Proceeding of the 2015 IEEE
International Conference on Information and Automation
Lijing, China, August 2015

Removing Dynamic 3D Objects from Point Clouds of a Moving RGB-D Camera

Canben Yin[†], Shaowu Yang[†], Xiaodong Yi[†*], Zhiyuan Wang[†], Yanzhen Wang[†], Bo Zhang[†], Yuhua Tang[†]
[†]*State Key Laboratory of High Performance Computing, School of Computer*
National University of Defense Technology
Changsha, P. R. China, 410073
corraddr: yixiaodong@nudt.edu.cn

***Abstract*—Most state-of-the-art visual simultaneous localization and mapping (SLAM) systems are designed for applications in static environments. However, during a SLAM process, dynamic objects in the field-of-view of the camera will affect the accuracy of visual odometry and loop-closure detection. In this paper, we present a solution to removing dynamic objects from RGB images and their corresponding depth images when a RGB-D camera is mounted on a mobile robot for visual SLAM. We transform two selected successive images to the same image coordinate frame through feature matching. Then we detect candidate image pixels of dynamic objects by applying a threshold to the image difference between the two images. Furthermore, we utilize depth information of the candidate pixels to decide whether true dynamic objects are found. Finally, in order to extract a complete 3-dimensional (3D) dynamic object, we find the correspondence between the object and a cluster of the point cloud computed from RGB-D images. To evaluate the performance of detecting and removing dynamic objects, we do experiments in various indoor scenarios, which demonstrate the efficiency of the proposed algorithm.**

***Keywords*-Removing dynamic 3D objects, Image difference, RGB-D camera, Point cloud, Mobile robot, Visual SLAM**

I. Introduction

How to achieve accurate localization and environment mapping is considered to be an essential problem for autonomous navigation of many mobile robots. SLAM is proposed to focus on solving this problem. The majority of robotics researchers fuse multiple sensors to improve the accuracy of the localization and mapping. However, the proposed systems can be too sophisticated to be efficiently applied. Others pay more attention to designing efficient algorithms for a specific kind of sensors, like cameras. Cameras are widely used for various applications of mobile robots because of their advantages over other sensors, e.g. being low cost and light weight, having large detection range and rich features, etc. Visual-SLAM has also been a popular topic in the robotics community, especially since Kinect-like RGB-D cameras were brought to public.

Visual SLAM systems have achieved accurate results, producing 2D or 3D maps and providing accurate localization in static indoor environments. In visual SLAM systems, pose tracking is maintained by detecting and tracking visual features in sequential images. Since real world is inherently unpredictable, environments can be highly dynamic and unpredictable for mobile robots [23]. In such scenarios, the pose tracking of visual SLAM could be affected by visual features of dynamic objects. Loop-closure detection is also crucial for visual SLAM. When a robot has returned to a previously visited area after having exploring new terrains for a certain period of time, loop-closure detection can be used to efficiently correct the pose drift of the pose estimates. However, if there are textured dynamic objects in environments, robots are more likely to miss loop closures or to detect false-positive loops.

The above mentioned problems in visual SLAM can be solved by removing dynamic objects from the images before they are used for SLAM. Thus, the accuracy of pose tracking and loop-closure detection can be improved. Moreover, a static 3D map without dynamic objects can be achieved. In this paper, we address how to remove dynamic objects from point clouds and RGB-D images when a Kinect sensor is also moving. In order to find candidate dynamic objects in color images, we retrieve image differences between two selected successive images and find out the candidate regions of dynamic objects, with the assistance from the corresponding depth images. Furthermore, we extensively utilize the information from the point clouds produced from the RGB-D images to finally decide the image regions corresponding to the dynamic objects, which will be removed from both the RGB-D images and the point cloud.

In the rest of the paper, related work on visual SLAM and dynamic-object detection are reviewed in sect. II. Technical approach is explained in sect. III, followed by experiments and results in sect. IV. Finally, conclusions and future work are presented in sect. V.

