Multimed Tools Appl
DOI 10.1007/s11042-016-3275-8

Robot manipulator self-identification for surrounding obstacle detection

Xinyu Wang[1,3] · Chenguang Yang[2] · Zhaojie Ju[4] · Hongbin Ma[1,3] · Mengyin Fu[1,3]

Received: 25 May 2015 / Revised: 5 November 2015 / Accepted: 11 January 2016

Abstract Obstacle detection plays an important role for robot collision avoidance and motion planning. This paper focuses on the study of the collision prediction of a dual-arm robot based on a 3D point cloud. Firstly, a self-identification method is presented based on the over-segmentation approach and the forward kinematic model of the robot. Secondly, a simplified 3D model of the robot is generated using the segmented point cloud. Finally, a collision prediction algorithm is proposed to estimate the collision parameters in real-time. Experimental studies using the Kinect® sensor and the Baxter® robot have been performed to demonstrate the performance of the proposed algorithms.

Keywords Manipulator self-identification · Superpixel · Collision prediction · Point cloud

This work was partially supported by EPSRC grants EP/L026856/1 and EP/J004561/1 (BABEL) as well as NSFC grant 61473038.

✉ Chenguang Yang
cyang@theiet.org

[1] School of Automation, Beijing Institute of Technology, Beijing, 100081, People's Republic of China

[2] College of Engineering, Swansea University, Swansea, SA1 8EN, UK

[3] State Key Lab of Intelligent Control and Decision of Complex System, Beijing Institute of Technology, Beijing, 100081, People's Republic of China

[4] School of Computing, University of Portsmouth, Portsmouth, UK

1 Introduction

In recent years, the fields of robotic application have increased rapidly. Different from traditional working environment, such as the assembly line, robots are operating in more complex and dynamic environments for new tasks, such as collaborating with humans. As a prerequisite of collision avoidance and motion planning, obstacle detection is playing an important role to guarantee the safety of human co-workers, the surrounding facilities and the robot itself, and also to complete the tasks more intelligently.

Various types of sensors have been used in previous studies to achieve environmental perception. Most of the sensors provide a 3D point cloud of the manipulator and the surrounding environment. To understand the surrounding environment, the point cloud processing method can be quite different with different sensor arrangements, i.e., eye-in-hand arrangement and eye-to-hand arrangement. For the eye-in-hand arrangement, the sensors are mounted on the end-effector of the robot. In [12], Natarajan uses a 3D Time-Of-Flight (TOF) camera as a proximity sensor to identify the position of the obstacle near the elbow of the manipulator. In [11], Danial introduces a manipulator robot surface following algorithm using a 3D model of the vehicle body panels acquired by a network of rapid but low resolution RGB-D sensors. The advantages of eye-in-hand arrangement are providing higher resolution point cloud of the obstacle, and that the algorithm is relatively simple because the coordinate transformation is not needed to measure the distance between the robot and the obstacle. However, the limited viewing angle and inevitable occlusion of the vision will bring a lot of blind spots [2, 13, 14, 24]. For the eye-to-hand arrangement, the sensor is arranged outside the working space and senses the whole working space in all the time [15, 19]. In [14], Pan presents a new collision- and distance- query algorithm, which can efficiently handle large amounts of point cloud sensor data received at real-time rates. Sukmanee presents a method to distinguish between a manipulator and its surroundings using a depth sensor in [20], where the iterative least square (ILS) and iterative closest point (ICP) algorithms are used for coordinate calibration and the matching between the manipulators model and point cloud. The eye-to-hand arrangement can provide a large enough viewing angle, but an unavoidable problem is to segment the robot itself from the environment during the point cloud processing.

Self-identification is defined as a process to identify the robot itself in the 3D point cloud. In previous works, the general solutions of self-identification are based on the pre-built 3D models of the robots or simple neighbourhoods of the robot skeletons with predefined radiuses. The manipulators are simply detected and deleted from the point clouds based on 3D models or the neighbourhood regions [7, 10, 15, 20]. However, an accurate 3D model is not always available for a given robot, and the installation of accessories, such as additional grippers, may greatly reduce the accuracy and reusability of the 3D model. In addition, the inevitable calibration error and the simplified 3D model of the manipulator may also result in incomplete deletion. Some of the points on the robot may not be identified as the robot, but as surrounding environment. These remaining points near the manipulator will cause great trouble to the subsequent motion planning or obstacle avoidance control. On the other hand, when using the simple neighbourhood of the robot skeleton with a predefined radius [19], the radius has to be large enough to guarantee that every single part of the robot, including the convex portions, are covered by the neighbourhood.

Because of the inappropriately large radius, some obstacles near the robot may also be covered by the neighbourhood and be detected as part of the robot itself. In this case, the robot has to avoid the obstacle even if the obstacle is far away from the robot to prevent the obstacle from dropping into the blind area, which may reduce the flexibility of the robot.

A better way to segment the robot from the environment is to generate a 3D model automatically based on the point cloud. Several model generation approaches can be found in previous studies. In [3], a Curvature-based approach is presented to achieve multi-scale feature extraction from 3D meshes and unstructured point clouds. In [22], a novel shape-sensitive point sampling approach is presented to reconstruct a low-resolution point-cloud model, which can modify sampling regions adaptively. These methods can generate 3D models for a wide variety of objects, but are designed only for off-line applications. And the processing time of these methods is up to tens of seconds, which makes them unable to work in real time. To speed up the processing time, priori information and assumptions of the object or environment to be modelled are always helpful. In [23], a fast, automated, scalable generation method is proposed to generate the textured 3D models of indoor environments, where the regular shape of the wall and the ceiling have been taken into consideration. Similarly, the Manhattan world assumption is widely considered for the generation of the building structures [5, 18, 27].

In this paper, the kinematic model of the robot is used as the priori information. A self-identification method is proposed to automatically generate a simplified 3D model for the robot based on the point cloud in real time. The robot skeleton is firstly created by the forward kinematic model of the robot and matched with the point cloud by calibration. Thereafter, the point cloud is over-segmented into superpixels using continuous k-means clustering method and then segmented into several meaningful categories. Next, a simplified robot 3D model is generated based on the skeleton of the robot, and updated using the segmented points. Finally, the collision prediction method is designed based on the robot model and the obstacle points to achieve the real time collision avoidance feature.

Our proposed method distinguishes with most existing methods and the novelties are summarized as below:

(i) An over-segmentation method in 3D space is designed based on continuous k-means clustering method to over-segment the 3D point cloud into superpixels.
(ii) A segmentation method is proposed based on the forward kinematics of the robot to segment the superpixels into several meaningful categories.
(iii) A simplified 3D robot model is generated automatically and updated on-line based on the robot skeleton and the segmented superpixels.
(iv) A collision prediction method is proposed based on the simplified 3D robot model and the obstacle points.

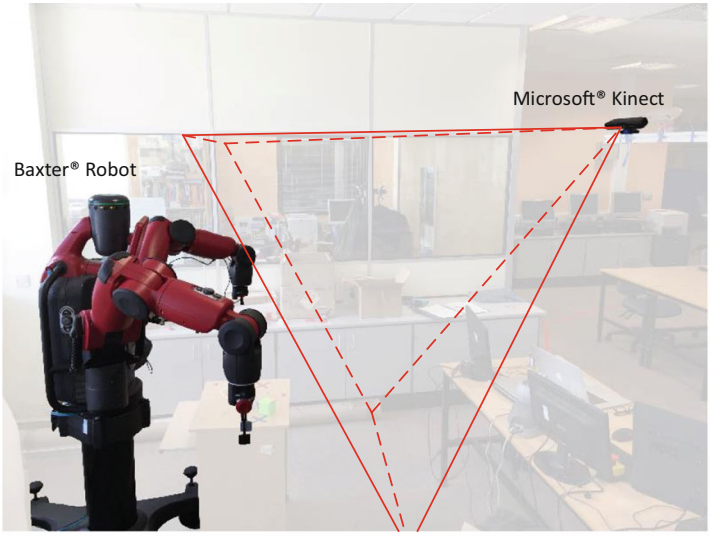
The remainder of this paper is organized as follows: Section 2 gives some preliminaries used in this paper. Section 3 shows the collision predication method, including self-identification and obstacle detection. Section 4 briefly introduces the control strategy of the robot to achieve the obstacle avoidance; Section 5 presents several experiments compared with the previous method to evaluate the proposed obstacle detection method; finally, we conclude this paper in Section 6.

2 Preliminaries

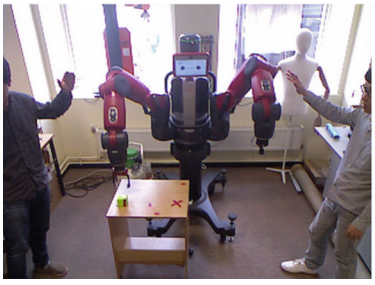
2.1 3D point cloud

As a low-cost typical RGB-D sensor, the Microsoft Kinect® sensor can generate a detailed point cloud with around 300,000 points at 30 Hz, and is widely used for the point cloud capture [1, 6, 11, 14–16, 20, 21, 25]. The raw data obtained from the Kinect® sensor includes an RGB image and a depth image, as shown in Fig. 1. A 3D point cloud in Cartesian coordinate is generated from these two images. Each pixel in the depth image corresponds to a point in the 3D point cloud. Denote the set of points in the 3D point cloud as $\boldsymbol{P} = \{\boldsymbol{p_i}\,(x_i, y_i, z_i)\,|i = 0, 1, \ldots, n\}$, where n is the total number of pixels in the depth image. The coordinate of the i^{th} point $\boldsymbol{p_i}$ is calculated by (1) [21].

