Removing Moving Objects from Point Cloud Scenes

Krystof Litomisky
klitomis@cs.ucr.edu

Abstract. Three-dimensional simultaneous localization and mapping is a topic of significant interest in the research community, particularly so since the introduction of cheap consumer RGB-D sensors such as the Microsoft Kinect. Current algorithms are able to create rich, visually appealing maps of indoor environments using such sensors. However, state-of-the-art systems are designed for use in static environments. This restriction means, for instance, that there can be no people moving around the environment while the mapping is being done. This severely limits the application space for such systems. To address this issue, we present an algorithm to explicitly identify and remove moving objects from multiple views of a scene. We do this by finding corresponding objects in two views of a scene. If the position of an object with respect to the other objects changes between the two views, we conclude that the object is moving and should therefore be removed. After the algorithm is run, the two views can be merged using any existing registration algorithm. We present results on scenes collected around a university building.

1 Introduction

Although point clouds and sensors that provide point cloud data have been around for several decades, the introduction in 2010 of the Microsoft Kinect RGB-D (RGB color + per-pixel depth) sensor reinvigorated the field dramatically. One particularly popular area of research has been using RGB-D sensors for Simultaneous Localization and Mapping (SLAM) in primarily indoor environments. Current state of the art systems have achieved impressively accurate results, producing visually-appealing maps of moderately-sized indoor environments [1]. Such algorithms typically rely on the Iterative Closest Point (ICP) for point cloud registration, and also incorporate loop-closing techniques to detect when an agent has returned to a previously visited area [2].

In existing mapping applications, the maps are created in static environments: there are no moving objects, and, in particular, no people walking around. In more dynamic environments, not handling moving objects appropriately can lead to maps that contain moving objects as permanent features, inconsistent maps, or even registration failure. Our work addresses this issue by introducing a novel preprocessing step to explicitly identify and remove moving objects from point cloud frames prior to registration. A sample result of our algorithm is in Fig. 1.

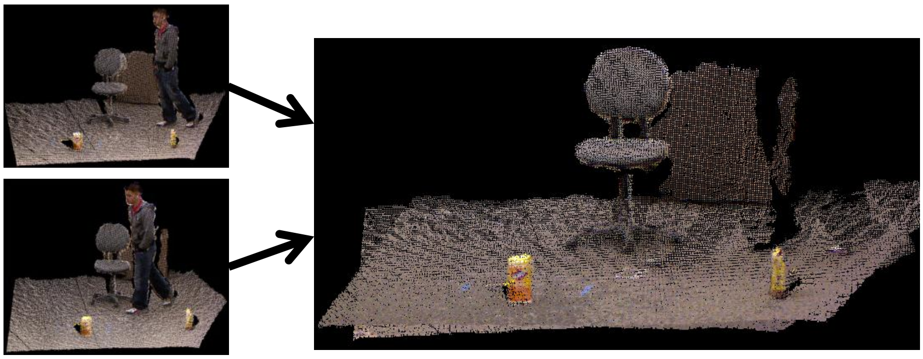

Fig. 1. A sample result of our algorithm. The top row shows two input point clouds. Our algorithm identified the person as a moving object and removed him from each point cloud. We then aligned and merged the clouds to produce the cloud in the bottom row.

To identify moving objects, we compare two frames with significant overlap. The viewpoint between the two frames can change, and some amount of time must elapse before the second frame is captured. We require this so that the position of moving objects changes between the frames.

An overview of our approach is in Fig. 2. Having captured two frames, we segment out individual clusters, and find which cluster from the first frame corresponds to each cluster from the second. We then analyze the spatial relationship of each cluster in a frame to the other cluster in the frame. If this relationship changes from one frame to the next, we conclude that the cluster in question must be moving a moving object and remove it. Having done this, we can apply any existing registration algorithm to register, align, and merge the two clouds.

2 Related Work

Three dimensional Simultaneous Localization and Mapping has been an area of immense research interest. Over the years, a number of approaches using different technologies have been developed, including range scans [3, 4], stereo cameras [5], monocular cameras [6], and recently also consumer RGB-D cameras [1]. However, explicitly identifying and removing moving objects from point cloud data has not been a topic of great interest in the research community. A number of works deals with the issue implicitly.

