a substantial number of points accurately representing the chair's shape. Conversely, $P_{t'}$ only consists of a few points, failing to capture the chair adequately. Consequently, due to the algorithm's tendency to predict removal in cases with limited point coverage, these false positives were triggered.

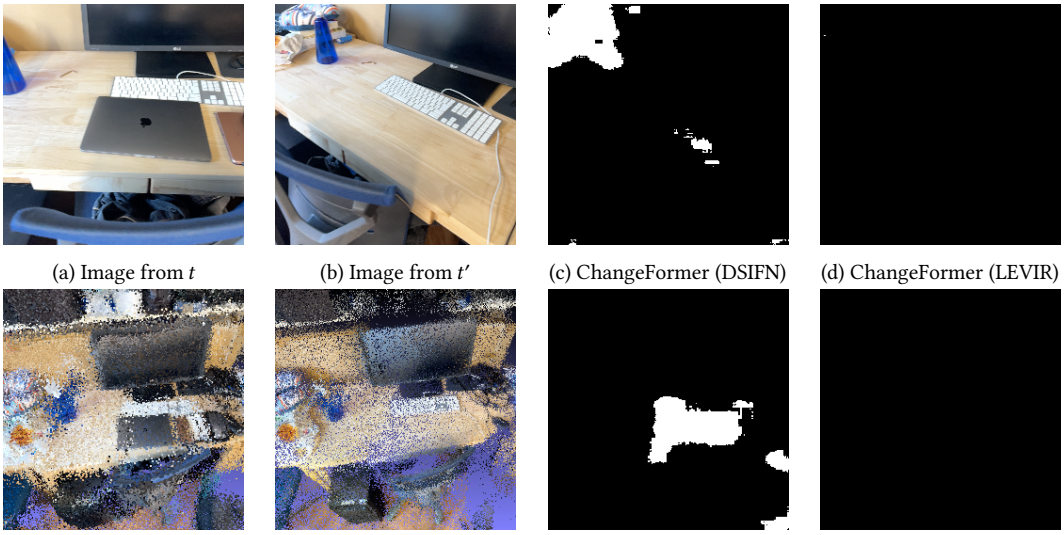
Since there is no false negative, as explained above, no red-colored point can be seen in Figure 7 (a).

Figure 8 provides a visualization from a different angle, showcasing objects on a table, such as a laptop and a smartphone. These objects are typically ignored as change detection targets in previous works due to their close proximity to each other, often just a few centimeters apart. However, in the proposed algorithm, we successfully detected the removal of such objects by leveraging pixel-level object segmentation rather than relying solely on distance-based criteria. This approach allowed us to accurately identify and classify the removal of objects, even in challenging scenarios where objects are spatially close to each other.

5.3. Comparison with 2D Change Detection

Figure 9 presents the change detection results obtained using the pre-trained ChangeFormer [11] model. This figure showcases the same scene as shown in Figure 8, with a focus on the successful detection of the laptop removal. Notably, Figure 9 (a) and (b) exhibit misalignment because they were recorded by humans, as opposed to robots whose movements can be pre-defined and controlled.

In Figure 9 (c), the ChangeFormer model trained on DSIFN-CD detects changes in the top left corner, although no actual changes occurred in that region. Additionally, while the pixels at the center seemingly detect the re-



(a) Image from t (b) Image from t' (c) ChangeFormer (DSIFN) (d) ChangeFormer (LEVIR)

(e) Point Cloud captured at t (f) Point Cloud captured at t' (g) ChangeFormer (DSIFN) (h) ChangeFormer (LEVIR)

Figure 9: 2D change detection by ChangeFormer [11]. (a) and (b) are recorded images at t and t' around the same scene, and (c) and (d) are the detection result. (e) and (f) are point clouds of the same scene at t and t' captured from the same angle, and (g) and (h) are the detection result. In both cases, ChangeFormer trained on LEVIR-CD does not detect any changes.

moval of the laptop, their size is significantly smaller compared to the actual change. On the other hand, Figure 9 (e) and (f) depict manually captured images achieved by aligning two point clouds. In this particular case, Figure 9 (g) successfully detects the removal.

