Robot programming by demonstration: a novel system for robot trajectory programming based on robot operating system

Hong-Da Zhang[1] · Shou-Bin Liu[1] · Qu-Jiang Lei[1] · Yue He[1] · Yang Yang[1] · Yang Bai[1]

Abstract In this article, a new trajectory programming system that allows non-expert users to intuitively and efficiently program trajectories for robots is proposed. The system tracks a pen-shaped marker and obtains its position and orientation by processing the point cloud data of the workspace. A graphical user interface, which enables users to save and execute the acquired trajectory immediately after performing trajectory demonstration, is designed and developed for the system. The performance of the developed system is experimentally evaluated by using it to program trajectories for a UR5 robot. The results indicate that compared with traditional kinesthetic programming, the developed system has the potential of significantly reducing the ergonomic stress and workload of users. The system is developed based on the robot operating system, which facilitates its integration with different robot control systems.

Keywords Programming by demonstration (PbD) · Trajectory programming · Point cloud · Robot operating system (ROS)

1 Introduction

Robots, in general, are complex machines, and professional knowledge and programming skills are necessary to control them [1]. For small and medium-sized enterprises (SMEs), it is considerably difficult to have a fully automated robotic system that can handle all types of tasks, such as part handling, assembly, and welding. In some cases, even if a robotic system has been designed for these applications, frequent reprogramming is required to cope with variations in work contents. This can be performed by hiring a team of experienced robotic engineers because regular factory staffs typically have limited robot programming skills [2]. Robot reprogramming, however, remains difficult, thus limiting the application of robots in SMEs and prevents the further use of robots on a daily basis. In order to simplify the programming of industrial robots and make them more accessible to the general public, robotics researchers have developed a new approach for robot programming. The technique involves observing the tasks demonstrated by humans or other robots—hence the name, robot programming by demonstration (PbD or RPD) [3, 4].

In the traditional RPD system, teach pendants have been adopted to demonstrate the movements that the robot should perform. This technique has been used for industrial manipulators for many years. During the demonstration, the teach pendant is employed to move the robot to desired positions, which are recorded to generate a robot program that will reproduce the demonstrated task [1]. This type of teaching method, however, is neither convenient nor intuitive, and certain programming skills are required to perform a feasible demonstration. Alternatively, kinesthetic teaching (also known as playback programming) provides an accessible method for non-experts to quickly and easily program robots. In the kinesthetic teaching process, a human manipulator guides the robot's joints by hand to achieve a certain pose or movement through a desired trajectory while the robot is placed in compliant mode. This program technique has become the most popular method of robot programming because it is easy to

✉ Qu-Jiang Lei
qj.lei@aaia-ai.org

[1] Guangzhou Institute of Advanced Technology, Chinese Academy of Sciences, Guangzhou 511458, People's Republic of China

comprehend and does not require additional hardware, such as cameras, magnetic sensors, and data gloves. Problems occur, however, when the robot is large; in such a case, a smaller surrogate dummy robot is employed to execute the task [5]. Moreover, during kinesthetic teaching, direct physical contact between the operator and robot is inevitable; this can be inconvenient and even dangerous.

To resolve this problem, a novel robot programming system that enables users to demonstrate, execute, and save robot trajectories without any professional knowledge of programming or even skills of operating robots is proposed. The technique is low-cost, intuitive, efficient, and most importantly, safe. The system is implemented with a robot operating system (ROS) [6], which affords advantages, such as modularity and extensibility. This means that it can be integrated with literally all ROS compatible robot control systems. The remainder of this paper is organized as follows. In Sect. 2, related investigations on designing robot trajectory programming systems are reviewed. In Sect. 3, the proposed system is introduced and explained in detail. To test the system's performance, experiments are conducted and described in Sect. 4, in which the advantages and drawbacks of the system are also discussed. Certain system improvements performed are presented in Sect. 5. Finally, the conclusions drawn and discussions of future research directions are summarized in Sect. 6.

2 Related work

According to Billard et al. [7], current RPD systems can be broadly divided into two categories: trajectory encoding RPD system (a low-level representation of skill in the form of nonlinear mapping between sensory and motor information, where the focus is on learning trajectories from a human demonstrator) and symbolic encoding RPD system (a high-level representation of skills that decomposes it into a sequence of predefined motion elements). As we focus on the former type of RPD systems, related work in the field of robot trajectory programming is reviewed.