II. Related Work

In robotics, mapping and localization are often coupled together. Therefore, simultaneous localization and mapping (SLAM) was proposed for solving it [20], [21]. Over three decades, researchers have developed a large number of SLAM systems using different kindes of sensors, likely range scans[3], monocular cameras[5], [6], stereo cameras[15], and RGB-D cameras. In RGB-D SLAM work [1], [8], [9], researchers have developed efficient matching algorithms, pose estimation approaches, loop-closure algorithms. In their work, the effects of dynamic objects to SLAM are often ignored, resulting these objects to be included in the maps of SLAM systems. Recently, real-time appearance-based mapping (RTABMap) [12], [13], [14], a SLAM system with a global loop closure detection approach, is developed. It is able to construct large-scale and long-term map. Its global map is updated when a loop-closure is detected. However, without loop-closures been detected, it can not remove dynamic objects from the static map.

Motion detection, i.e. detecting motions of objects relative to the background, is a basic step in security monitoring system. There are many approaches proposed to adapt to different scenarios, such as image difference, background subtraction [7], optical flow[22], etc. The work in [19] presents a algorithm for detecting dynamic objects from a static background based on frame difference. The absolute difference is calculated between the consecutive frames. Then, the difference image is binarized. Finally, dynamic objects are marked in the binary image. Similar work is accomplished in [17]. However, this work considers the effect of the speed of a object: When the speed is two low, parts of the dynamic object will be missing in the binary image. Double-difference image was proposed in [11] to solve the above mentioned problem. A double-difference image is obtained by an AND operation between image frames k and $k-1$, and then between frames k and $k+1$. So motion regions of double-difference image keeps the shape of the dynamic objects at time t.

Most dynamic-object-detection algorithms are proposed for static cameras and can be easily affected by illumination changes. The work in [10] presents a solution to detecting dynamic objects under motions of both objects and sensors. Moving objects are estimated using image difference and adaptive particle filter. However, this method is designed for 2-dimensional (2D) scenarios. The method proposed in [16] can remove moving objects in 3-dimensional (3D) space. It finds moving objects by segment out individual clusters and analyze the spatial relationship of each cluster. If the position of an object with respect to other objects changes between two views, this object is considered a moving object. The major limitations of this approach are that it uses only depth images from a static camera, and it can hardly be implemented for real-time applications.

III. Technical Approach

Overview of our approach is shown in Fig. 1. Considering a moving camera, we transform two selected successive images to the same image coordinate frame through feature matching and compute the homography matrix. Then, we detect candidate image pixels of dynamic objects in the RGB images using image differences between two gray-scale images converted from two RGB images. Depth information is used to locate the objects in each of the two images. The point cloud produced from the RGB-D images are segmented into individual clusters after filtering and downsampling, so that we can find the correspondences between dynamic objects and individual clusters. Finally, we obtain static scene by removing dynamic objects from the point clouds and their corresponding RGB-D images.

A. *Obtaining pixel correspondences*

Due to motions of the RGB-D camera mounted on a mobile robot, to make image difference between two RGB images, we need to transform them to the same image coordinate system. This can be done by calculating their homography matrix with feature matches. First, both RGB images are converted to gray-scale images for further image processing. Then, we match two gray-scale images using image features. Researchers have proposed a number of image features, like SIFT, SURF and ORB. The SIFT and SURF features have been successfully applied in object detection and recognition applications. However, they are not efficient enough for real-time visual SLAM applications. As a result, computationally-efficient feature detector and descriptor are required. Thus, We use ORB feature [18] for feature matching to find the homography between two images.

ORB features are computed in both gray-scale images at time-stamp T_{k-1} (the last frame), as illustrated in Fig. 2a, and time-stamp T_k (the current frame), as shown in Fig. 2b. The time interval $\Delta T = T_k - T_{k-1}$ should be able to ensure sufficient movement of the dynamic objects between the last frame and the current frame. In this paper, we chose ΔT to be one-third second. Brutforce matching with cross checking is used to find feature matches between the two images. The result of feature matching is illustrated in Fig. 2c. Since false matches and matches corresponds to features of dynamic objects may affect the accuracy of the final homography, we utilize RANdom SAmple Consensus(RANSAC) [4] algorithm when computing homography using the OpenCV [2] implementation.