In their RGB-D mapping work, Henry et al. [1] do not address moving objects specifically, and do not present any results from environments with moving objects. However, the use of a surfel representation [7] allows them to deal with some instances of moving objects. Because surfels have an area, they allow reasoning about occlusion. This ultimately allows this representation to remove moving objects by eliminating surfels which occlude other surfels. However, this approach will fail to eliminate moving objects when these objects do not occlude other known objects. Furthermore, converting all data to surfels is computationally inefficient.

Similarly, the Kinect Fusion work [8], which implements a real-time version of ICP that runs on CUDA-enabled GPUs, deals with moving objects implicitly by using a volumetric representation. Kinect Fusion is not primarily a SLAM algorithm; as a result, the approach does not scale well. In particular, with current GPUs, the approach cannot be used for environments larger than a moderately small room (approximately a 5m x 5m x 5m volume).

Other volumetric approaches, such as the Octree-based OctoMap [4], also identify and eliminate moving objects only implicitly. The OctoMap's probabilistic nature means that it may require a substantial amount of different measurements until a particular measurement of a moving object is discarded.

Unlike previous related work, which deals with moving objects in only an implicit and incomplete way, this paper presents a focused and systematic approach for removing moving objects from point cloud scenes captured from a potentially moving platform. An additional advantage of our approach is that it deals directly with the point cloud data. This means that after eliminating moving objects through our approach, any existing algorithm for point cloud registration – or any other desired application – can be applied straightforwardly.

3 Technical Approach

A high-level overview of our approach is given in Fig. 2. For notational clarity, we will refer to the first point cloud we process as the "source" cloud, and to the second cloud we process as the "target" cloud. Our approach is symmetric, however, and which cloud is designated as "source" does not affect our results. For some of the point cloud processing and manipulation tasks, we rely on the Point Cloud Library (PCL) [9].

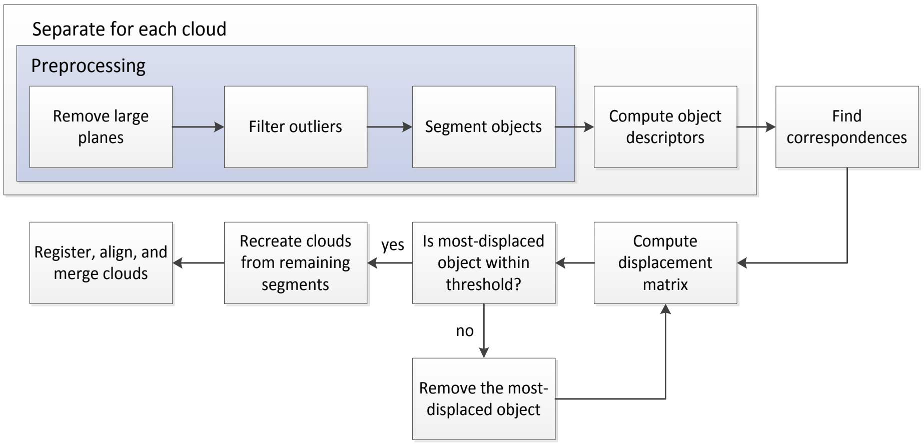
Separate for each cloud
Preprocessing
Remove large planes
Filter outliers
Segment objects
Compute object descriptors
Find correspondences
Compute displacement matrix
Is most-displaced object within threshold?
yes
no
Remove the most-displaced object
Recreate clouds from remaining segments
Register, align, and merge clouds

Fig. 2. An overview of our system.

3.1 Preprocessing

Before identifying moving clusters, we temporarily remove large planes, filter outlier points, and run segmentation to identify individual clusters.

The reason for removing large planes is twofold: first, we can safely assume that large planes are not parts of moving objects, so we do not need to consider them in subsequent steps of the algorithm. Second, removing large planes – in particular, the floor – improves the performance of our segmentation algorithm.

To identify planes, we use a Random Sample Consensus (RANSAC) algorithm. While this algorithm requires certain parameters, for the purposes of our task the results are not sensitive to specific values. Having removed any large planes, we filter out outlier points. This effectively removes points that are due to sensor noise.

We now use a Euclidean Cluster Extraction algorithm to get individual clusters. Due to the quantization of the depth data, which is inevitable with consumer RGB-D sensors, we need to use a somewhat large cluster tolerance value. We settled on 15 cm. In other words, if two points are less than 15 cm apart, they will be part of the same cluster. This value reliably puts points from the same object into the same cluster, while also correctly putting separate objects in separate clusters in the majority of cases.