As previously mentioned, traditional trajectory programming methods, such as kinesthetic teaching and the use of teach pendants, are inconvenient and time-consuming. Moreover, from the perspective of safety, both techniques pose a substantial risk to the operator. More reliable and practical techniques are therefore urgently required. Landa-Hurtado et al. [8] developed a flexible trajectory generation approach based on Microsoft Kinect sensor to generate new trajectories for an ABB IRB1600ID welding robot. During trajectory programming, the camera detects the skeleton of the operator, and the way-points position information is extracted from the operator's right hand. The robot autonomously generates a trajectory that connects these points, complying with position and orientation constraints. The relative position of the operator's left hand compared to that of the right is also extracted as a means of human robot interaction (HRI) to control the system's running mode. It should be noted, however, that the accuracy of the three-dimensional (3D) position of the operator's skeleton is not appropriate for industrial application. Moreover, the technique of comparing the relative position of the user's two hands as a means of HRI is practically not recommended in industrial scenarios. A similar system is proposed by Moe and Schjølberg [9]. The system developed by the latter is capable of capturing position and orientation information during a trajectory demonstration from Kinect and a smartphone accelerator, respectively. Moreover, it allows real-time interaction with the robot with average response times of 0.675 0 s and 1.155 0 s for position and orientation, respectively. To indicate the start and end of the hand guiding process, the system recognizes a "focus" gesture. Once the gesture is perceived, the system tracks the hand that performed the "click" gesture (the hand is quickly pushed forward and then pulled back), which signals the start of guiding. The end of guiding is completed by clicking using the other hand. During the task demonstration, however, the user must hold a smartphone on the other hand to control the orientation of the robot's end effectors. This makes it cumbersome to some extent and unsuitable for industrial applications. More similar research works related to robot programming through gestures are conducted in Refs. [10–12].

The application of augmented reality (AR) in industrial robot spatial programming is also a popular research trend in the RPD field. Different from conventional forms of visualization, this technique enhances a camera image by adding spatially related information, thereby enabling the user to visualize information of robot programs or intuitively coordinate systems. Task level information, such as poses and trajectories, can also be visualized within the real robot environment. Lambrecht and Krüger [13] presented a spatial programming system for industrial robots that included different modules for gesture-based definition of poses, trajectories, and tasks. Additionally, the system covers program evaluation in an AR application as well as program adaption through spatial interaction with virtual objects representing the robot program. Their system consists of the following basic hardware components: industrial robot (KUKA robot), handheld device (Samsung Galaxy S II smartphone), and motion tracking system (Kinect Sensor). The Kinect sensor tracks the 3D trajectories of the user's hand, and the smartphone is utilized to recognize two-dimensional hand gestures through its camera and display virtual objects and trajectories with its screen. Moreover, for haptic feedback, the phone will vibrate once the user performs a virtual object griping task. Using their developed system, users can easily program and modify robot

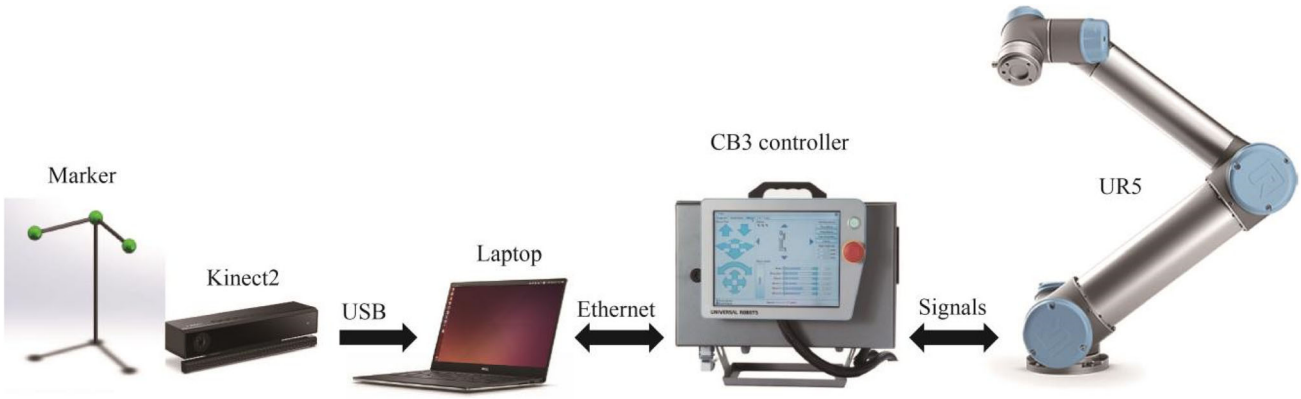


Fig. 1 System hardware setup

trajectories as well as acquire visual feedbacks in the AR simultaneously. The orientation information of the trajectory, however, cannot be acquired during programming, and the accuracy of the 3D trajectory obtained by Kinect is not adequate for industrial application. A similar system related to robot programming in an AR environment is developed by Ng et al. [14]. The system contains a laser range finder, a pan-tilt assembly unit (PTU), and a camera. During programming, the camera captures the workpiece image, and users can define a target position by simple point-and-click on the image. The Cartesian coordinates of the corresponding position is thereafter acquired using a laser range finder and the PTU. A virtual frame and tool will be superimposed at the particular position to provide visual feedback to the user, and the virtual tool's orientation can be modified to a desired state in the AR environment. Apart from generating trajectories by selecting points in the image, their system allows users to generate trajectories through line and curve detection from a selected region of interest. The method can eliminate potential human errors caused by jerks and jitters while performing repetitive mouse-clicking along the joint lines. Their system is capable of achieving planar positioning and vertical liftoff distance accuracies of approximately 0.5 mm and 1 mm, respectively. It should be noted, however, that it is cumbersome for users to specify the orientation of each point in the generated trajectory, thus making it inefficient for robot programming. Works related to AR application in the industrial scenario are well studied in Refs. [15–17].

3 Material and methods

In this section, the developed trajectory teaching system is discussed in detail. Compared with the aforementioned technologies, the proposed system is more economical and efficient. Moreover, a graphical user interface is developed to allow convenient interactions with the system.