After the homography is calculated, we can find the pixel correspondences between the two images by transforming the last frame to the current image coordinate system. We use $P_L = (x_L, y_L)$ to denote a point which is located in the last frame. P_L is augmented to be $P_L = (x_L, y_L, 1)$ by adding 1 as the last element. Corresponding point of P_L in the current frame is denoted by $P_C = (x_C, y_C, 1)$. The relation between P_L and P_C, as shown in Fig. 3, is decided by the homography matrix

$$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}, \tag{1}$$

as

$$P_C{}' = s \begin{pmatrix} x_C \\ y_C \\ 1 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x_L \\ y_L \\ 1 \end{pmatrix}. \tag{2}$$

The corresponding coordinates in the current frame is $P_C = P_C{}'/s$, where s is a scale factor.

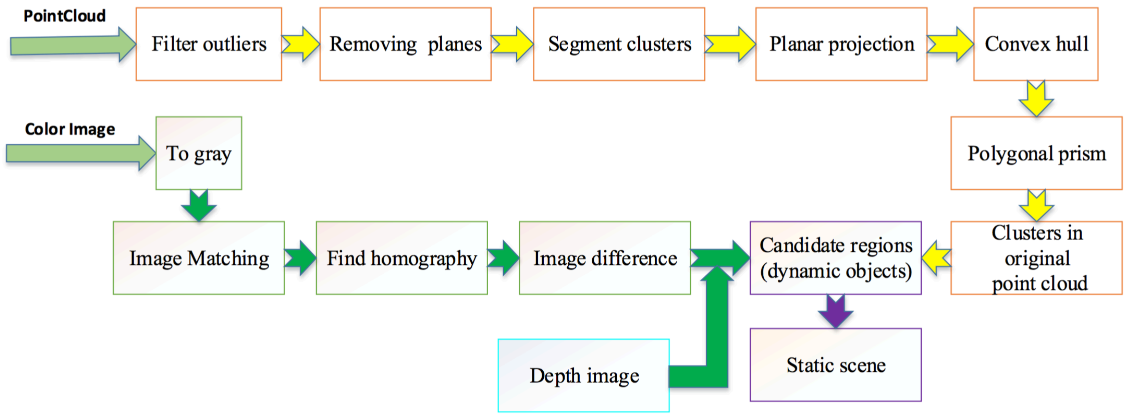


Figure 1: Overview of our approach for removing dynamic objects from RGB-D images.

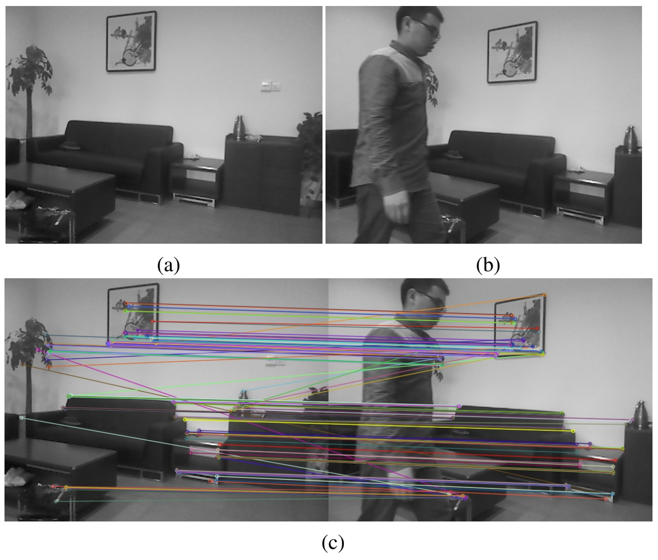


Figure 2: Feature matching result using ORB features. (a) the last frame. (b) the current frame. (c) the result of feature matching, with few false matches.