Each point cloud can now be represented by a set of clusters. Let $C_s = \{s_1, s_2, \dots, s_{n_s}\}$ be the source point cloud, and $C_t = \{t_1, t_2, \dots, t_{n_t}\}$ be the target point cloud. Here, s_i is the set of points representing the i^{th} cluster in the source cloud, and t_i is the set of points representing the i^{th} cluster of the target cloud.

3.2 Computing Object Descriptors

We use the Viewpoint Feature Histogram (VFH) proposed by Rusu et al. [10] as our object descriptor. The VFH has been shown to have good object classification performance; it outperforms spin images for these tasks [10]. The VFH stores the relationships between the pan, tilt, and yaw angles between pairs of normals in a cluster, as well as the relationships between the viewpoint direction and the cluster surface normals. The result is a histogram descriptor with 308 bins.

3.3 Finding Cluster Correspondences

We now need to find cluster correspondences, or which cluster from the target cloud corresponds to each cluster of the source cloud. We calculate the distance between a pair of clusters in feature space in two different ways, and compare the final performance in Section 4.

The first distance measure is the sum of absolute differences between two clusters' VFH descriptors, while the second distance measure comes from an insight on the ordering of the VFH: the bins for each angle "category" are consecutive. For example, if the yaw angle between two points changed slightly, the pair would move from one bin into a bin immediately next to it. As a result, small object deformations (such as a person's pose changing as he walks) as well as changes in sensor position would cause nonlinear local deformations to the object's histogram. Fig. 3 illustrates the local deformations of a part of the VFH due to sensor motion.

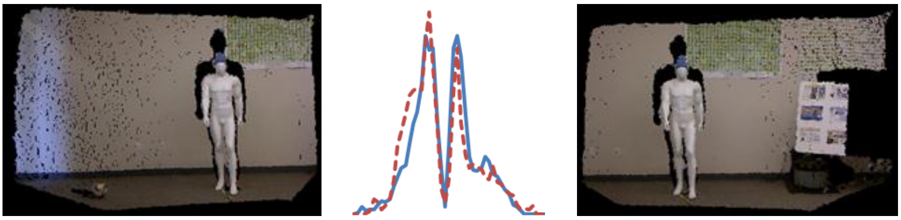

Fig. 3. Local deformations of the yaw component of the Viewpoint Feature Histogram of the mannequin extracted from the two views above. Histogram for the mannequin on the left is solid blue; right is dashed red. Dynamic Time Warping addresses such local deformations.

Such issues have been addressed in the literature with Dynamic Time Warping [11]. In essence, dynamic time warping finds a nonlinear, monotonic mapping from one sequence to the other, allowing for shifts and deformations between the two series. We use dynamic time warping with a warping constraint given by the Sakoe-Chiba Band [12], which restricts the amount of warping to the amount created by camera or object movement in highly dynamic scenes.

Using either distance measure, we compute an $n_s \times n_t$ feature distance matrix $F = \left[f_{i,j}\right]$ where

$$f_{i,j} = dist_f\left(s_i, t_j\right) \quad (1)$$

F thus holds the distance from each cluster in the source cloud to each cluster in the target cloud. To specify the cluster correspondences, we iteratively select a pair of clusters (s_i, t_j) such that $f_{i,j}$ is the minimum of all s_i and t_j that have not previously been selected as corresponding to any cluster.

Due to either sensor motion and/or object motion, n_s need not equal n_t, and there may therefore be leftover objects for which we found no correspondence. We remove all such clusters from their cloud. This may remove objects that could provide useful data, but this is an acceptable tradeoff to ensure the removal of any moving objects that appear only in one frame.

Having identified cluster correspondences and removed any clusters for which no correspondences were found, we have $n = \min(n_s, n_t)$ clusters left for each cloud. Before proceeding further, we reorder the clusters in C_t such that cluster s_i corresponds to cluster $t_i \; \forall \; i$.

3.4 Identifying and Removing Moving Objects

We are now ready to identify and remove moving objects that appear in both point clouds. We calculate the Euclidean distance in world-coordinate space between each pair of clusters for each cloud. This gives rise to two $n \times n$ matrices, D_S for the source cloud and D_T for the target cloud. For the source cloud,

$$D_S = \begin{bmatrix} 0 & d^s_{1,2} & \dots & d^s_{1,n} \\ d^s_{2,1} & 0 & & \\ \vdots & & \ddots & \vdots \\ d^s_{n,1} & & \dots & 0 \end{bmatrix} \tag{2}$$

where $d^s_{i,j}$ is the world-coordinate-space Euclidean distance between cluster i and cluster j in the source point cloud. It is straightforward to see that the diagonal elements of D_S are equal to 0, and that the matrix is symmetric. D_T is defined in the same way for the target cloud.