3.1 Hardware components

The system hardware components consist of a Microsoft Kinect camera mounted on a shelf above the work platform, an Ubuntu 16.04 laptop with ROS kinetic, a UR5 robot, a CB3 robot controller, and a key element (an inexpensive 3D printed color marker). The marker consists of three 1.5-cm radius green spheres and three 7.5-mm diameter white cylindrical rods; the angle between each rod is 90°. The system hardware setup is shown in Fig. 1.

3.2 Trajectory teaching

To program a trajectory for a robot, the user can accomplish the task by simply moving the color marker along a desired path. Thereafter, the system will automatically track and record both the position and orientation information of its trajectory. This can be immediately executed after the demonstration. The trajectory information can be saved and reloaded in the form of extensible markup language (XML) files that can be further processed (e.g., optimization and generalization).

For the purpose of tracking the trajectory of the color marker, a ROS program is developed to track and publish the marker's real-time position and orientation information. The program employs the point cloud information of the working scenario acquired from the Kinect sensor as input and outputs the current pose of the color marker in the base frame of the robot. The workflow of the program is presented in Fig. 2.

3.2.1 Point cloud segmentation

To determine the position of the color marker, the point cloud of the marker should first be extracted from the data acquired by Kinect. Although a number of complex techniques can be adopted to perform point cloud segmentation, color filtering is employed to accomplish the task. This is because under a homogeneous illumination

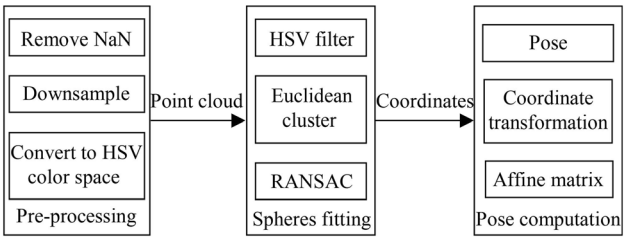


Fig. 2 Workflow of the marker tracker program

condition, the method is robust and computationally efficient [18]. In computer vision and image analysis, the HSV color space is widely employed for feature detection and image segmentation [19]. A colored object in an image can be easily identified by an HSV filter, which can also be applied in point cloud segmentation to extract a specific color component of point cloud data.

The point clouds acquired through Kinect usually contain invalid values because of its range limitation or unusually reflecting surfaces in the platform. These invalid points should therefore be removed before further processing. The color information of the point cloud is usually encoded in the RGB color space. A conversion between the RGB and HSV point clouds should be performed before color filtering. After preprocessing the point cloud, an HSV color filter is applied to separate the marker's point cloud (see Fig. 3). In order to improve system robustness and flexibility, users can specify a color criterion for the filter by selecting a colored point from point clouds on a point cloud viewer. This means that users can choose different marker colors provided it can be easily distinguished in the working platform.

3.2.2 *Point cloud clustering*

After color filtering, the point cloud data contain point clouds of the surfaces of the three colored spheres. To separately acquire point clouds for each sphere, the data should be classified into three different clusters. This task is accomplished through the Euclidean clustering algorithm, which is available in the point cloud library (PCL) [20]. The algorithmic steps are presented as Algorithm 1.

Algorithm 1 Euclidean clustering algorithm

Input a point cloud dataset P;

Create an empty list of clusters (C) and a queue of points to be checked (Q);

For every point $p_i \in P$, perform the following steps:

Add p_i to the current queue, Q;

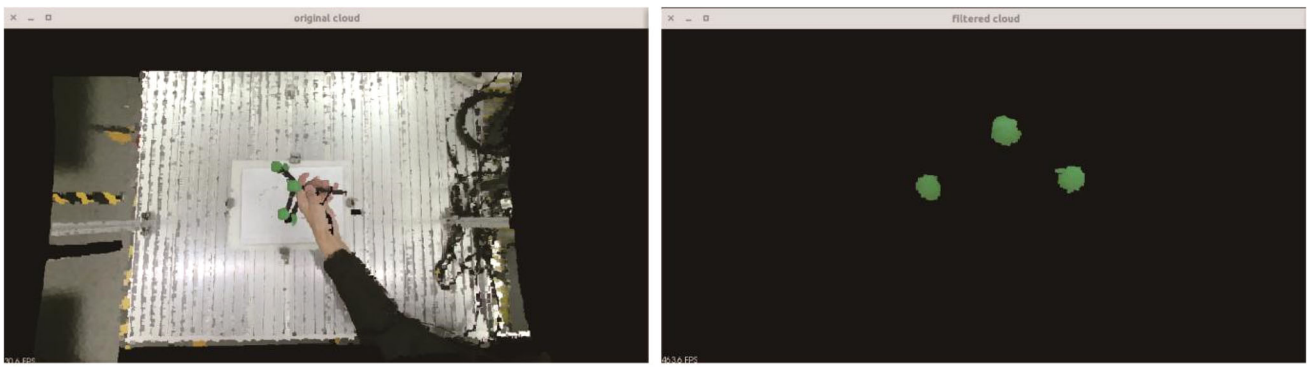
For every point $P_i \in Q$, perform the following:

Search for set P_i^k of point neighbors of P_i inside a sphere with radius $r < d_{\text{th}}$;

For every neighbor $p_i^k \in P_i^k$, check if the point is already inside Q; otherwise, add it to Q;

When no more points can added to Q, add Q to the list of clusters, C, and clear Q;

The algorithm terminates when all points ($p_i \in P$) have been processed and included in C.