B. Motion detection

Before computing image difference between two images, Gaussian filter is applied to each gray-scale image to reduce image noises. With the pixel correspondences calculated from Eq. 2, we can obtain the absolute image differences between the two images. The resulted image differences are binarized with a threshold T_i. When the absolute difference of a pixel is larger than a threshold, this pixel is labeled in bright as a candidate pixel of a dynamic objects, and otherwise, labeled in dark. Unfortunately, many factors may cause incorrect binarization, e.g. illumination changes. To improve the accuracy of dynamic-object detecting, combining depth information in the binarization is a straightforward choice for RGB-D cameras. Another reason that we need depth

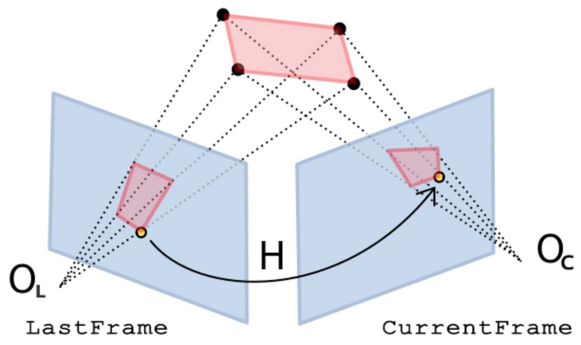


Figure 3: Pixel correspondences through homography between two images. With the homography, two images are transformed to the same image coordinate system for computing image differences.

information is that we have to find out in which frame does a dynamic object exists and results a candidate pixel in the binary image: in the last frame, or in the current frame. For each candidate pixel, we compare its depth values in the last depth image and the current depth image as follows.

Let the depth value of a candidate pixel P to be $d(x_L, y_L)$ in the last frame, the depth value to be $d(x_C, y_C)$ in the current frame, and the depth difference $d' = d(x_L, y_L) - d(x_C, y_C)$, the location of the dynamic object locates is defined in the following manner: If $d' > T_d$, where T_d is a predefined threshold, the dynamic object resulting P should be locating in the current frame. If $d' < -T_d$, the dynamic object should be locating in the last frame. For other cases, the candidate pixel P should be considered as a false detection. After we process each candidate pixel in this way, two binary images can be obtained: One labeling candidate pixels of dynamic objects in the last frame, the other one labeling that in the current frame. The results of this process will be shown in Sect. IV. The motion detection algorithm is shown in another form in Algorithm 1.

Algorithm 1 Motion Detection

```
Require: lastImage, currentImage, lastDepth
    currentDepth, $T_d$
Ensure: lastFrame, currentFrame
 1: for each $row \in [0, lastImage.rows - 1]$ do
 2:     for each $col \in [0, lastImage.cols - 1]$ do
 3:         $x_L \leftarrow row, y_L \leftarrow col, z \leftarrow 1$
 4:         $x' \leftarrow h_{11} \cdot x_L + h_{12} \cdot y_L + h_{13}$
 5:         $y' \leftarrow h_{21} \cdot x_L + h_{22} \cdot y_L + h_{23}$
 6:         $s \leftarrow h_{31} \cdot x_L + h_{32} \cdot y_L + h_{33}$
 7:         $x_C \leftarrow x' / s, y_C \leftarrow y' / s$
 8:         $diff \leftarrow abs(P_L(x_L, y_L) - P_C(x_C, y_C))$
 9:         $d_L \leftarrow depth(x_L, y_L)$
10:         $d_C \leftarrow depth(x_C, y_C)$
11:         if $d_L - d_C > T_d$ then
12:             $currentFrame(x_C, y_C) \leftarrow 255$
13:             $lastFrame(x_L, y_L) \leftarrow 0$
14:         else if $d_C - d_L > T_d$ then
15:             $lastFrame(x_L, y_L) \leftarrow 255$
16:             $currentFrame(x_C, y_C) \leftarrow 0$
17:         else
18:             $lastFrame(x_L, y_L) \leftarrow 0$
19:             $currentFrame(x_C, y_C) \leftarrow 0$
20:         end if
21:     end for
22: end for
```