We now need a measure of how much the position of each cluster has changed from the source cloud to the target cloud. To do this, we calculate the displacement vector $\Delta = [\delta_1, \delta_2, \dots, \delta_n]^T$, where δ_i is the displacement of cluster i,

$$\delta_i = \sum_{k=1}^{n} \left| d^s_{i,k} - d^t_{i,k} \right| \tag{3}$$

In essence, δ_i is the sum of how much the distance of cluster i to each other cluster has changed from one cloud to the other cloud.

We now iteratively remove the cluster which has the greatest displacement value as long as this value is above a threshold ε. After removing a cluster, we recalculate Δ before checking the displacement values against the threshold. In order to find the optimal value of ε, we generated a ROC curve (Fig. 4). For the ROC curve, a true positive is a moving object that was removed from the scene, and a false positive is a static object that was removed from the scene. See Section 4 for details regarding the data used.

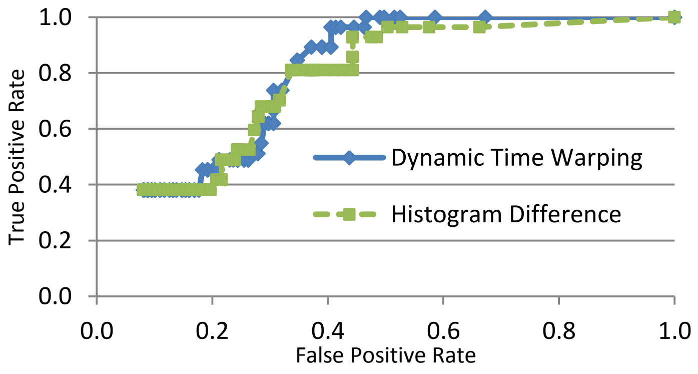

Fig. 4. Object removal threshold ROC curve. A true positive is a removed moving object, and a false positive is a removed static object.

We ran our algorithm for different thresholds for both the histogram difference and dynamic time warping distance measures. We got better results with dynamic time warping. In particular, we were able to correctly remove all moving objects with $\varepsilon = 0.7$ meters when using dynamic time warping. We therefore use this value in our experiments (Section 4).

If any incorrect correspondences are made in the previous step, all clusters that were identified incorrectly will have large displacement values in Δ. As a result, these clusters will be removed during this step of the algorithm. While this can remove useful static objects, it will not cause any additional moving objects to be included in the recreated cloud.

3.5 Recreating and Merging the Clouds

Having removed all of the moving objects from C_s and C_t, we reconstruct each cloud from its remaining clusters as well as the planes that had been removed from it. After this, the clouds are in a standard point cloud format, and any existing registration, alignment, and merging algorithms can be used to concatenate the clouds. The result of this operation is a point cloud model of the environment with no moving objects in the point cloud, even though such objects might have been present in one or both of the original clouds.

4 Experiments

We tested our algorithm on 14 scenes collected around a university building. Each scene consists of two views, and each view has up to 2 moving people. Fig. 5 shows some results, demonstrating several different scenarios our algorithm can deal with.

Fig. 5 (a) shows a scene with multiple moving people, as well as significant camera motion between the two point cloud frames. As a result of this, we cannot find good correspondences for all objects in the scene. For example, we remove the table and wall to the right of the scene. We are able to correctly match the chair to the right of the scene, even though only a small part of it is visible in the top frame.

Note that there is a third person in the scene, sitting at a desk in the background. This person is put into the same cluster as the desk he is sitting at by our algorithm, and since he is not moving he is also included in the recreated cloud.

Fig. 5(b) shows our results on a scenario where a person who is not present in one point cloud moves into view by the time the second point cloud is captured. Our algorithm correctly identifies that there is no equivalent object for the person in the other frame, and removes the person from the point cloud.

Fig. 5(c) shows a scene where a walking person completely changes direction from one frame to the next. Nevertheless, our algorithm matches the person in one frame to the person in the next frame and correctly removes him from both clouds.

4.1 Quantitative Analysis

Fig. 4 shows the ROC curve obtained by running our algorithm on the dataset, where a true positive is defined as a moving object that is removed from the scene and a false positive is defined as a static object that was removed from the scene.