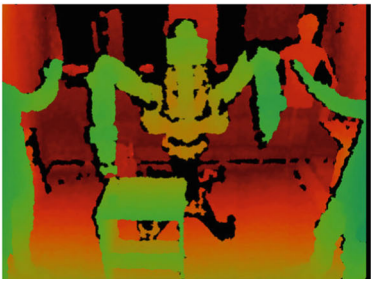
$$\begin{aligned} x_i &= d_i(x_i^c - c_x)/f_x \\ y_i &= d_i(y_i^c - c_y)/f_y \\ z_i &= d_i, \end{aligned} \tag{1}$$



(a) The system setup.



(b) The RGB image.



(c) The depth image.

Fig. 1 The raw data obtained from the Kinect® sensor

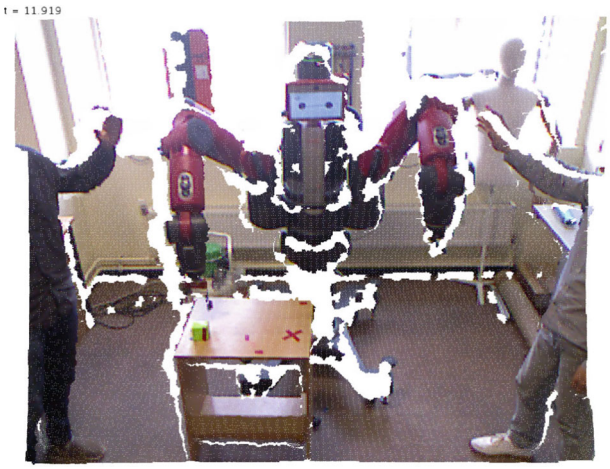


Fig. 2 The 3D point cloud

where d_i is the depth value of the i^{th} pixel in the depth image; (x_i^c, y_i^c) is the coordinate of the i^{th} pixel in the image coordinate system; c_x, c_y, f_x and f_y are the elements of intrinsic parameters of the depth camera.

The colors of the points are obtained from the corresponding pixels in the RGB image, which have the same coordinates as in the depth image. The generated point cloud is shown in Fig. 2.

2.2 Manipulator kinematic model

The forward kinematics solution is used to acquire the skeleton of the manipulator. The self-identification method that will be developed in Section 3.3 is designed based on the forward kinematics and the skeleton. In the skeleton, each link of the robot can be seen as a segment in 3-D space. The coordinates of the endpoints of the segments, i.e., the coordinates of the joints, in Cartesian space can be obtained based on the kinematic model of the robot, which can be found in our previous work in [4] for the Baxter® robot, as:

$$\boldsymbol{X_i'} = {}^{\boldsymbol{i-1}}\boldsymbol{A_i} \cdots {}^{\boldsymbol{1}}\boldsymbol{A_2}{}^{\boldsymbol{0}}\boldsymbol{A_1}\boldsymbol{X_0'}, i = 1, 2, \ldots, n, \tag{2}$$

where $\boldsymbol{X_i'} = [x_i', y_i', z_i', 1]^T$ is an augmented vector of the relative Cartesian coordinate of the i^{th} joint in the robot coordinate system; $\boldsymbol{X_0'}$ is the augmented coordinate of the base of the manipulator in the robot coordinate system; and ${}^{\boldsymbol{i-1}}\boldsymbol{A_i}$ is the link homogeneous transformation matrix, which can also be found in [4].

2.3 Calibration

The robot skeleton and the point cloud need to be placed into a unified coordinate system, in order to achieve self-identification of the manipulator. The coordinate transformation of the robot skeleton from the robot coordinates to the Kinect® coordinates can be obtained using homogeneous transformation as shown in (3).

$$\boldsymbol{X_i} = \boldsymbol{T}\boldsymbol{X_i'}, \tag{3}$$

where $\boldsymbol{T}$ is the transformation matrix, which will be defined below, $\boldsymbol{X_i} = [x_i\ y_i\ z_i\ 1]^T$ is the coordinate in the robot coordinate system, and $\boldsymbol{X'_i} = [x'_i\ y'_i\ z'_i\ 1]^T$ is the coordinate in the Kinect® coordinate system.

The transformation matrix $\boldsymbol{T}$ can be obtained by measuring the coordinates of four non-collinear points under the robot coordinates and the Kinect® coordinates. Assuming that we have four non-collinear points $\boldsymbol{s_1}$, $\boldsymbol{s_2}$, $\boldsymbol{s_3}$ and $\boldsymbol{s_4}$, the coordinates of the points in both the robot and the Kinect® coordinate system, $X - Y - Z$ and $X' - Y' - Z'$, are (x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3), (x_4, y_4, z_4), and (x'_1, y'_1, z'_1), (x'_2, y'_2, z'_2), (x'_3, y'_3, z'_3), (x'_4, y'_4, z'_4) respectively. The transfer matrix $\boldsymbol{T}$ from the Kinect® coordinates to the robot coordinates can be calculated by (4). The point cloud in the Kinect® coordinates and the transformed robot skeleton are shown in Fig. 3.

$$\boldsymbol{T} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x'_1 & x'_2 & x'_3 & x'_4 \\ y'_1 & y'_2 & y'_3 & y'_4 \\ z'_1 & z'_2 & z'_3 & z'_4 \\ 1 & 1 & 1 & 1 \end{bmatrix}^{-1} \tag{4}$$

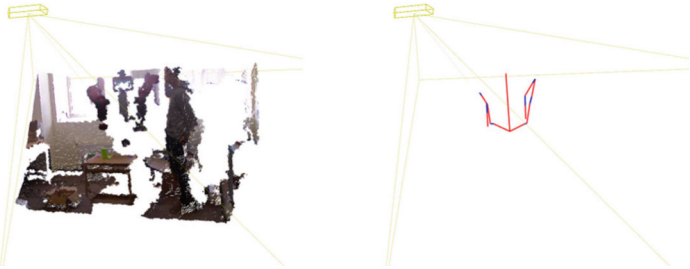
Remark 1 The coordinates of the four points in the robot coordinate system can be measured by moving the end-effector of the robot to the specified point and then using robot forward kinematics to calculate its position with respect to robot coordinate frame. The coordinates in the Kinect® coordinate system can be measured by manually selecting the points on the end-effector of the robot in the 3D point cloud.

Remark 2 Each pair of the coordinates satisfies (3). By measuring the coordinates of the four points, we can get four equations about $\boldsymbol{T}$. By solving these four equations together, the transformation matrix $\boldsymbol{T}$ can be obtained, as shown in (4).

3 Collision prediction

3.1 Region of interest

As the Kinect® sensor provides up to 300,000 points per frame, the point cloud processing algorithms can be highly time consuming. To improve the processing speed, a region



(a) The 3D Point Cloud in the Kinect® coordinates.

(b) The transformed robot skeleton.

Fig. 3 The calibration results

of interest (ROI) is determined based on the skeleton of the robot. Only the points with distances to the skeleton smaller than a pre-defined threshold r_{ROI} are considered in the following collision prediction algorithms. To obtain the minimum distance between a point and the robot skeleton, several interpolation points on each link of the skeleton are calculated as equal division points between two adjacent joints $\boldsymbol{X_i}$ and $\boldsymbol{X_{i+1}}$, as shown in (5).

$$\boldsymbol{x_k} = \boldsymbol{X_i} + \frac{j}{m}\left(\boldsymbol{X_{i+1}} - \boldsymbol{X_i}\right),\ i = 0, 1, \ldots, n,\ \ j = 0, 1, \ldots, m, \tag{5}$$

where $k = im + j$, $\boldsymbol{x_k}$ is the j^{th} interpolation points on the i^{th} link, m is the amount of the points that is interpolated on each link, and n is the total number of the links on the robot. Then, the distance between the point $\boldsymbol{p_i}$ and the robot skeleton is defined as the minimum distance between $\boldsymbol{p_i}$ and all the interpolated points $\boldsymbol{x_k}$ on the robot skeleton, and can be calculated by:

$$d\left(\boldsymbol{p_i}\right) = \min_{k=0,1,\ldots,m\times n} \left\|\boldsymbol{p_i} - \boldsymbol{x_k}\right\|. \tag{6}$$

Denote the set of points inside the ROI as $\boldsymbol{P_{ROI}} \subseteq \boldsymbol{P}$, then the points inside the ROI can be obtained by calculating the distances $d\left(\boldsymbol{p_i}\right)$ for all the points in the point cloud, and compare them with the predefined threshold r_{ROI}, as (7).

$$\boldsymbol{P_{ROI}} = \left\{\boldsymbol{p_i} \,|d\left(\boldsymbol{p_i}\right) < r_{ROI},\ \boldsymbol{p_i} \in \boldsymbol{P}\right\}. \tag{7}$$

The ROI is shown in Fig. 4, with $r_{ROI} = 0.4m$, where only 13.7 % points are included in the ROI.