C. Planar extraction

For a low-cost Kinect sensor, the precision of the raw point cloud, as produced directly from a color image and its associated depth image, would be affected by various factors, e.g. edges of surfaces and measurement noises. Thus, before extracting planar segments of the point cloud C_p, pre-processing is applied to smooth it. First, a passthrough filter is applied to C_p to obtain a filtered point cloud C_f: points will be discarded from C_p if their depth values are larger than a predefined threshold T_c. Furthermore, due to measurement noises, some points may have much larger depths than their neighbors. We assumed all points follow a normal distribution with a mean distance μ and a standard deviation σ to its κ neighbors. Let μ' and σ' be the global mean value and the global deviation, respectively, if $\mu > \mu' + \sigma'$, the points would be removed from the raw point cloud. When we perform a simple operation on each point of a cloud, the computational complexity is $O(n)$ (n being the number of points). If we search κ neighbors of a point, the computational complexity would be $O(\kappa n)$. Consequently, reasonably reducing the number of points in the point cloud to reduce the computational complexity would benefit real-time applications. This can be done by downsampling the point cloud. Therefore, the filtered point cloud C_f is divided into multiple voxels with a specific resolution T_v, the centriod of all points inside each voxel forms a downsampled point cloud C_d.

When dynamic objects are located on the ground, removing horizontal planes from the point cloud could ensure that these objects can be segmented out from the static scene later. Here, we consider most indoor scenarios in which the ground is a planar floor. In our system, we assume that the RGB-D camera is mounted in parallel with the ground plane within a specified angular deviation $\delta\alpha$. Then, let α be the angle between the normal direction of a planar segment C_s in C_d and the vertical direction in the camera coordinate system, if $\alpha < \delta\alpha$, C_s will be removed from C_d, forming a new point cloud C_r. Planes are extracted using a RANSAC scheme. Only those planes, which have more than a certain number of points locate in, would be considered in this step.

D. Point cloud segmentation

To retrieve point-cloud segments corresponding to independent objects, point cloud C_r need to be divided into several parts. Distances among points are taken into account to get individual segments by using Euclidean cluster extraction algorithm, which is implemented in PCL. The clustering process begins with selecting a point p_i. If the distance of p_i to its neighbor p_n is smaller than a threshold T_e, we add p_n to a queue Q. Then, we search the neighbors of all points in the queue Q and add their neighbors to Q, if again, the above mentioned distances is smaller than T_e and such neighbors are not added to Q yet. All the points in Q will be considered to be in one cluster when no new neighbors can be found. Finally, all points in the point cloud C_r are divided into different segments via iterations of of the above operations. To speed up this clustering process, we use $k-d$ tree to search neighbors of each point. Because we have removed the outliers and ground planes in the point cloud, potential dynamic objects could be segmented out from the background by setting proper threshold T_e.

E. Determining dynamic objects

Candidate image pixels and point-cloud segments of dynamic objects are expressed in different coordinate systems. Registering these two kinds of data requires to transform them to a same coordinate system to extract dynamic 3D objects. Assuming a pin-hole camera model, each point-cloud segment S_i could be projected to the image coordinate frame. The rectangular bounding box of the projection of S_i can be extracted as B_i, with its four corners marked as x_{min}, x_{max}, y_{min} and y_{max}. If the depth value of a candidate pixel in B_i is within the minimum depth z_{min} and the maximum depth z_{max} of S_i, we consider this candidate pixel is valid for a dynamic object corresponding to S_i. If the number of candidate pixels located in B_i meets certain conditions, we consider the point-cloud segment S_i represents a true dynamic object. Detail algorithm of the above process is shown in Algorithm 2, where point (x', y', z') is represented in the camera coordinate system, point (x_0, y_0) is represented in the image coordinate systems, and B_{3d} represents the 3D bounding box.

Algorithm 2 Determining dynamic objects

```
Require: $count$ is the number of clusters
    $f_x\ f_y\ c_x\ c_y$ is the camera model
 1: for each $i \in [0, count-1]$ do
 2:   for each $j \in [0, cluster.size-1]$ do
 3:     $x_0 \leftarrow (x' * f_x)/z' + c_x$;
 4:     $y_0 \leftarrow (y' * f_y)/z' + c_y$
 5:     $B_{3d} \leftarrow (x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max})$
 6:   end for
 7:   $num \leftarrow 0$
 8:   for each $row \in [0, currentImage.rows-1]$ do
 9:     for each $col \in [0, currentImage.cols-1]$ do
10:       if $currentFrame(row, col) == 255$ then
11:         if $row > min_x$ and $row < max_x$ and $col > min_y$
            and $col < max_y$ then
12:           if $cluster(row, col) > min_z$ and
              $cluster(row, col) < max_z$ then
13:             $num \leftarrow num + 1$
14:           end if
15:         end if
16:       end if
17:     end for
18:   end for
19:   if $num > T_r$ then
20:     return true
21:   end if
22: end for
```