(a) Original point cloud **(b)** Segmented point cloud

Fig. 3 Point cloud segmentation

By applying Algorithm 1, the point clouds can be easily separated into three clusters, each of which corresponds to a single sphere (see Fig. 4).

3.2.3 Sphere fitting

In order to track the position and orientation information of the moving marker, a coordinate system is set up according to the distribution of colored spheres. The position of the coordinate is placed at the center of the middle sphere; the pose of the coordinate is illustrated in Fig. 5. In this manner, the pose of the coordinate can be identified with respect to the origins of the three spheres. The point cloud of each sphere has been acquired; hence, the origin of each point cloud can be identified using data-fitting algorithms.

Fitting a model to noisy data (regression analysis) is frequently employed in computer vision for a wide range of objectives, such as reverse engineering. As a traditional technique, the method of least squares has become the most popular algorithm applied in regression analysis. The method can achieve an optimum result when the error distribution of the data is Gaussian. If the noise has non-zero mean components and/or the data contain outliers, however, then the method of least squares may output erroneous results [21].

To find the center of the sphere's point cloud, a robust regression method that remains reliable in the presence of various types of noise should be employed. In computer vision, the least-median-of-squares method and random sample consensus (RANSAC) method are robust algorithms with a certain resistance to outliers [22, 23]. The RANSAC algorithm is applied to our sphere fitting problem to find its origin. The basic algorithm can be summarized as Algorithm 2 [24].

Algorithm 2 RANSAC algorithm

Step 1 Randomly select the minimum number of points required to determine the model parameters.

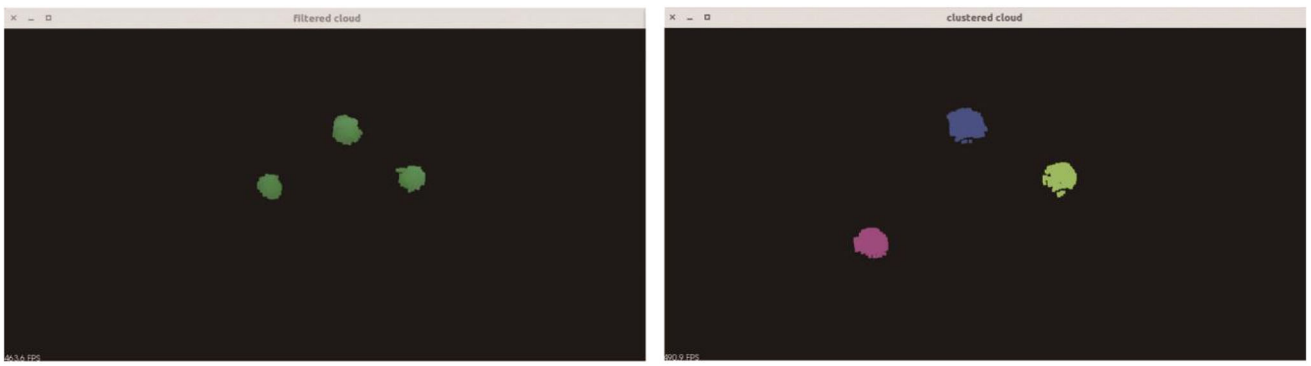
Step 2 Determine the model parameters.

Step 3 Determine the number of points from the set of all points that can fit with a predefined tolerance, ε.

Step 4 If the fraction of the number of inliers over the total number points in the set exceeds a predefined threshold, τ, then re-estimate the model parameters using all the identified inliers and terminate.

Step 5 Otherwise, repeat steps 1–4 (the maximum of N times) [24].

The RANSAC algorithm and several extended algorithms are available within the PCL (e.g., the maximum likelihood estimation sample and consensus, M-estimator sample and consensus and progressive sample and consensus). These algorithms provide fast and robust fitting results for the 3D detection of geometrically simple shapes, such as cylinders, spheres, and planes [25]. By applying the algorithm, the center of the sphere-shaped point cloud can be estimated, and the difference between the actual and resulting sphere radii is used as a criterion to ensure the accuracy of results. The fitting results are displayed in the point cloud viewer with red spheres (see Fig. 6).



(a) Segmented point cloud **(b)** Clustered point cloud

Fig. 4 Point cloud clustering

3.2.4 Pose computation and coordinate transformation

After gathering all three positions of the centers of spheres, the pose of the frame fixed on the marker in the camera frame can be easily identified. Suppose that the coordinates of the origin of the three spheres in the camera frame are P_1 (x_1, y_1, z_1), P_2 (x_2, y_2, z_2) and P_3 (x_3, y_3, z_3). Then, the distances between each point can be expressed as

$$\begin{aligned} &|P_iP_j| = \sqrt{(x_j - x_i) + (y_j - y_i) + (z_j - z_i)}, \\ &i, j \in \{1, 2, 3\}, \quad i \neq j. \end{aligned} \tag{1}$$