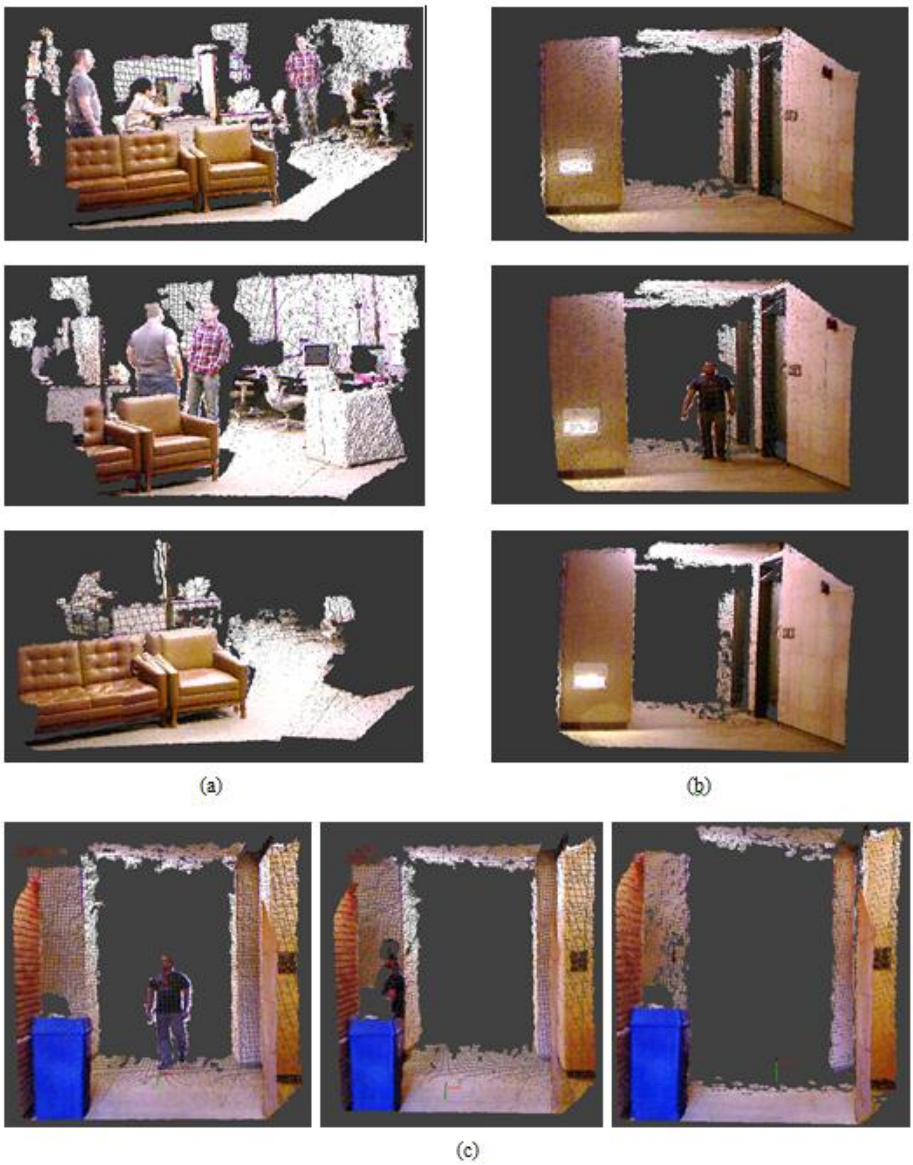

Fig. 5. Example results. Part (a) shows an office scene with three people (two walking/talking, one sitting at a computer). The top two frames from column (a) were merged together to produce the bottom point cloud. Part (b) shows a corridor scene, where the person just entered the corridor through one of the doors on the right. Part (c) shows a person first approaching the camera, and then proceeding to the left, having changed directions.

We get better results when using the dynamic time warping distance measure than with the simple histogram difference. In particular, for the object removal threshold $\varepsilon = 0.7$, we correctly remove all moving objects while also removing 47% of the static objects.

We also evaluate what fraction of the total points that belong to stationary objects we keep in the recreated clouds. For each scene, we calculate this number separately for each of the two frames, and then report their mean and standard deviation. Fig. 6 shows the results. On average, we keep 85% of the static points in a scene.