F. Completing and removing the dynamic objects

Till now, point-cloud segments (S_i) of dynamic objects have been found in the point cloud C_r. However, these segments have been downsampled for efficient-computation purpose. Hence, we need to turn to the original point cloud C_p to extract the complete dynamic objects. We project each segment S_i to a specific plane. Then, the convex hull of the projection can be retrieved, which is the minimum convex set in Euclidean space.

The convex hull H_i corresponding to S_i can be used to form a polygonal prism P_i, which is a 3D regular prism with H_i as its polygonal base. The height of P_i is set to be able to extract the complete object corresponding to S_i. All the points in C_p which locate inside of P_i are assumed to form a complete dynamic object. Finally, we can remove dynamic objects from the original point cloud, and thus, the corresponding image pixels in the RGB-D images.

IV. Experiments and Results

To evaluate the performance of detecting and removing dynamic objects, using a Thinkpad X240 2.10 GHz Intel Core i7-4600U 8G memory, we run our algorithm in four different indoor scenarios. In the experiments, a Kinect sensor is mounted on a turtlebot. When the mobile robot moves in the scenes, dynamic objects in the field-of-view of the camera are also moving.

The results of image differences, as obtained in Sect. III-B, are shown in Fig. 4c. The bright pixels in Fig. 4c are the candidate pixels of dynamic objects. Then, utilizing the depth information from depth image, these bright pixels are assigned to dynamic objects in the last frame and in the current frame, as illustrated in Fig. 4d and Fig. 4e.

The results of removing dynamic objects from original point clouds are shown in Fig. 5. Referring to the original images in the first two rows, we can find that dynamic objects are removed from the corresponding point clouds in the two bottom rows. In the first scene corresponding to the column of Fig. 5a, the average size of feature matches is 143, and the success rate of removing complete dynamic objects is 83.33%. In second scene shown in the column of Fig. 5b, more valid textures can be extracted, resulting 303 average feature matches and a success rate of 95.83%.

In the third scene, we perform our algorithm in two scenarios, with a single dynamic object and with two different dynamic objects, the results of which are shown in Fig. 5c and Fig. 5d, respectively. The average size of feature matches is 200 in this scene. There are 4.54% dynamic objects failed to be removed in the single dynamic-object scenario, which can be mainly caused by incorrect homography between the last and the current images. When there are no enough good feature matches, or the matches are dominated by false ones, incorrect homography may be resulted in. In the dual-dynamic-object scenario, 76.67% of the dynamic objects are successfully removed completely, and another 20% are removed partially. The incomplete segmentation of the point cloud corresponding to dynamic objects will result in incomplete removal of the objects. The ground truth data are obtained by manually analyzing the processed images.

The statistics of the above object-removing results are also depicted in Fig. 6. Despite doing experiments in the above different scenarios, robust performance of our algorithm can be achieved. We run our algorithm with the same parameters in all scenes. Some important parameters mentioned in Sect. III are shown in Table. I.

Average time costs of different processes of our algorithm are listed in Table. II. The process of calculating homography matrix includes image matching and computing homography matrix. Computing point cloud means calculating the point cloud C_p from a pair of RGB-D images. Furthermore, the time cost of finding candidate pixels includes the processes in Section. III-B, while removing ground plane includes the processes in Section. III-C. Moreover, removing dynamic objects represents all the processes in Section. III-D, Section. III-E and Section. III-F. Extracting