Considering the spatial distribution of the colored spheres of the marker, the three points form a right triangle in space, and the vertex of its right angle is the origin of the marker's frame. This can be easily identified by comparing the length of the triangle's edges; the intersection of the legs of the triangle is the origin. The position of the current marker can then be represented as

$${}^{\mathrm{C}}\boldsymbol{t}_{\mathrm{M}} = [{}^{\mathrm{C}}X_{\mathrm{M}}, {}^{\mathrm{C}}Y_{\mathrm{M}}, {}^{\mathrm{C}}Z_{\mathrm{M}}]^{\mathrm{T}}. \tag{2}$$

The direction vector of the shortest leg of the right triangle originating from ${}^{\mathrm{C}}O_{\mathrm{M}}({}^{\mathrm{C}}X_{\mathrm{M}}, {}^{\mathrm{C}}Y_{\mathrm{M}}, {}^{\mathrm{C}}Z_{\mathrm{M}})$ is considered as the positive direction of the x axis. Similarly, the direction of the y axis is defined by the other leg of the triangle. The direction vector of the z axis can be thus be determined as

$${}^{\mathrm{C}}\boldsymbol{z}_{\mathrm{M}} = {}^{\mathrm{C}}\boldsymbol{x}_{\mathrm{M}} \otimes {}^{\mathrm{C}}\boldsymbol{y}_{\mathrm{M}}. \tag{3}$$

The orientation of the marker with respect to the camera can therefore be described as

$${}^{\mathrm{C}}_{\mathrm{M}}\boldsymbol{R} = [{}^{\mathrm{C}}\boldsymbol{x}_{\mathrm{M}}, {}^{\mathrm{C}}\boldsymbol{y}_{\mathrm{M}}, {}^{\mathrm{C}}\boldsymbol{z}_{\mathrm{M}}], \tag{4}$$

where $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{z}$ are the unit vectors; ${}^{\mathrm{C}}_{\mathrm{M}}\boldsymbol{R}$ is an orthogonal matrix. The pose of the marker in the camera's coordinate system can be represented by a homogeneous transformation matrix

$${}^{\mathrm{C}}_{\mathrm{M}}\boldsymbol{H} = \begin{bmatrix} {}^{\mathrm{C}}_{\mathrm{M}}\boldsymbol{R} & {}^{\mathrm{C}}\boldsymbol{t}_{\mathrm{M}} \\ \boldsymbol{0}_{3\times1} & \boldsymbol{1} \end{bmatrix}. \tag{5}$$