3.2 Over-segmentation

The set of points $\boldsymbol{P_{ROI}}$ is firstly over-segmented into *superpixels* based on the k-means clustering method [9], which aims to partition the points in $\boldsymbol{P_{ROI}}$ into k_s sets

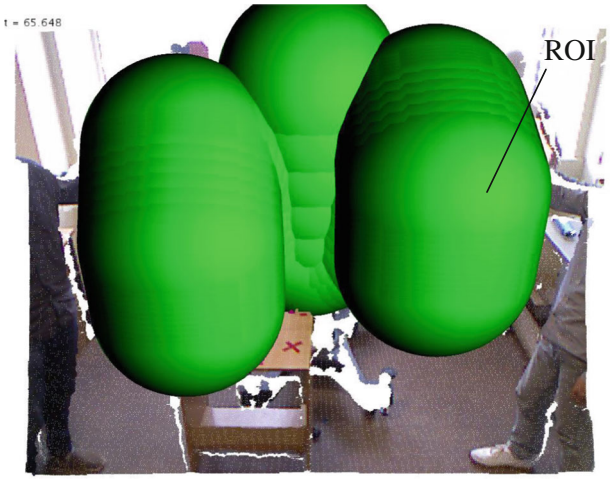
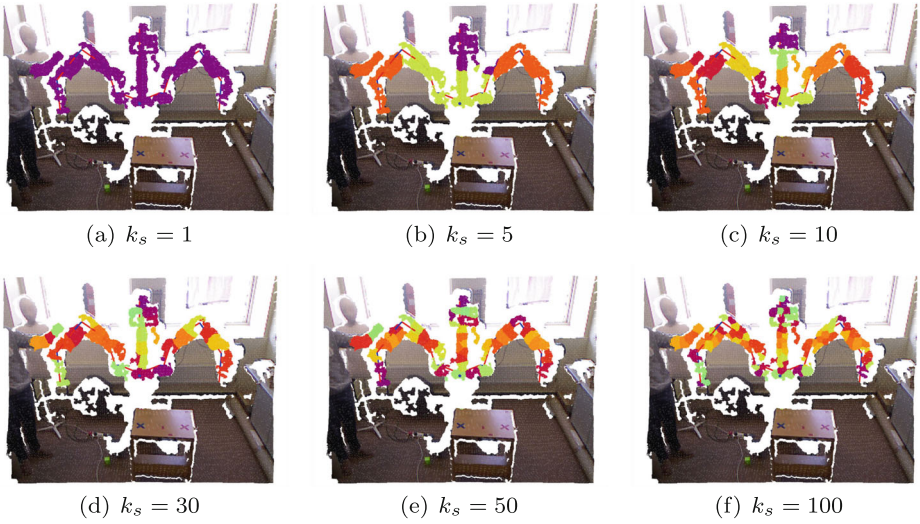


Fig. 4 The region of interest. $r_{ROI} = 0.4m$



(a) $k_s = 1$ (b) $k_s = 5$ (c) $k_s = 10$

(d) $k_s = 30$ (e) $k_s = 50$ (f) $k_s = 100$

Fig. 5 Superpixels with different k_s

$\boldsymbol{P_{ROI}} = \{\boldsymbol{S_1}, \boldsymbol{S_2}, \ldots, \boldsymbol{S_{k_s}}\}$ in order to minimize the within-cluster sum of squares (WCSS), or say

$$\arg\min_{\boldsymbol{P_{ROI}}} \sum_{i=1}^{k_s} \sum_{\boldsymbol{p} \in S_i} \|\boldsymbol{p} - \boldsymbol{\mu_i}\|^2 , \tag{8}$$

where $\boldsymbol{\mu_i}$ is the mean of the points in $\boldsymbol{S_i}$.

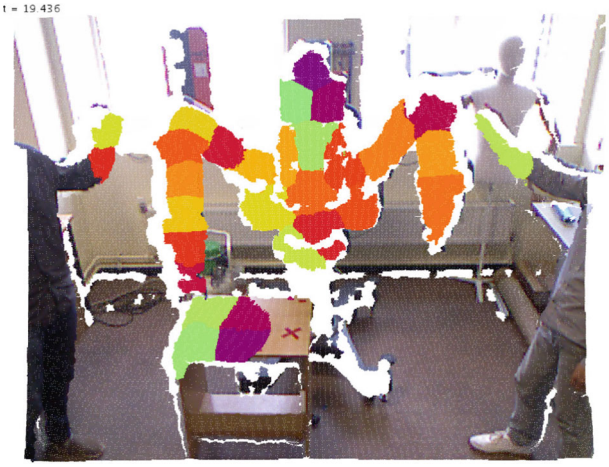


Fig. 6 Over-segmentation result of the $\boldsymbol{P_{ROI}}$ with $k_s = 30$

Definition 1 [17] The superpixels are sets of pixels that are roughly homogeneous in size and shape, and are local, coherent, and which preserve most of the structure necessary for segmentation at the scale of interest.

The amount of superpixels needs to be large enough to ensure that none of the superpixels contains more than one kinds of the following points, the robot, the obstacle and the object to be operated. The over-segmentation algorithm is shown in Algorithm 1 and the results are shown in Fig. 5 with different number of superpixels, and Fig. 6 with the amount of superpixels $k_s = 30$.

Algorithm 1 *Over-segmentation*

Input:
Point cloud $\boldsymbol{P_{ROI}}$*, number of over-segmented sets* k_s*, and* $\boldsymbol{\mu_i}$ *from the last frame*

Output:
The over-segmented sets $\{\boldsymbol{S_1}, \boldsymbol{S_2}, \ldots, \boldsymbol{S_k}\}$
The means $\{\boldsymbol{\mu_1}, \boldsymbol{\mu_2}, \ldots, \boldsymbol{\mu_k}\}$

```
if P_ROI is the first frame then
    randomly select μ_i
else
    inherit μ_i from the last frame
end if
while convergence of μ_i has not reached do
    for each points p_j in P_ROI do
        for each mean point μ_i do
            calculate distance ‖p_j − μ_i‖
        end for
        put p_j into set S_i with minimum distance
    end for
    for each set S_i do
        calculate the mean of the points in S_i
        update μ_i
    end for
end while
```

Remark 3 The amount of superpixels k_s can be determined based on pilot experiments. Extra superpixels will increase the computational load. While small number of superpixels may lead to the problem that some superpixel may contain the robot and the obstacle at the same time. In this work, different k_s are tried to find the minimum value of k_s, which can split the obstacle and the robot into different superpixels, as shown in Fig. 5. Based on these experiments, k_s is eventually set as $k_s = 30$.

Remark 4 In classic k-means clustering algorithms, the initial means $\boldsymbol{\mu_i}$ are selected randomly for each set of the observations, which will cause unstable clustering results between different frames of point cloud. As the over-segmentation needs to be implemented on each frame, and a smooth clustering result is important for the control strategy, a continu-

ous k-means clustering method is proposed. In this clustering method, the initial means $\boldsymbol{\mu_i}$ are selected randomly only on the first frame, and for the following frames the means are inherited from the last clustering result,

3.3 Self-identification

3.3.1 Simplified model

Unlike the previous methods [15, 20], our proposed method does not depend on the accurate 3D model of the robot. Instead of using the priori 3D model, we have designed a self-identification method based on the 3D point cloud to automatically generate a simplified 3D model of the robot, which is suitable for the obstacle detection. In this model, the robot is presented by several spheres along the skeleton of the robot. The centres of the spheres are the interpolation points $\boldsymbol{x_k}$, which is obtained from (5). The initial radiuses of the spheres are set to 0, as shown in Fig. 7, where the red thick lines are the skeleton of the robot, and the green spheres are the initial simplified model of the robot. The self-identification process is to estimate the radius of each sphere to approximate the real situation based on the segmentation of the point cloud.

3.3.2 Segmentation of the robot

The set of points on the robot $\boldsymbol{P_r} \subseteq \boldsymbol{P_{ROI}}$ is segmented from $\boldsymbol{P_{ROI}}$ to estimate the simplified robot model. As $\boldsymbol{P_{ROI}}$ is already over-segmented, the segmentation is based on the superpixels $\{\boldsymbol{S_1}, \boldsymbol{S_2}, \ldots, \boldsymbol{S_k}\}$. The superpixels on the robot are the superpixels that have at least one point near the robot skeleton. In other words,

$$\boldsymbol{P_r} = \{\boldsymbol{S_i} \,|\exists\, \boldsymbol{p} \in \boldsymbol{S_i} : d(\boldsymbol{p}) < d_{TH}, \boldsymbol{S_i} \in \boldsymbol{P_{ROI}}\} \tag{9}$$

where d_{TH} is a predefined distance threshold. Thus, the robot segmentation algorithm is designed as shown in Algorithm 2. The segmented points on the robot $\boldsymbol{P_r}$ are shown in Fig. 8

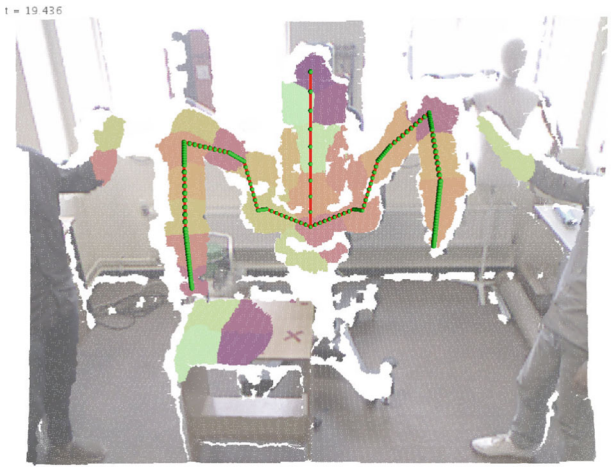


Fig. 7 The simplified model

Algorithm 2 *Segmentation of the Robot*

Input:
The superpixels $\{\boldsymbol{S_1}, \boldsymbol{S_2}, \ldots, \boldsymbol{S_k}\}$,
The interpolation points of the robot skeleton $\{\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_{m\times n}}\}$