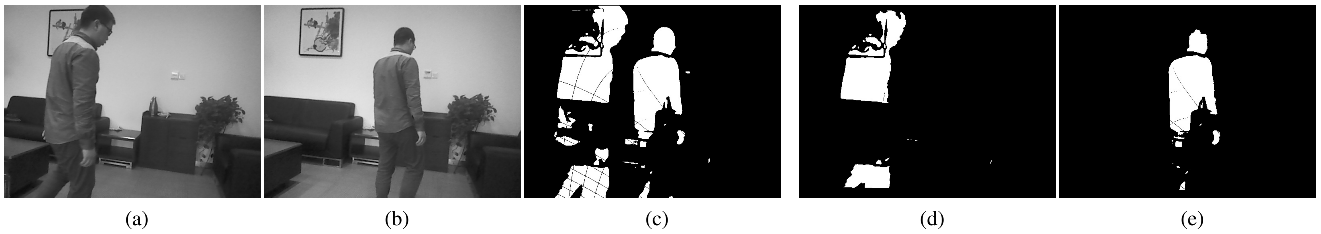


Figure 4: Motion detection results based on image differences and depth images. (a) The last image, as described in Sect. III-A. (b) The current image. (c) The results of image differences. (d) Assigning the candidate pixels in (c) to dynamic objects in the last image and (e) to that in the current image.

Table I: Parameters

T_i	$T_d(m)$	$T_c(m)$	$T_e(m)$	$P_i(m)$	$T_v(m)$	T_r
40	0.2	0.1	0.06	0.020	0.2	3000

point-cloud segments included in Section. III-F costs the largest portion of the time. Nevertheless, since the total time cost is only around 0.45 second, it would not affect the real-time performance of the mapping of a SLAM system, which usually has a much lower frequency than that of the pose tracking task in SLAM.

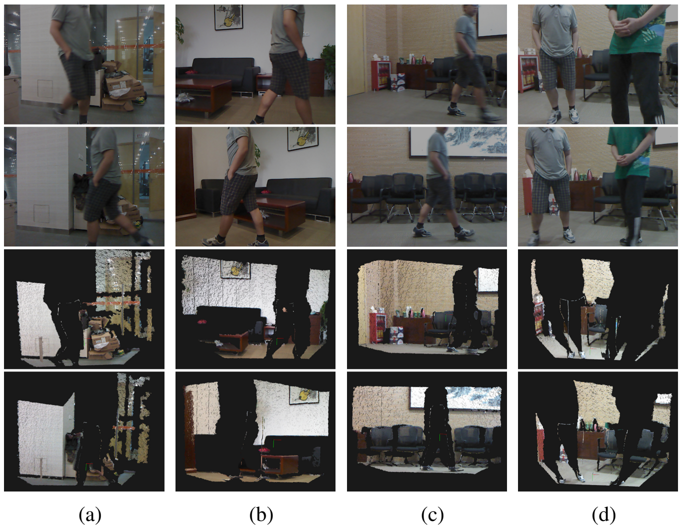
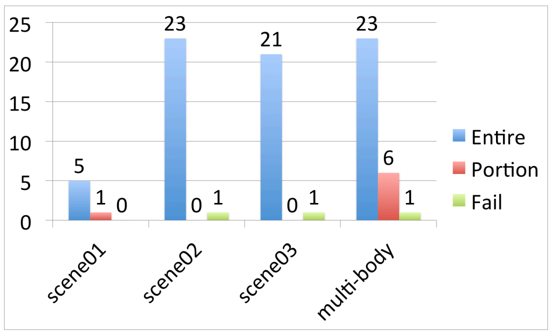


Figure 5: The results of removing dynamic objects. Top two rows: original point cloud produced by the last RGB-D images and the current RGB-D images, from top to bottom, respectively. Bottom two rows: point clouds after removing dynamic objects from the original point clouds.



(a)

Figure 6: The number of dynamic objects been removed in the image sequences of different scenarios. The number of objects been entirely removed is marked as *Entire*, while partially-removed objects marking with *Portion* and those failed to be removed marked with *Fail*.

Table II: Time costs

Modules	Running Time(ms)
Calculating homography matrix	73.24
Computing point cloud	10.25
Finding candidate pixels	22.27
Downsampling point cloud	40.40
Removing ground plane	6.75
Removing dynamic objects	335.10
Total Time	448.01

V. Conclusions and future work

We proposed a solution to remove dynamic objects from RGB-D images of a moving Kinect sensor. By eliminating dynamic objects from the original point cloud, the static scene can be built for RGB-D mapping. Unlike most of those existing motion detection algorithms which are designed for static cameras, we solve the problem by computing image differences after transforming two images into a same coordinate system with their homography. Depth information is also utilized for more accurate motion detecting. Point clouds are used to finally locate dynamic objects. In our experiments, we achieved robust performance of removing dynamic objects in different indoor scenarios.