It is worth noting that these results are not surprising, considering that ChangeFormer was not specifically designed or trained to detect changes in unaligned images. However, in metaverse applications, it is expected that both robots and humans contribute to data collection (e.g., image capture) for immediate updates to the virtual world. Consequently, captured images are not always perfectly aligned, and cases resembling Figure 9 (e) and (f) are less likely to arise, especially when the detection target instance is not pre-determined. From this perspective, our proposed algorithm demonstrates its ability to detect changes in object instances, even when captured from different angles or under misalignment conditions.

5.4. Comparison with Related Work of 3D Change Detection

For reference, we conducted a comparison of our change detection results with the performance of PoChaDeHH, HGI-CD, and SiamGCN [16] algorithms on their street scene dataset, as presented in Table 4. It should be noted that direct comparison between our algorithm and the reference algorithms is challenging due to the following reasons:

Table 4
Comparison of class-wise recall with T. Ku et al.'s work [16]. In binary classification, class-wise recall for label 0 is the usual recall, and that for label 1 is specificity. The scores of the T. Ku et al.'s work are cited from [16].

Method	no change	removed	added
Ours	0.64	**1.00**	-
PoChaDeHH	0.77	0.48	0.67
HGI-CD	**0.81**	0.28	0.20
SiamGCN	0.66	0.86	**0.94**

- Classification Differences: The reference algorithms are designed for five-class classification, including categories such as *no change*, *removed*, *added*, *change*, and *color change*. In contrast, our algorithm focuses on detecting the removal of object instances.
- Dataset Variation: The reference algorithms utilize a different dataset consisting of street scenes, which may introduce variations in terms of scene composition, object types, and background elements.
- Known Object Positions: The positions of the objects of interest are provided in the reference algorithms, whereas our algorithm operates without this prior knowledge.

Despite these differences, our algorithm demonstrates superior performance in terms of recall for the *removed*

class compared to the reference algorithms. Even when considering the *added* class as equivalent to the *removed* class, our algorithm still exhibits a slight performance advantage of 0.06. However, the recall for the *no change* class is comparatively lower than that of PoChaDeHH and HGI-CD; according to [16], these algorithms tend to predict *no change*, but this specialization comes at the expense of generalization performance for other classes. Conversely, SiamGCN, which showcases the best generalization performance among the reference algorithms, exhibits a recall rate similar to ours.

6. Conclusion

In this study, we have presented a change detection algorithm that relies on panoptic segmentation and k-NN, operating without the need for positional information about the object of interest.

Our label merge algorithm effectively combines different instance labels that may correspond to the same object instance, resulting in a reduced number of labels. We have demonstrated its success in merging labels for instances with complex shapes, such as chairs.

Through experiments conducted on an indoor point cloud dataset, our change detection algorithm has proven its ability to detect the removal of closely situated objects. Unlike 2D change detection techniques, our algorithm surpasses the limitations of capturing changes from a single angle and showcases its capability to detect changes in objects captured from different angles. Furthermore, our algorithm has been compared with a state-of-the-art algorithm, revealing its competitive performance in terms of recall, particularly for the *removed* class.

In future research, we propose exploring techniques to assess the quality of input images. Blurred images caused by camera shake can adversely impact the segmentation performance, and addressing this issue would enhance the overall accuracy of our algorithm. Additionally, as multiple frames may capture the same scene with minimal differences, removing duplicates could be considered to reduce the number of frames for processing, ultimately improving computational efficiency.

References

[1] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, M. Bennamoun, Deep learning for 3d point clouds: A survey, IEEE Transactions on Pattern Analysis and Machine Intelligence 43 (2021) 4338–4364. doi:10.1109/TPAMI.2020.3005434.

[2] C. R. Qi, H. Su, K. Mo, L. J. Guibas, Pointnet: Deep learning on point sets for 3d classification and segmentation, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 652–660.

[3] C. R. Qi, L. Yi, H. Su, L. J. Guibas, Pointnet++: Deep hierarchical feature learning on point sets in a metric space, Advances in neural information processing systems 30 (2017).

[4] X. Chen, H. Ma, J. Wan, B. Li, T. Xia, Multi-view 3d object detection network for autonomous driving, in: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, 2017, pp. 1907–1915.

[5] T. Yin, X. Zhou, P. Krahenbuhl, Center-based 3d object detection and tracking, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2021, pp. 11784–11793.