Output:
The set of points on the robot $\boldsymbol{P_r}$

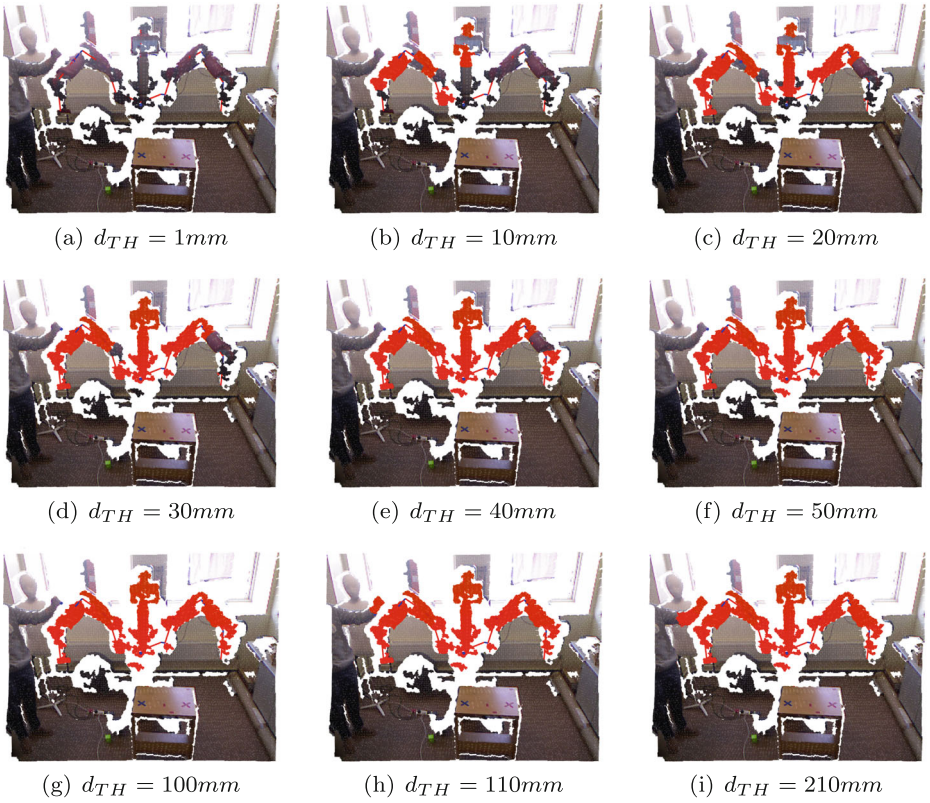
```
for each superpixel S_i do
    for each point p_j in S_i do
        calculate distance to the skeleton d(p_j) by (6)
        if d(p_j) < d_TH then
            put all the points in S_i into P_r
            break
        end if
    end for
end for
```

Remark 5 The threshold d_{TH} is determined based on the ground truth of the segmentation, by searching the minimum distances $d(\boldsymbol{p})$ of the points in each super pixel on the robot. The threshold should be just larger than all the minimum distances. The segmentation results with different d_{TH} are shown in Fig. 9. As can be seen, there is a large acceptable range for d_{TH}, from $50mm$ to $100mm$. A too small d_{TH} will result in an incomplete segmentation of the robot, as shown in Figs. 9a–e. And a too large d_{TH} may make the obstacle been identified as a part of the robot body. Thus, $d_{TH} = 50mm$ is chosen in this work particularly for the Baxter® robot.

3.3.3 Model update law

The model update law is used to update the radius of each sphere on the simplified robot model, so as to let the robot model exactly contain all the points in $\boldsymbol{P_r}$. To achieve this, the set of points $\boldsymbol{P_r}$ is re-segmented for the spheres based on distances, as $\boldsymbol{P_r} = \{\boldsymbol{X_1}, \boldsymbol{X_2}, \ldots, \boldsymbol{X_{m\times n}}\}$. The centres of the spheres $\boldsymbol{x_k}$ can be seen as the center of the corresponding sub-set $\boldsymbol{X_k}$ and the points with minimum distances to $\boldsymbol{x_k}$ are added into $\boldsymbol{X_k}$. The

Fig. 8 Segmentation result with $d_{TH} = 50mm$



(a) $d_{TH} = 1mm$ (b) $d_{TH} = 10mm$ (c) $d_{TH} = 20mm$

(d) $d_{TH} = 30mm$ (e) $d_{TH} = 40mm$ (f) $d_{TH} = 50mm$

(g) $d_{TH} = 100mm$ (h) $d_{TH} = 110mm$ (i) $d_{TH} = 210mm$

Fig. 9 Segmentation results with different d_{TH}

containing problem is now reformed as to let each sphere with a center of $\boldsymbol{x_k}$ exactly contains the points in the corresponding sub-set $\boldsymbol{X_k}$, which can be easily achieved by set the radius r_k as the maximum distance between the points in $\boldsymbol{X_k}$ and the sphere center $\boldsymbol{x_k}$.

In practice, the update law may also encounter two problems, i.e., the noise in the point cloud and the missing points caused by occlusion. For the first problem, as the noise in the

(a) Real Situation. (b) Self-Identification Result.

Fig. 10 Self-identification

point cloud will affect the stability of the model, a filter is designed for each of the sphere as (10).

$$r_k^*(t) = k_{fr} r_k^*(t-1) + (1 - k_{fr}) r_k(t),\ k = 1, 2, \ldots, m \times n, \tag{10}$$

where $r_k(t)$ is the raw radius at time t, $r_k^*(t)$ is the filtered radius at time t, and $k_{fr} \in (0, 1)$ is the factor of the filter. For the second problem, if most of the points in one sub-set $\boldsymbol{X_k}$ is not visible, the corresponding radius r_k will shrink rapidly, which is incorrect. To solve this problem, a threshold of the amount of the points in $\boldsymbol{X_k}$, T_k, is set as a prerequisite of the update law. If $\boldsymbol{X_k}$ contains less than T_k points, the update of r_k will be cancelled for this frame of data. The model update law is designed as shown in Algorithm 2. The identified robot model is shown in Fig. 10.

Algorithm 3 *Robot model update law*

Input:
The set of points on the robot $\boldsymbol{P_r}$,
The centres of the spheres $\{\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_{m\times n}}\}$,
The radius of the spheres from the lase frame $\left\{r_1^*, r_2^*, \ldots, r_{m\times n}^*\right\}$,

Output:
The updated radius of spheres $\left\{r_1^*, r_2^*, \ldots, r_{m\times n}^*\right\}$

if *the first frame* **then**
  set the initial radius $\left\{r_1^*, r_2^*, \ldots, r_{m\times n}^*\right\}$ *as 0*
end if
for *each point* $\boldsymbol{p_i}$ *in* $\boldsymbol{P_r}$ **do**
  for *each center of sphere* $\boldsymbol{x_j}$ **do**
    calculate distance $\|\boldsymbol{p_i} - \boldsymbol{x_j}\|$
  end for
  put $\boldsymbol{p_i}$ *into set* $\boldsymbol{X_j}$ *with minimum distance*
end for
for *each set* $\boldsymbol{X_k}$ **do**
  if *the amount of points in* $\boldsymbol{X_k}$ *is larger than* T_k
    for *each point* $\boldsymbol{p_i}$ *in* $\boldsymbol{X_k}$
      calculate distance $\|\boldsymbol{p_i} - \boldsymbol{x_k}\|$
    end for
    r_k = *the maximum distance*
    update radius r_k^* *using filter (10)*
  else
    inherit r_k^* *from the last frame*
  end if
end for

Remark 6 The threshold T_k is used to guarantee that the radius r_k will be updated only if X_k contains enough points to indicate the general size of the robot. It also relates to the density of the point cloud. The higher the density is, the larger T_k should be. In this work, the threshold is set as $T_k = 20$.

3.4 Obstacle detection

The surrounding points $\boldsymbol{P_{SUR}} = \boldsymbol{P_{ROI}} - \boldsymbol{P_r}$ can be further divided into four categories based on the superpixels, namely, the points on the obstacle for the left arm and the right arm $\boldsymbol{P_{ol}}$ and $\boldsymbol{P_{or}}$, the points on the object to be operated $\boldsymbol{P_d}$, and the other points. The operating object set of points $\boldsymbol{P_d} \subseteq \boldsymbol{P_{SUR}}$ is defined as the points in the superpixels with means $\boldsymbol{\mu}$ near the end-effector of the robot, which can be obtained by

$$\boldsymbol{P_d} = \left\{ \boldsymbol{S_i} \left| \|\boldsymbol{\mu_i} - \boldsymbol{X_n}\| < d_{pd}, \boldsymbol{S_i} \in \boldsymbol{P_{SUR}} \right. \right\}, \tag{11}$$

where $\boldsymbol{X_n}$ is the end-effector of the robot, $\boldsymbol{\mu_i}$ is the mean of the superpixel $\boldsymbol{S_i}$, d_{pd} is the distance threshold for calculating $\boldsymbol{P_d}$ and is set to $d_{pd} = 300mm$ in this work.

The obstacle sets $\boldsymbol{P_{ol}} \subseteq \boldsymbol{P_{SUR}} - \boldsymbol{P_d}$ and $\boldsymbol{P_{or}} \subseteq \boldsymbol{P_{SUR}} - \boldsymbol{P_d}$ are defined as the points in the superpixels with means $\boldsymbol{\mu}$ near the movable links of the manipulators. Take $\boldsymbol{P_{ol}}$ as an example, it can be obtained by

$$\boldsymbol{P_{ol}} = \{ \boldsymbol{S_i} \, | d_l(\boldsymbol{\mu_i}) < d_{obs}, \boldsymbol{S_i} \in \boldsymbol{P_{SUR}} - \boldsymbol{P_d} \}, \tag{12}$$

where $d_l(\cdot)$ is the distance to the movable links of the left arm, which is defined similar as (6) using the interpolated points on the movable links of the left arm, d_{obs} is the predefined detection range of the obstacles, and is set to $d_{obs} = 300mm$ in this work.