In future work, we will integrate our algorithm into a RGB-D SLAM system to achieve better mapping performance, and thus, better pose tracking performance. On the other hand, camera-pose estimates from the SLAM system can be used to facilitate more robust homography computing to improve the success rate of removing dynamic objects.

Acknowledgment

This work is supported by Research on Foundations of Major Applications, Research Programs of NUDT, Project ZDYYJCYJ20140601.

References

[1] Cedric Audras, A Comport, Maxime Meilland, and Patrick Rives. Real-time dense appearance-based slam for rgb-d sensors. In *Australasian Conf. on Robotics and Automation*, 2011.

[2] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library.* " O'Reilly Media, Inc.", 2008.

[3] Christian Brenneke, Oliver Wulf, and Bernardo Wagner. Using 3d laser range data for slam in outdoor environments. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 1, pages 188–193. IEEE, 2003.

[4] H Cantzler. Random sample consensus (ransac). *Institute for Perception, Action and Behaviour, Division of Informatics, University of Edinburgh*, 1981.

[5] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):1052–1067, 2007.

[6] Ethan Eade and Tom Drummond. Scalable monocular slam. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 469–476. IEEE, 2006.

[7] Ahmed Elgammal, David Harwood, and Larry Davis. Non-parametric model for background subtraction. In *Computer VisionECCV 2000*, pages 751–767. Springer, 2000.

[8] Felix Endres, Jürgen Hess, Nikolas Engelhard, Jürgen Sturm, Daniel Cremers, and Wolfram Burgard. An evaluation of the rgb-d slam system. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1691–1696. IEEE, 2012.

[9] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments. *The International Journal of Robotics Research*, 31(5):647–663, 2012.

[10] Boyoon Jung and Gaurav S Sukhatme. Detecting moving objects using a single camera on a mobile robot in an outdoor environment. In *International Conference on Intelligent Autonomous Systems*, pages 980–987, 2004.

[11] Yoshinari Kameda and Michihiko Minoh. A human motion estimation method using 3-successive video frames. In *International conference on virtual systems and multimedia*, pages 135–140, 1996.

[12] Mathieu Labbé and François Michaud. Memory management for real-time appearance-based loop closure detection. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 1271–1276. IEEE, 2011.

[13] Mathieu Labbe and Francois Michaud. Appearance-based loop closure detection for online large-scale and long-term operation. *Robotics, IEEE Transactions on*, 29(3):734–745, 2013.

[14] Mathieu Labbe and François Michaud. Online global loop closure detection for large-scale multi-session graph-based slam. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 2661–2666. IEEE, 2014.

[15] Thomas Lemaire, Cyrille Berger, Il-Kyun Jung, and Simon Lacroix. Vision-based slam: Stereo and monocular approaches. *International Journal of Computer Vision*, 74(3):343–364, 2007.

[16] Krystof Litomisky and Bir Bhanu. Removing moving objects from point cloud scenes. In *Advances in Depth Image Analysis and Applications*, pages 50–58. Springer, 2013.

[17] Ester Martínez-Martín and Ángel P del Pobil. Motion detection in static backgrounds. In *Robust Motion Detection in Real-Life Scenarios*, pages 5–42. Springer, 2012.

[18] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: an efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571. IEEE, 2011.

[19] Nishu Singla. Motion detection based on frame difference method. *International Journal of Information & Computation Technology., ISSN*, pages 0974–2239.

[20] Randall Smith, Matthew Self, and Peter Cheeseman. Estimating uncertain spatial relationships in robotics. In *Autonomous robot vehicles*, pages 167–193. Springer, 1990.

[21] Randall C Smith and Peter Cheeseman. On the representation and estimation of spatial uncertainty. *The international journal of Robotics Research*, 5(4):56–68, 1986.

[22] Ashit Talukder and Larry Matthies. Real-time detection of moving objects from moving vehicles using dense stereo and optical flow. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 4, pages 3718–3725. IEEE, 2004.

[23] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.