[6] Z. Yang, Y. Sun, S. Liu, X. Shen, J. Jia, Ipod: Intensive point-based object detector for point cloud, arXiv preprint arXiv:1812.05276 (2018).

[7] M. Voelsen, J. Schachtschneider, C. Brenner, Classification and change detection in mobile mapping lidar point clouds, PFG – Journal of Photogrammetry, Remote Sensing and Geoinformation Science 89 (2021) 195 – 207.

[8] F. J. Lawin, M. Danelljan, P. Tosteberg, G. Bhat, F. S. Khan, M. Felsberg, Deep projective 3d semantic segmentation, in: Computer Analysis of Images and Patterns: 17th International Conference, CAIP 2017, Ystad, Sweden, August 22-24, 2017, Proceedings, Part I 17, Springer, 2017, pp. 95–107.

[9] H. Duan, J. Li, S. Fan, Z. Lin, X. Wu, W. Cai, Metaverse for social good: A university campus prototype, in: Proceedings of the 29th ACM international conference on multimedia, 2021, pp. 153–161.

[10] A. El Saddik, Digital twins: The convergence of multimedia technologies, IEEE multimedia 25 (2018) 87–92.

[11] W. G. C. Bandara, V. M. Patel, A transformer-based siamese network for change detection, in: IGARSS 2022-2022 IEEE International Geoscience and Remote Sensing Symposium, IEEE, 2022, pp. 207–210.

[12] J. Chen, Z. Yuan, J. Peng, L. Chen, H. Huang, J. Zhu, Y. Liu, H. Li, Dasnet: Dual attentive fully convolutional siamese networks for change detection in high-resolution satellite images, IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 14 (2020) 1194–1206.

[13] K. Sakurada, T. Okatani, K. Deguchi, Detecting changes in 3d structure of a scene from multi-view images captured by a vehicle-mounted camera, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2013, pp. 137–144.

[14] H. Andreasson, M. Magnusson, A. Lilienthal, Has somethong changed here? autonomous difference detection for security patrol robots, in: 2007 IEEE/RSJ International Conference on Intelligent

Robots and Systems, 2007, pp. 3429–3435. doi:10.1109/IROS.2007.4399381.

[15] U. Katsura, K. Matsumoto, A. Kawamura, T. Ishigami, T. Okada, R. Kurazume, Spatial change detection using voxel classification by normal distributions transform, in: 2019 International Conference on Robotics and Automation (ICRA), 2019, pp. 2953–2959. doi:10.1109/ICRA.2019.8794173.

[16] T. Ku, S. Galanakis, B. Boom, R. C. Veltkamp, D. Bangera, S. Gangisetty, N. Stagakis, G. Arvanitis, K. Moustakas, Shrec 2021: 3d point cloud change detection for street scenes, Computers Graphics 99 (2021) 192–200. URL: https://www.sciencedirect.com/science/article/pii/S0097849321001369. doi:https://doi.org/10.1016/j.cag.2021.07.004.

[17] S. Nikoohemat, M. Koeva, S. Oude Elberink, C. Lemmen, Change detection from point clouds to support indoor 3d cadastre, The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences 42 (2018) 451–457.

[18] K. Sakurada, M. Shibuya, W. Wang, Weakly supervised silhouette-based semantic scene change detection, in: 2020 IEEE International conference on robotics and automation (ICRA), IEEE, 2020, pp. 6861–6867.

[19] R. B. Rusu, N. Blodow, M. Beetz, Fast point feature histograms (fpfh) for 3d registration, in: 2009 IEEE international conference on robotics and automation, IEEE, 2009, pp. 3212–3217.

[20] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[21] H. Chen, Z. Shi, A spatial-temporal attention-based method and a new dataset for remote sensing image change detection, Remote Sensing 12 (2020) 1662.

[22] C. Zhang, P. Yue, D. Tapete, L. Jiang, B. Shangguan, L. Huang, G. Liu, A deeply supervised image fusion network for change detection in high resolution bi-temporal remote sensing images, ISPRS Journal of Photogrammetry and Remote Sensing 166 (2020) 183–200.

[23] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, R. Girshick, Detectron2, https://github.com/facebookresearch/detectron2, 2019.