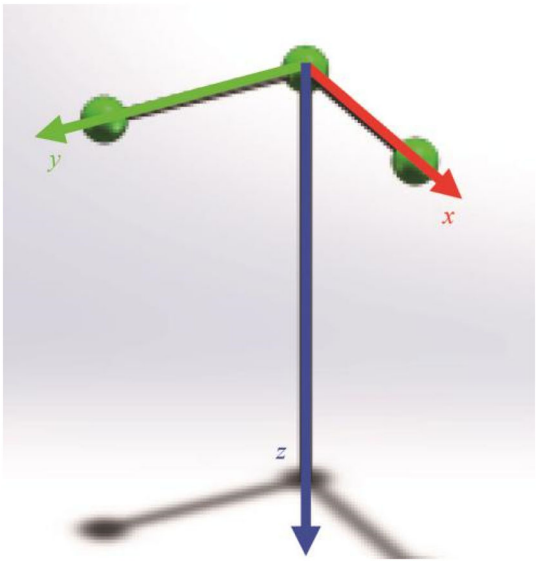


Fig. 5 Marker's frame

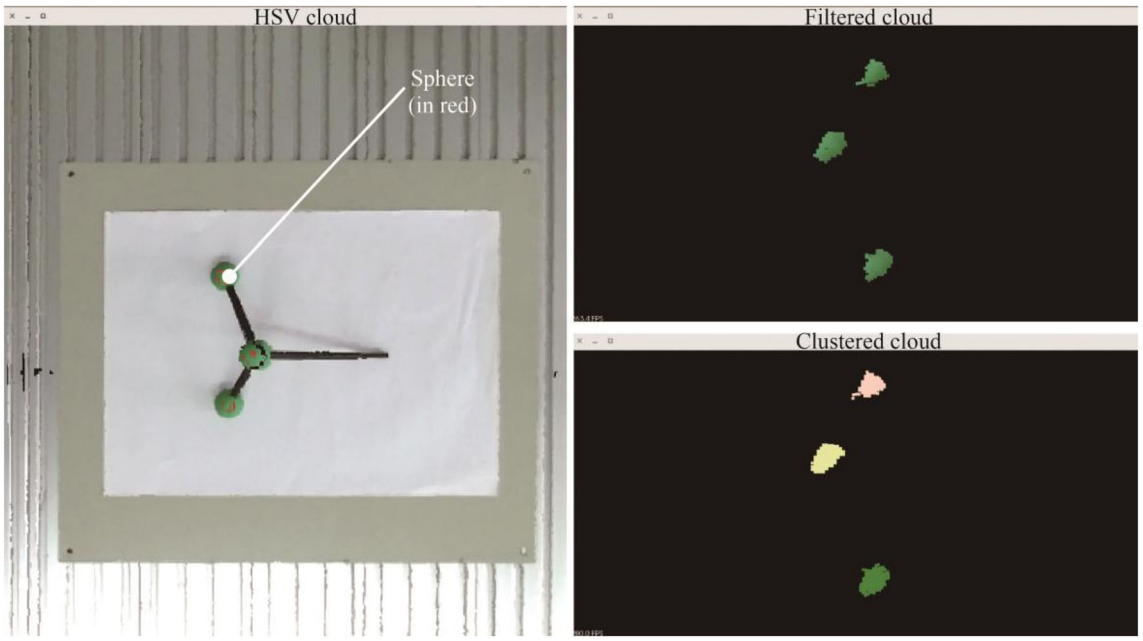


Fig. 6 Sphere fitting result

Figure 7 shows the pose of the marker in the camera frame in the point cloud viewer. After the camera calibration, the transformation from the camera coordinates to the robot's base frame can be described by a transformation matrix, ${}^{\mathrm{B}}_{\mathrm{C}}\boldsymbol{H}$. The pose of the marker in the robot's coordinate system can then be expressed by a homogeneous transformation matrix

$$ {}^{\mathrm{B}}_{\mathrm{M}}\boldsymbol{H} = {}^{\mathrm{B}}_{\mathrm{C}}\boldsymbol{H}\,{}^{\mathrm{C}}_{\mathrm{M}}\boldsymbol{H} = \begin{bmatrix} {}^{\mathrm{B}}_{\mathrm{M}}\boldsymbol{R} & {}^{\mathrm{B}}\boldsymbol{t}_{\mathrm{M}} \\ \mathbf{0}_{3\times 1} & \mathbf{1} \end{bmatrix}, \tag{6} $$

where ${}^{\mathrm{B}}\boldsymbol{t}_{\mathrm{M}} = [{}^{\mathrm{B}}X_{\mathrm{M}}, {}^{\mathrm{B}}Y_{\mathrm{M}}, {}^{\mathrm{B}}Z_{\mathrm{M}}]^{\mathrm{T}}$, ${}^{\mathrm{B}}_{\mathrm{M}}\boldsymbol{R} = [{}^{\mathrm{B}}\boldsymbol{x}_{\mathrm{M}}, {}^{\mathrm{B}}\boldsymbol{y}_{\mathrm{M}}, {}^{\mathrm{B}}\boldsymbol{z}_{\mathrm{M}}]$.

The position of the origin of the marker's frame in the robot's coordinate system, ${}^{\mathrm{B}}O_{\mathrm{M}}({}^{\mathrm{B}}X_{\mathrm{M}}, {}^{\mathrm{B}}Y_{\mathrm{M}}, {}^{\mathrm{B}}Z_{\mathrm{M}})$, can thus be determined, and the orientation can be identified by the rotation matrix, ${}^{\mathrm{B}}_{\mathrm{M}}\boldsymbol{R}$.

After obtaining the pose of the marker in the robot's frame, the position and orientation information of the marker's tip can be further obtained. A coordinate system named "tool frame" is fixed on the tip; it has the same orientation as the marker's frame and a translation along the z axis. Suppose that the length of the pole below the middle sphere is l mm. Then, the homogeneous transformation matrix from the tool frame to the marker's frame, ${}^{\mathrm{M}}_{\mathrm{T}}\boldsymbol{H}$, can be denoted as

$$ {}^{\mathrm{M}}_{\mathrm{T}}\boldsymbol{H} = \begin{bmatrix} {}^{\mathrm{M}}_{\mathrm{T}}\boldsymbol{H} & {}^{\mathrm{M}}\boldsymbol{t}_{\mathrm{T}} \\ \mathbf{0}_{3\times 1} & \mathbf{1} \end{bmatrix} = \begin{bmatrix} \mathbf{I}_{3\times 3} & [0,0,l]^{\mathrm{T}} \\ \mathbf{0}_{3\times 1} & \mathbf{1} \end{bmatrix}. \tag{7} $$

The pose of the tool frame in the robot's frame ${}^{\mathrm{B}}_{\mathrm{T}}\boldsymbol{H}$ can be expressed as

$$ {}^{\mathrm{B}}_{\mathrm{T}}\boldsymbol{H} = {}^{\mathrm{B}}_{\mathrm{M}}\boldsymbol{H}\,{}^{\mathrm{M}}_{\mathrm{T}}\boldsymbol{H} = \begin{bmatrix} {}^{\mathrm{B}}_{\mathrm{T}}\boldsymbol{R} & {}^{\mathrm{B}}\boldsymbol{t}_{\mathrm{T}} \\ \mathbf{0}_{3\times 1} & \mathbf{1} \end{bmatrix}, \tag{8} $$

where ${}^{\mathrm{B}}\boldsymbol{t}_{\mathrm{T}} = [{}^{\mathrm{B}}X_{\mathrm{T}}, {}^{\mathrm{B}}Y_{\mathrm{T}}, {}^{\mathrm{B}}Z_{\mathrm{T}}]^{\mathrm{T}}$ and ${}^{\mathrm{B}}_{\mathrm{T}}\boldsymbol{R} = [{}^{\mathrm{B}}\boldsymbol{X}_{\mathrm{T}}, {}^{\mathrm{B}}\boldsymbol{Y}_{\mathrm{T}}, {}^{\mathrm{B}}\boldsymbol{Z}_{\mathrm{T}}]$.