The four categories are shown in Fig. 11, where the green points indicates the obstacle points $\boldsymbol{P_{ol}}$ and $\boldsymbol{P_{or}}$, and the blue points indicates the points $\boldsymbol{P_d}$ on the object to be operated.

Remark 7 The points $\boldsymbol{P_d}$ on the object to be operated are found near the end-effector positions of the manipulators and is excluded from the obstacle points $\boldsymbol{P_{ol}}$ and $\boldsymbol{P_{or}}$. This is because that there will definitely be a collision between the operating object and the end-effector when the robot is manipulating the object. For instance, when the manipulator is grabbing the object with a gripper, the object will intrude into the gripper. However, these kinds of "collision" are intentional and should not be avoided.

3.5 Collision points estimation

The collision points, $\boldsymbol{p_{cr}} \in \{\boldsymbol{x_k} | k = 1, 2, \ldots, nm\}$ and $\boldsymbol{p_{co}} \in \boldsymbol{P_{ol}}$ *or* $\boldsymbol{P_{or}}$, are the two points either on the robot or on the obstacle, which covers the nearest distance between the robot and the obstacle, as shown in Fig. 11. The noise on the depth image from the Kinect®

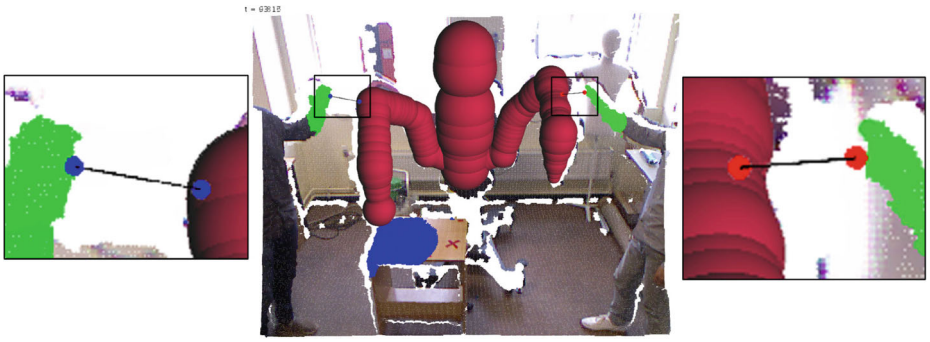


Fig. 11 Obstacle detection

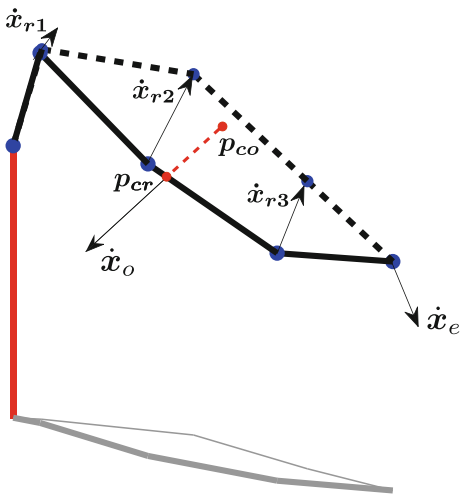


Fig. 12 The artificial parallel system built using Baxter® kinematic model [4]. The *solid black line* indicates the real manipulator. The *dashed black line* indicates the manipulator in the artificial parallel system

sensor will transfer into the point cloud, and will lead to frequent changes in $\boldsymbol{p_{co}}$. A first-order filter is designed to obtain a stable and smooth changing collision point $\boldsymbol{p^*_{co}}$, which is given by (13).

$$\boldsymbol{p^*_{co}}(t) = k_{fo}\boldsymbol{p^*_{co}}(t-1) + (1-k_{fo})\boldsymbol{p_{co}}(t), \tag{13}$$

where $\boldsymbol{p_{co}}(t)$ is the raw collision point on the obstacle at time t; $\boldsymbol{p^*_{co}}(t)$ is the filtered collision point; and $k_{fo} \in (0, 1)$ is the filtering strength parameter. With a larger k_f, the filtered collision point $\boldsymbol{p^*_{co}}(t)$ will change more smoothly yet with larger time delay. Thus, k_f is selected to guarantee both the control smoothness of the robot and a satisfactory reaction rate.

4 Robot control strategy

4.1 Inverse kinematics

In order to control the end-effector of the manipulator following the reference trajectory in Cartesian space, the numerical inverse kinematic method is used to calculate the reference joint velocities. Consider one manipulator arm of the robot. Denote the joint velocities as $\dot{\boldsymbol{\theta}}$, then the end-effector velocity can be given by (14).

$$\dot{\boldsymbol{x}} = \boldsymbol{J}\dot{\boldsymbol{\theta}}, \tag{14}$$

where $\dot{\boldsymbol{x}}$ is the end-effector velocity and $\boldsymbol{J}$ is the Jacobian matrix of the manipulator.

If the dimension of $\dot{\boldsymbol{\theta}}$ is larger than the dimension of $\dot{\boldsymbol{x}}$, i.e., the degree of freedom (DOF) of the manipulator arm is more than the number of the desired end-effector velocity components, the manipulator will become kinematically redundant, which can be used to achieve some secondary goals, e.g., obstacle avoidance, in addition to the end-effector trajectory following task.

For the inverse kinematic problem of the kinematically redundant manipulator, infinite number of solutions can be find. The general solution is given by (15).

$$\dot{\boldsymbol{\theta}} = \boldsymbol{J}^{\dagger}\dot{\boldsymbol{x}} + (I - \boldsymbol{J}^{\dagger}\boldsymbol{J})z, \tag{15}$$

where $\boldsymbol{J}^{\dagger} = \boldsymbol{J}^{T}(\boldsymbol{J}\boldsymbol{J}^{T})^{-1}$ is the pseudo-inverse of $\boldsymbol{J}$ and $\boldsymbol{z}$ is an arbitrary vector, which can be used to achieve the obstacle avoidance [8].

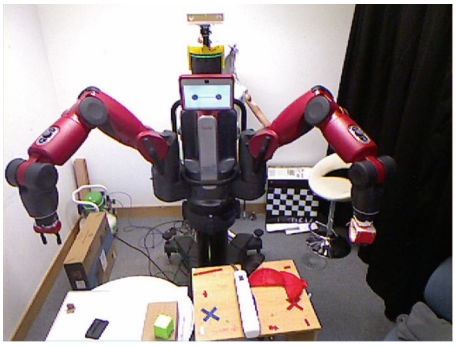
4.2 Collision avoidance strategy

When the collision points $\boldsymbol{p_{cr}}$ and $\boldsymbol{p_{co}}$ are found, the manipulator needs to move away from the obstacle. The desired velocity $\dot{\boldsymbol{x}}_o$ moving away from the obstacle is chosen as below in (16) according to our previous work [26].

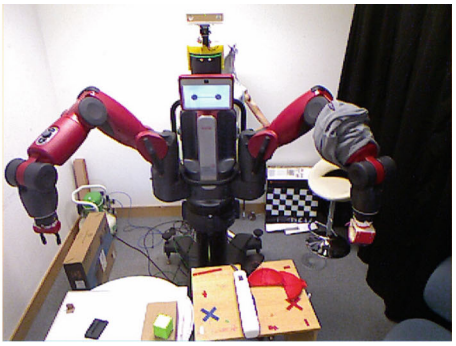
$$\dot{\boldsymbol{x}}_o = \begin{cases} \boldsymbol{0}\,, d \geq d_o \\ \dfrac{d_o - d}{d_o - d_c} v_{max} \dfrac{\boldsymbol{p_{cr}} - \boldsymbol{p_{co}}}{d}\,, d_c < d < d_o \\ v_{max} \dfrac{\boldsymbol{p_{cr}} - \boldsymbol{p_{co}}}{d}\,, d \leq d_c \end{cases}, \tag{16}$$

where $d = \|\boldsymbol{p_{cr}} - \boldsymbol{p_{co}}\|$ is the distance between the obstacle and the manipulator; v_{max} is the maximum obstacle avoidance velocity; d_o is the distance threshold that the manipulator starts to avoid the obstacle; d_c is the minimum acceptable distance and the manipulator will avoid at the maximum speed.

On the other hand, in order to eliminate the influence of the obstructing when the obstacle has been removed, the manipulator is expected to restore its original state. To achieve that, an artificial parallel system of the manipulator is designed in the controller in real time to simulate its pose without the influence of the obstacle, as shown in Fig. 12, where the dashed



(a) RGB image without clothes.