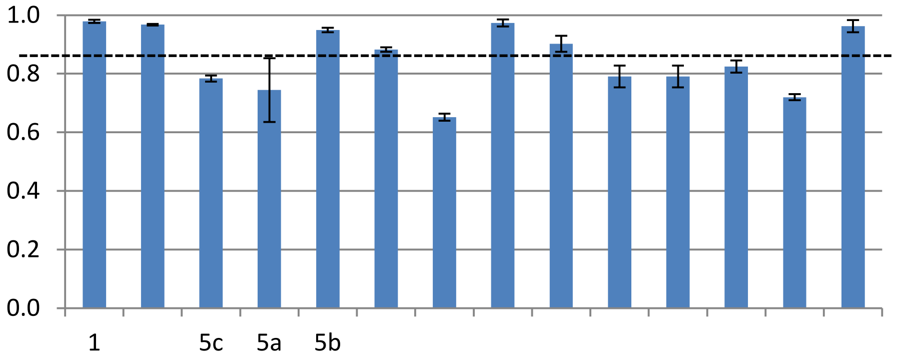

Fig. 6. Fraction of static points kept for each of the 14 scenes on which we evaluated our algorithm. Numbers along the x-axis indicate which figure corresponds to the particular scene, if applicable. The mean is 85% of static points retained.

Even if we correctly identify and remove only those points belonging to a moving object, the fraction retained will be less than 1. This is because we filter outliers while processing the frames. It is also inevitable that some static objects be removed due to occlusion or camera movement. For example, in Fig. 5(a), the desk to the right of the scene in the view in the middle image does not appear at all in the view in the top image. For this reason, it is reasonable that an algorithm would remove it. In a SLAM scenario, we would get multiple views of the scene with the desk in it, and would ultimately keep it.

5 Conclusions and future work

We introduce early work on an algorithm for identifying moving objects in point cloud scenes. We can eliminate moving objects from scenes while retaining most of the static objects. Thus, when used as a preprocessing step, our approach can complement existing point cloud-based SLAM algorithms. This will allow existing SLAM approaches to create consistent maps even in the presence of moving objects, making the system applicable in more scenarios.

In future work, we will consider different features for object correspondence identification, as well as different conditions for removing planes as well as moving objects. The running time of the algorithm is dominated by segmentation; ways to reduce this time should be considered in future work.

References

[1] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, “RGB-D Mapping: Using depth cameras for dense 3D modeling of indoor environments,” in *the 12th International Symposium on Experimental Robotics (ISER)*, 2010.

[2] P. Newman, “SLAM-Loop Closing with Visually Salient Features,” in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 635-642.

[3] S. May et al., “Three-dimensional mapping with time-of-flight cameras,” *Journal of Field Robotics*, vol. 26, no. 11–12, pp. 934-965, Nov. 2009.

[4] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard, “OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems,” in *Proc. of the ICRA 2010 Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation*, 2010.

[5] T. Lemaire, C. Berger, I.-K. Jung, and S. Lacroix, “Vision-Based SLAM: Stereo and Monocular Approaches,” *International Journal of Computer Vision*, vol. 74, no. 3, pp. 343-364, Feb. 2007.

[6] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, “MonoSLAM: real-time single camera SLAM.,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1052-67, Jun. 2007.

[7] H. Pfister, M. Zwicker, J. Van Baar, and M. Gross, “Surfels: Surface elements as rendering primitives,” in *Proceedings of the 27th annual conference on Computer Graphics and Interactive Techniques*, 2000, pp. 335–342.

[8] R. Newcombe et al., “KinectFusion: Real-time dense surface mapping and tracking,” in *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, 2011, pp. 127-136.

[9] R. B. Rusu and S. Cousins, “3D is here: Point Cloud Library (PCL),” in *Proceedings of the 2011 IEEE International Conference on Robotics and Automation, Shanghai, China*, 2011.

[10] R. B. Rusu, G. Bradski, R. Thibaux, and J. Hsu, “Fast 3D Recognition and Pose Using the Viewpoint Feature Histogram,” in *International Conference on Intelligent Robots and Systems*, 2010, pp. 2155-2162.

[11] D. Berndt and J. Clifford, “Using dynamic time warping to find patterns in time series,” in *New York*, 1994, vol. 398, pp. 359-370.

[12] H. Sakoe and S. Chiba, “Dynamic programming algorithm optimization for spoken word recognition,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 26, no. 1, pp. 43-49, Feb. 1978.