Coordinate ${}^{\mathrm{B}}O_{\mathrm{T}}({}^{\mathrm{B}}X_{\mathrm{T}}, {}^{\mathrm{B}}Y_{\mathrm{T}}, {}^{\mathrm{B}}Z_{\mathrm{T}})$ is the position of the origin of the tool frame in the robot's coordinate system, and its orientation can be represented by the rotation matrix, ${}^{\mathrm{B}}_{\mathrm{T}}\boldsymbol{R}$. The relationship of the coordinates is presented in Fig. 8.

3.3 Graphical user interface

A graphical user interface (GUI) based on the Qt and ROS that allows users to control the system's running mode and drive the robot to execute the recorded trajectories is developed (see Fig. 9) [26]. The interface is mainly composed of two panels: information and command panels. The information panel consists of two tags, each of which is responsible for displaying different information. The "Robot Info" tag is used to present the robot information, including the position and orientation of its end effector and joint angles. This tag also contains several buttons that allow users to move the robot to predefined positions, and operating information related to trajectory planning and execution is also displayed in the "Info" window. The

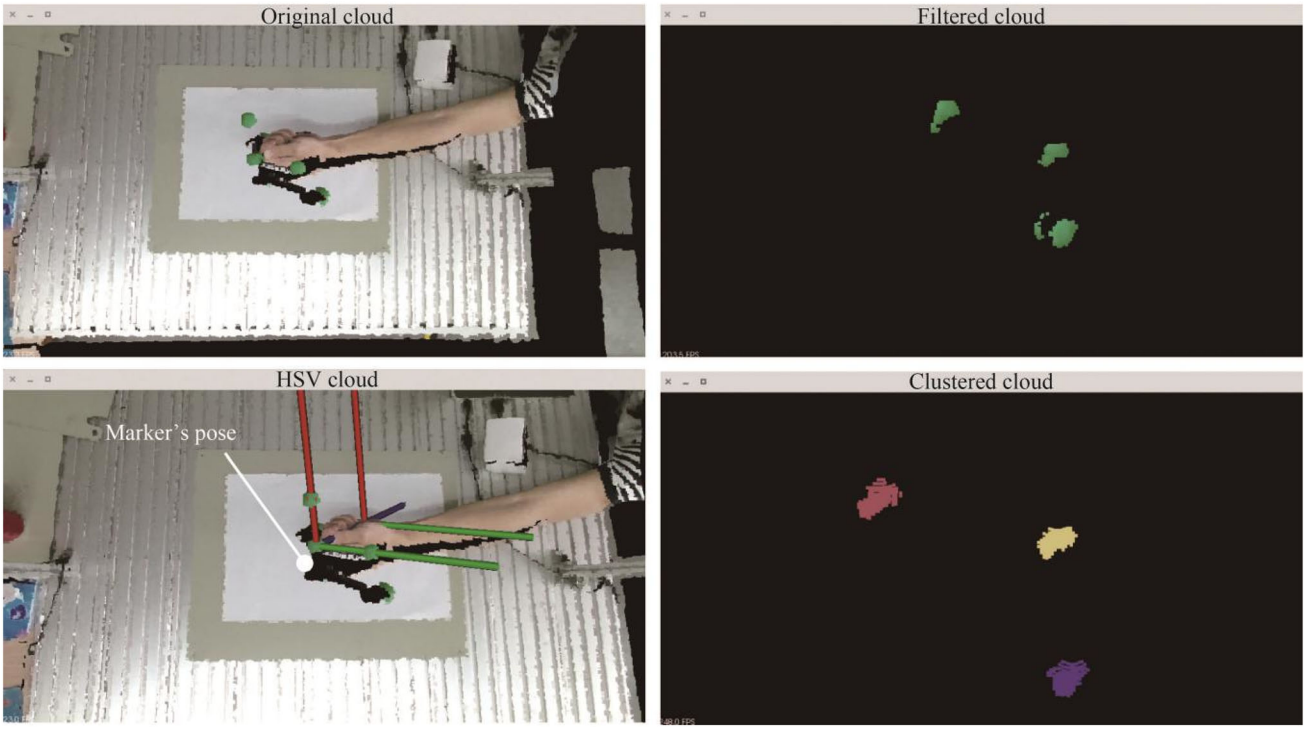


Fig. 7 Marker's pose in camera frame

"Scene Info" tag is mainly responsible for displaying the pose information of the marker and operating information of trajectory tracking, recording, etc.

The command panel on the right is the system's control center, which consists of five different sections. From top to bottom, the first region contains certain options related to the connection with the ROS master, which is the communication center of the ROS system. Users are allowed to set the URL and IP for the ROS master or choose to employ environment variables. If the check box "Remember settings on startup" is marked, the interface will store the current settings and reload it during the next startup.

Below the "ROS Master" section is the "RPD Control" panel, which allows users to control the trajectory demonstration process. Users can specify a time duration for the trajectory teaching procedure, which will be automatically terminated at the end of the countdown. Manually stopping the recording process is also supported. After acquiring the trajectory information of the marker, users should click the "Finish" button to indicate the termination of the teaching process and enable the system to perform trajectory manipulations (e.g., planning and saving).