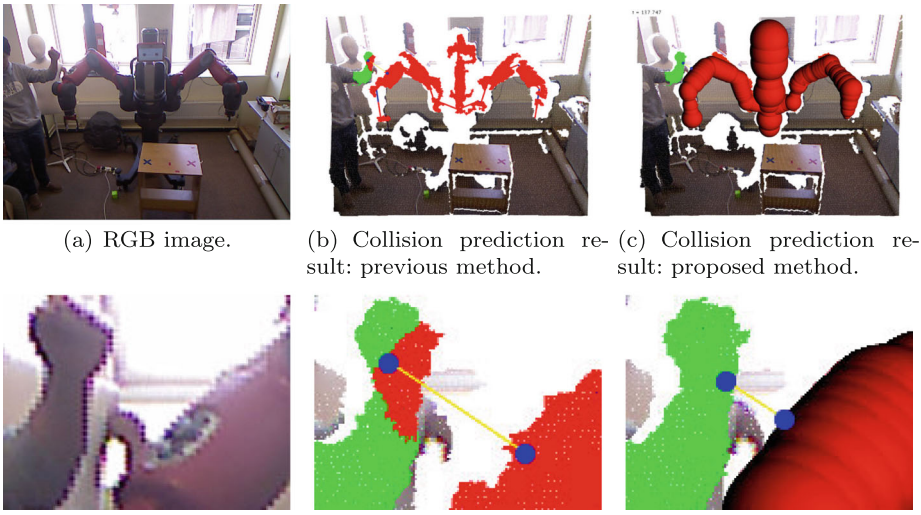


(b) RGB image with clothes.

(c) 3D model image without clothes.

(d) 3D model image with clothes.

Fig. 13 Self-Identification results with and without clothes on the left arm



(a) RGB image. (b) Collision prediction result: previous method. (c) Collision prediction result: proposed method.

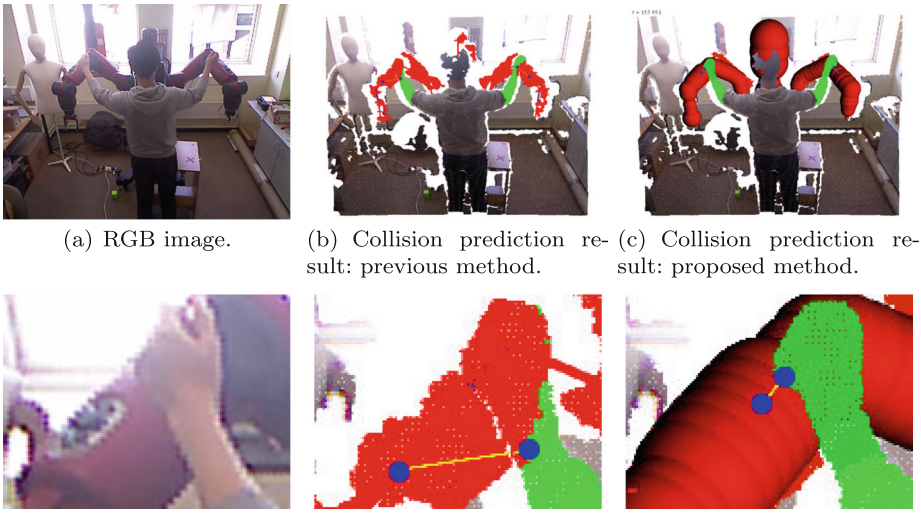
(d) Close-up of the collision area: RGB image. (e) Close-up of the collision area: previous method. (f) Close-up of the collision area: proposed method.

Fig. 14 Collision prediction results. Scene 1

black line indicates the parallel system. The restoring velocity $\dot{\boldsymbol{x}}_r$ is then designed as below [26]

$$\dot{\boldsymbol{x}}_r = \boldsymbol{K}_r \boldsymbol{e}_r, \tag{17}$$

where $\boldsymbol{K}_r$ is a symmetric positive definite matrix and $\boldsymbol{e}_r = [\boldsymbol{e}_{r1}\ \boldsymbol{e}_{r2}\ \boldsymbol{e}_{r3}]^T$ is the position errors of the joints between the parallel system and the real system.



(a) RGB image. (b) Collision prediction result: previous method. (c) Collision prediction result: proposed method.

(d) Close-up of the collision area: RGB image. (e) Close-up of the collision area: previous method. (f) Close-up of the collision area: proposed method.

Fig. 15 Collision prediction results. Scene 2

The control strategy can be obtained by (18), based on our previous work [26].

$$\dot{\boldsymbol{\theta}}_{\boldsymbol{d}} = \boldsymbol{J_e}^{\dagger}\left(\dot{\boldsymbol{x}}_d + \boldsymbol{K_e e_x}\right) + \left(\boldsymbol{I} - \boldsymbol{J_e^{\dagger}} \boldsymbol{J_e}\right)\left[\alpha \boldsymbol{z_o} + (1-\alpha)\boldsymbol{z_r}\right] \tag{18}$$

where

$$\boldsymbol{z_o} = \left[\dot{\boldsymbol{x}}_o^T \boldsymbol{J_o}\left(\boldsymbol{I} - \boldsymbol{J_e^{\dagger}} \boldsymbol{J_e}\right)\right]^{\dagger}\left(\dot{\boldsymbol{x}}_o^T \dot{\boldsymbol{x}}_o - \dot{\boldsymbol{x}}_o^T \boldsymbol{J_o} \boldsymbol{J_e^{\dagger}} \dot{\boldsymbol{x}}_e\right) \tag{19}$$

$$\boldsymbol{z_r} = \left[\dot{\boldsymbol{x}}_r^T \boldsymbol{J_r}\left(\boldsymbol{I} - \boldsymbol{J_e^{\dagger}} \boldsymbol{J_e}\right)\right]^{\dagger}\left(\dot{\boldsymbol{x}}_r^T \dot{\boldsymbol{x}}_r - \dot{\boldsymbol{x}}_r^T \boldsymbol{J_r} \boldsymbol{J_e^{\dagger}} \dot{\boldsymbol{x}}_e\right) \tag{20}$$

5 Experiments

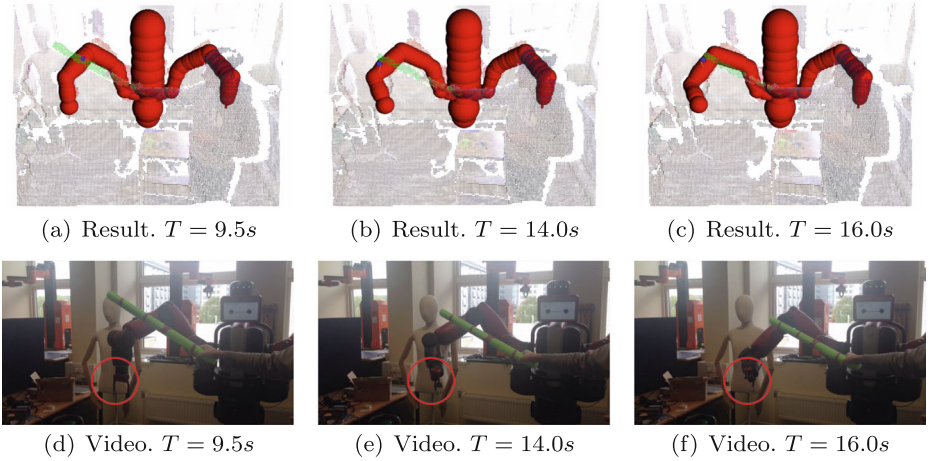
In order to verify the performance of the proposed collision prediction method, three groups of experiments are designed, i.e., self-identification, collision prediction and collision avoidance. The setup of the system is shown in Fig. 1c. The obstacle detection program is designed based on Java and run under Microsoft Windows 8.1 and Processing 2. The computer used for the obstacle detection program is equipped with an Intel® Core i5-3470 CPU at the clock speed of 3.2 GHz.

A typical previous obstacle detection method proposed in [15] is implemented as the contrast of the proposed method, which is described as follow. The previous method uses the neighbourhood of the robot skeleton with a fixed predefined radius to detect the obstacle. Specifically, all the points inside the neighbourhood is considered as the points on the robot; the points outside the neighbourhood and inside the ROI is considered as the obstacle. The radius of the neighbourhood is selected as small as possible under the premise of that no points on the robot will be considered as the obstacle with such a radius. In practice, several convex portions on the robot, e.g., the elbow, makes it impossible to set the radius small enough to fit most part of the robot. Otherwise, the convex portions will be considered as obstacles close to the robot and influence the obstacle avoidance control.

(a) Result. $T = 8.3s$ (b) Result. $T = 12.9s$ (c) Result. $T = 17.1s$

(d) Video. $T = 8.3s$ (e) Video. $T = 12.9s$ (f) Video. $T = 17.1s$

Fig. 16 Collision avoidance results and video frames based on the previous method with the ROI same as the proposed method



(a) Result. $T = 9.5s$ (b) Result. $T = 14.0s$ (c) Result. $T = 16.0s$

(d) Video. $T = 9.5s$ (e) Video. $T = 14.0s$ (f) Video. $T = 16.0s$

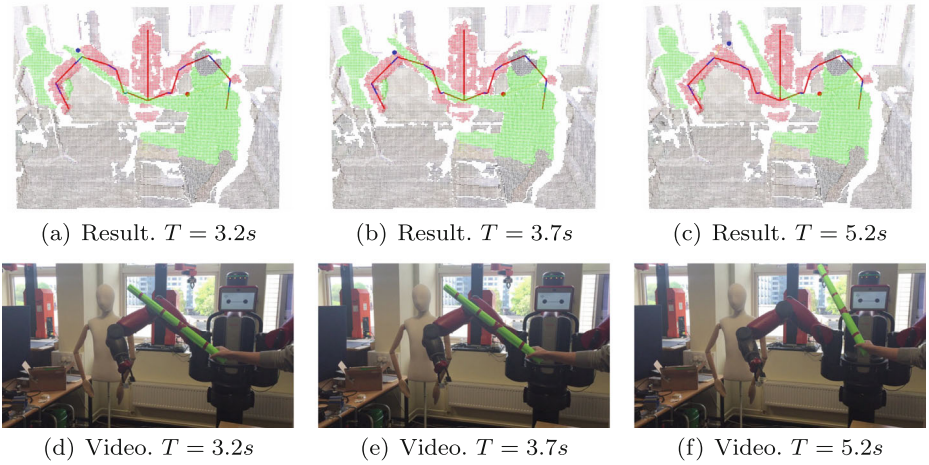
Fig. 17 Collision avoidance results and video frames based on the proposed method

5.1 Self-identification

In the first group of experiments, the proposed self-identification method is tested. In order to test the 3D model update law, the left arm of the Baxter® robot is wrapped with a piece of clothing to change the shape of the manipulator arm. The identification result with and without the wrapped clothes is shown in Fig. 13. As can be seen, when part of the arm has become thicker, the model changes automatically to fit the actual situation.

5.2 Collision prediction

In the second group of comparative experiments, the robot is set to a stationary pose to test the collision prediction quality of the two methods. The results are shown in Figs. 14 and



(a) Result. $T = 3.2s$ (b) Result. $T = 3.7s$ (c) Result. $T = 5.2s$

(d) Video. $T = 3.2s$ (e) Video. $T = 3.7s$ (f) Video. $T = 5.2s$

Fig. 18 Collision avoidance results and video frames based on the previous method with large ROI

Table 1 Evaluation of the proposed and the previous method

	Proposed method	Previous method with same ROI	Previous method with large ROI
Detection accuracy	97.6 %	71.5 %	94.9 %
Frame rate	11.7 fps	14.1 fps	12.6 fps

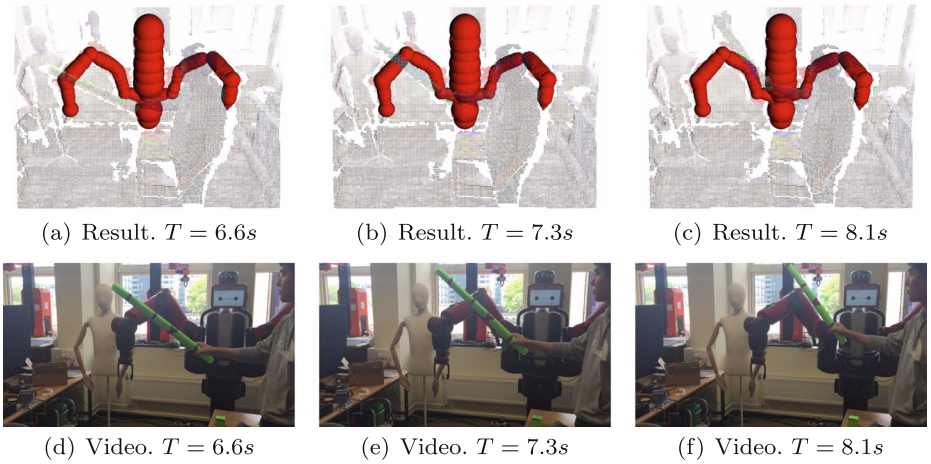
15 for two different scenes. Each of the figures contains the RGB image from the Kinect®, the result of the previous method and the proposed method.

As can be seen in Fig. 14b and e, based on the previous method, part of the human hand is identified as the robot, and the collision points is not quite accurate. When the obstacle is even closer to the robot, as in Fig. 15b and e, the whole hand is identified as the robot based on the previous method, and the collision points have a large deviation compared to the real situation. In contrast, based on the proposed method, the human hand is completely segmented from the robot, and the collision points are accurate enough for the collision avoidance task.

5.3 Collision avoidance

In the third group of comparative experiments, the collision prediction results are sent to the collision avoidance controller proposed in Section 4.

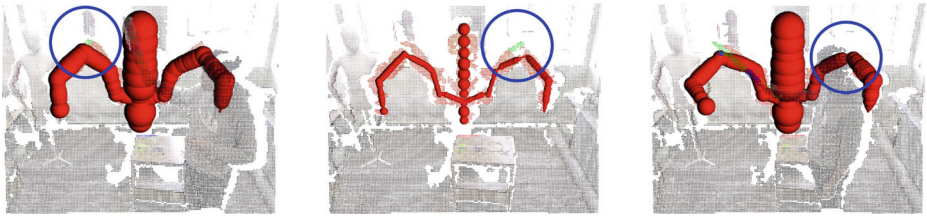
The ROI of the previous method is firstly set the same as the proposed method, the collision avoidance result is shown in Figs. 16 and 17. As can be seen in Fig. 16, based on the previous method, when the obstacle is moving close to the robot quickly, part of the obstacle is identified as the robot and the collision point is incorrectly far from the robot. Thus, the robot cannot move away from the obstacle. In contrast, based on the proposed method, the obstacle is always segmented from the robot, and the collision point is accurate. So the robot is able to avoid the collision.



(a) Result. $T = 6.6s$ (b) Result. $T = 7.3s$ (c) Result. $T = 8.1s$

(d) Video. $T = 6.6s$ (e) Video. $T = 7.3s$ (f) Video. $T = 8.1s$

Fig. 19 Collision avoidance results and video frames based on the proposed method



(a) (b) (c)

Fig. 20 Example of failed detection frames. The *blue circles* pointed out the faulty parts

Then, the ROI of the previous method is set twice larger than the proposed method to enable the robot avoid the obstacle in advance so that the obstacle is less likely to run into the neighbourhood of the skeleton and to be identified as the robot. However, the larger ROI brings a lot more unnecessary avoidance action. As can be seen in Fig. 18, the obstacle is passing by the robot with a quite large distance. While the robot changes its pose to avoid the obstacle. In contrast, based on the proposed method, the same passing by obstacle does not bring any unnecessary movements, as shown in Fig. 19.

The detection accuracy and the processing frame rate are shown in Table 1. As can be seen, the proposed method has a much higher detection accuracy than the previous method when they use a same ROI. When the previous method is implemented with a large ROI to reach an acceptable detection accuracy, it also brings a lot more unnecessary avoidance action. In addition, the proposed method is able to run in a satisfactory frame rate, which is only slightly slower than the previous method.

Figure 20 shows some examples of the failed detection frames, where blue circles pointed out the faulty parts. In Fig. 20a, the protruding portion on the elbow of the robot is detected as an obstacle. This is because the threshold d_{TH} is too small and the superpixel on the elbow is too far away from the robot skeleton. In Fig. 20b, the simplified 3D model has not been fully updated because the running time of the program is too short. The excessively small 3D model causes the wrong detection. In Fig. 20c, the missing in the original point cloud makes partial of the 3D model unable to be updated correctly.

Currently, these failed cases do not affect the obstacle avoidance control because these problems occur only occasionally and the filter specified in (13) is able to deal with them and to obtain a stable and smooth changing collision point.

In the future works, we will try to work out suitable solutions for these problems. A variable threshold d_{TH} and a threshold automatic turning strategy based on the generated 3D model may solve the problem in Fig. 20a. An improved model update law with faster convergence can deal with the problem in Fig. 20b. A global optimization method taking the radius of the neighbouring spheres into consideration for the 3D model update law may solve the problem in Fig. 20c.

6 Conclusions

In this paper, a self-identification method has been developed based on the 3D point cloud and the robot skeleton. The point cloud is firstly over-segmented into superpixels based on the continuous k-means clustering method. The superpixels on the robot are then identified

using the robot skeleton obtained from the forward kinematics. According to the segmented point cloud, a simplified 3D model of the robot is generated automatically in real-time. Based on the real-time 3D model, an accurate collision prediction method is proposed. Comparative experiments demonstrate that the robot system equipped with the proposed self-identification and collision prediction methods can avoid the collision, even if the obstacle is close to the robot, and at the same time, prevent unnecessary robot movements when the obstacle is far enough from the robot.

Nomenclature

$\boldsymbol{X_i'}$	An augmented vector of relative Cartesian coordinate of the i^{th} joint in the robot coordinate system
$\boldsymbol{X_0'}$	The augmented coordinate of the base of the manipulator in the robot coordinate system
$\boldsymbol{^{i-1}A_i}$	The link homogeneous transformation matrix
d_i	The depth value of the i^{th} pixel in the depth image
$x_i^c,\ y_i^c$	The coordinate of the i^{th} pixel in the image coordinate system
c_x, c_y	The principal point in the intrinsic parameter of the depth camera
f_x, f_y	The focal length in terms of pixels in the intrinsic parameter of the depth camera
$x_s,\ y_s,\ z_s$	The Cartesian coordinates of the end effectors of the manipulator
$x_m,\ y_m,\ z_m$	The Cartesian coordinates of the end effectors of Omni
$\boldsymbol{X_i}$	The coordinate of the i^{th} point in the Kinect® coordinate system
$\boldsymbol{x_k}$	The j^{th} interpolation points on the i^{th} link, $k = im + j$
$\boldsymbol{P_{ROI}}$	The set of points inside the ROI
r_{ROI}	The radius of the ROI
k_s	The amount of the superpixels
$\boldsymbol{S_i}$	The set of points in the i^{th} superpixel
$\boldsymbol{\mu_i}$	The mean of points in $\boldsymbol{S_i}$
$\boldsymbol{P_r}$	The set of points on the robot
$r_k(t)$	The raw radius of the k^{th} sphere on the robot model at time t
k_{fr}	The factor of the filter in the robot model update law
$\boldsymbol{P_{SUR}}$	The surrounding points of the robot
$\boldsymbol{P_{ol}}$	The points on the obstacle for the left arm
$\boldsymbol{P_{or}}$	The points on the obstacle for the right arm
$\boldsymbol{P_d}$	The set of points on the operating object
$\boldsymbol{p_{cr}}$	The collision point on the robot
$\boldsymbol{p_{co}}$	The collision point on the obstacle
$\boldsymbol{\dot{x}}$	The end-effector velocity
$\boldsymbol{J}$	The Jacobian matrix of the manipulator
$\boldsymbol{J}^{\dagger}$	The pseudo-inverse of $\boldsymbol{J}$
d	The distance between the obstacle and the manipulator
v_{max}	The maximum obstacle avoidance velocity
d_o	The distance threshold that the manipulator starts to avoid the obstacle
d_c	The minimum acceptable distance and the manipulator will avoid at the maximum speed
$\boldsymbol{e_r}$	The position errors of the joints between the parallel system and the real system

Acknowledgments The authors would like to thank Professor Angelo Cangelosi of Plymouth University for his technical support.

References

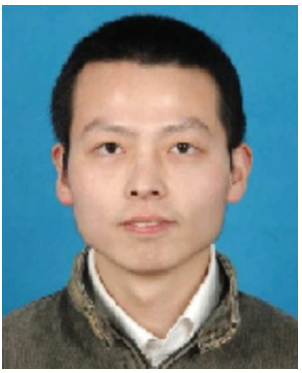
1. Belhadj H, Ben Hassen S, Kaaniche K, Mekki H (2013) Kuka robot control based kinect image analysis. In: 2013 International conference on individual and collective behaviors in robotics (ICBR), pp 21–26
2. Ben-Tzvi P, Charifa S, Shick M (2010) Extraction of 3d images using pitch-actuated 2d laser range finder for robotic vision. In: 2010 IEEE international workshop on robotic and sensors environments (ROSE), pp 1–6
3. Ho H, Gibbins D (2009) Curvature-based approach for multi-scale feature extraction from 3d meshes and unstructured point clouds. Comput Vis IET 3(4):201–212. doi:10.1049/iet-cvi.2009.0044
4. Ju Z, Yang C, Ma H (2014) Kinematics modeling and experimental verification of baxter robot. In: 2014 33rd Chinese control conference (CCC), pp 8518–8523
5. Li-Chee-Ming J, Gumerov D, Ciobanu T, Armenakis C (2009) Generation of three dimensional photo-realistic models from lidar and image data. In: 2009 IEEE toronto international conference science and technology for humanity (TIC-STH), pp 445–450. doi:10.1109/TIC-STH.2009.5444457
6. Luo R, Ko MC, Chung YT, Chatila R (2014) Repulsive reaction vector generator for whole-arm collision avoidance of 7-dof redundant robot manipulator. In: 2014 IEEE/ASME international conference on advanced intelligent mechatronics (AIM), pp 1036–1041
7. Lyubova N, Filliat D, Ivaldi S (2013) Improving object learning through manipulation and robot self-identification. In: 2013 IEEE international conference on robotics and biomimetics (ROBIO), pp 1365–1370. doi:10.1109/ROBIO.2013.6739655
8. Maciejewski AA, Klein CA (1985) Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments. Int J Robot Res 4(3):109–117
9. MacQueen J (1967) Some methods for classification and analysis of multivariate observations. In: Proceedings of the fifth berkeley symposium on mathematical statistics and probability, volume 1: statistics. University of California Press, Berkeley, pp 281–297
10. Michel P, Gold K, Scassellati B (2004) Motion-based robotic self-recognition. In: Proceedings. 2004 IEEE/RSJ international conference on intelligent robots and systems, 2004. (IROS 2004), vol 3, pp 2763–2768. doi:10.1109/IROS.2004.1389827
11. Nakhaeinia D, Fareh R, Payeur P, Laganiere R (2013) Trajectory planning for surface following with a manipulator under rgb-d visual guidance. In: 2013 IEEE international symposium on safety, security, and rescue robotics (SSRR), pp 1–6
12. Natarajan S, Vogt A, Kirchner F (2010) Dynamic collision avoidance for an anthropomorphic manipulator using a 3d tof camera. In: Robotics (ISR), 2010 41st international symposium on and 2010 6th German conference on robotics (ROBOTIK), pp 1–7
13. Ohno K, Kensuke K, Takeuchi E, Zhong L, Tsubota M, Tadokoro S (2011) Unknown object modeling on the basis of vision and pushing manipulation. In: 2011 IEEE international conference on robotics and biomimetics (ROBIO), pp 1942–1948
14. Pan J, Sucan I, Chitta S, Manocha D (2013) Real-time collision detection and distance computation on point cloud sensor data. In: 2013 IEEE International Conference on Robotics and Automation (ICRA), pp 3593–3599
15. Rakprayoon P, Ruchanurucks M, Coundoul A (2011) Kinect-based obstacle detection for manipulator. In: 2011 IEEE/SICE international symposium on system integration (SII), pp 68–73
16. Reddivari H, Yang C, Ju Z, Liang P, Li Z, Xu B (2014) Teleoperation control of baxter robot using body motion tracking. In: 2014 International conference on multisensor fusion and information integration for intelligent systems (MFI), pp 1–6
17. Ren X, Malik J (2003) Learning a classification model for segmentation. In: Ninth IEEE international conference on computer vision, 2003. Proceedings, vol 1, pp 10–17

18. Rottensteiner F (2003) Automatic generation of high-quality building models from lidar data. IEEE Comput Graph Appl 23(6):42–50. doi:10.1109/MCG.2003.1242381
19. Saveriano M, Lee D (2013) Point cloud based dynamical system modulation for reactive avoidance of convex and concave obstacles. In: 2013 IEEE/RSJ international conference on intelligent robots and systems (IROS), pp 5380–5387
20. Sukmanee W, Ruchanurucks M, Rakprayoon P (2012) Obstacle modeling for manipulator using iterative least square (ils) and iterative closest point (icp) base on kinect. In: 2012 IEEE international conference on robotics and biomimetics (ROBIO), pp 672–676
21. Thumbunpeng P, Ruchanurucks M, Khongma A (2013) Surface area calculation using kinect's filtered point cloud with an application of burn care. In: 2013 IEEE international conference on robotics and biomimetics (ROBIO), pp 2166–2169
22. Tseng JL (2009) Shape-sensitive surface reconstruction for low-resolution point-cloud models. In: International conference on computational science and its applications, 2009. ICCSA '09, pp 198–207. doi:10.1109/ICCSA.2009.28
23. Turner E, Cheng P, Zakhor A (2015) Fast, automated, scalable generation of textured 3d models of indoor environments. IEEE J Sel Top Signal Proc 9(3):409–421. doi:10.1109/JSTSP.2014.2381153
24. Um D, Gutierrez M, Bustos P, Kang S (2013) Simultaneous planning and mapping (spam) for a manipulator by best next move in unknown environments. In: 2013 IEEE/RSJ international conference on intelligent robots and systems (IROS), pp 5273–5278
25. Wang B, Yang C, Xie Q (2012) Human-machine interfaces based on emg and kinect applied to teleoperation of a mobile humanoid robot. In: 2012 10th world congress on intelligent control and automation (WCICA), pp 3903–3908
26. Wang X, Yang C, Ma H, Cheng L (2015) Shared control for teleoperation enhanced by autonomous obstacle avoidance of robot manipulator. In: 2015 IEEE/RSJ international conference on intelligent robots and systems (IROS), pp 4575–4580
27. Yaguchi H, Takaoka Y, Yamamoto T, Inaba M (2013) A method of 3d model generation of indoor environment with manhattan world assumption using 3d camera. In: 2013 IEEE/SICE international symposium on system integration (SII), pp 759–765. doi:10.1109/SII.2013.6776686

Xinyu Wang PhD student at the School of Automation, Beijing Institute of Technology. He was working with Dr Chenguang Yang as a visiting student in the UK from 2014 to 2015. He received his bachelor degree from Beijing Institute of Technology in 2011. His research interest covers robotics and autonomous vehicle.

Chenguang Yang Senior Lecturer at College of Engineering, Swansea University. He received his B.Eng. degree in Measurement and Control from Northwestern Polytechnical University, Xi'an, China, in 2005 and his Ph.D. degree in Control Engineering from the National University of Singapore, Singapore, in 2010. His current research interests include robotics, control, and human-robot interaction.

Zhaojie Ju (M'08) received the B.S. in automatic control and the M.S. in intelligent robotics both from Huazhong University of Science and Technology, China, in 2005 and 2007 respectively, and the Ph.D. degree in intelligent robotics at the University of Portsmouth, UK, in 2010. Dr Ju is currently a Senior Lecturer in the School of Computing, University of Portsmouth, UK. He previously held research appointments in the Department of Computer Science, University College London and Intelligent Systems and Biomedical Robotics group, University of Portsmouth, UK. His research interests are in machine intelligence, robot learning and pattern recognition.



Hongbin Ma Professor at the School of Automation, Beijing Institute of Technology. He received his bachelor degree from Zhengzhou University in 2001 and Ph. D. degree from the Academy of Mathematics and Systems Science, Chinese Academy of Sciences in 2006. He has received the 13th Huo Ying Dong Young Teacher Award for Higher Education in 2012. His research interest covers adaptation, learning and recognition, especially adaptive estimation, as well as their applications in robots and autonomous systems.

Mengyin Fu Professor of Cheung Kong Scholars Program, professor at the School of Automation, Beijing Institute of Technology. His research interest covers integrated navigation, intelligent navigation, image processing, learning and recognition as well as their applications.