The "Plan and Execute" panel contains functions related to trajectory planning and execution. Users can immediately execute the recorded trajectories by clicking the "Plan" and "Execute" buttons after the trajectory demonstration process. This is accomplished through a motion planning library (i.e., "MoveIt"), which can generate feasible robot trajectories according to the recorded trajectory information [27].

The "Save and Load" panel mainly includes file manipulation functions, allowing users to save and load the recorded trajectory information in the form of XML files (see Figs. 10 and 11). The "Scene Objects" panel is composed of four buttons with different functions. It provides users with the ability to add collision objects to the planning space of MoveIt and thus eliminate the danger of collision with obstacles during trajectory execution.

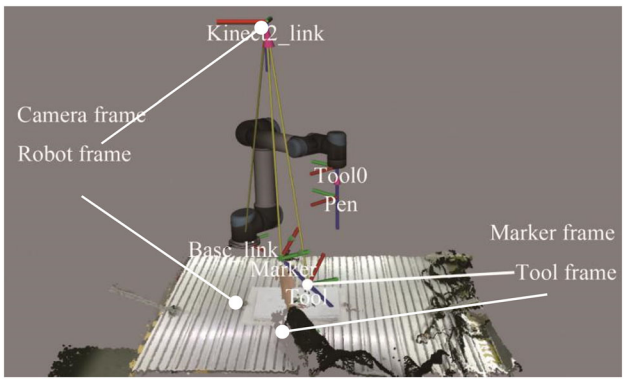


Fig. 8 Frames displayed in rviz

During the trajectory demonstration, the recorded trajectory information can be intuitively visualized through rviz, which is a 3D visualizer for displaying sensor data and state information from ROS [28]. Users can visualize and monitor the current sensor information acquired from Kinect and working scene. The coordinate system and resulting robot trajectories can also be observed conveniently.

4 Experiment and discussion

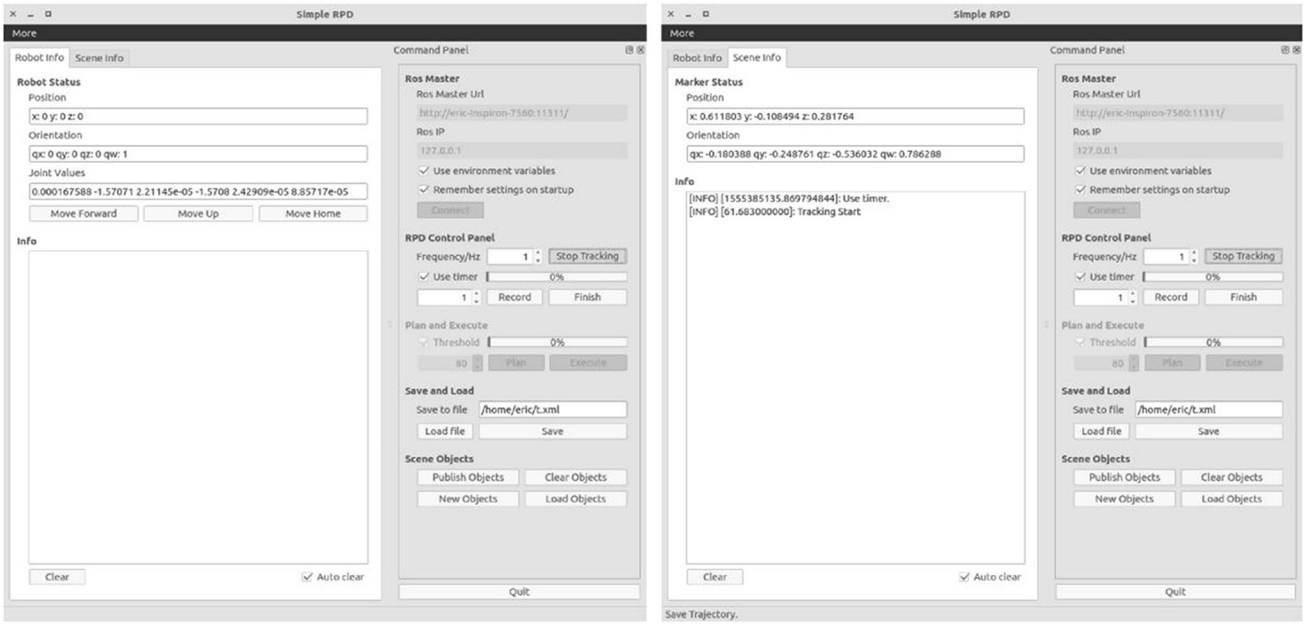
To test the performance of the developed system, a validation experiment, which assesses the accuracy and efficiency of the system compared with the traditional approach of trajectory programming (i.e., kinesthetic programming), is conducted. Five subjects are invited to program and execute a simple trajectory that contains five waypoints for the UR5 robot with the developed system. The same trajectory is programmed by employing UR5's kinesthetic programming interface.

The required time of both approaches is recorded, and execution results are also saved by mounting a pen on the robot's end effector. A piece of paper, which contains nine circles with the same radii (0.75 mm) scattered in the middle of the edges or at the corners of a rectangle (150 mm $\times$ 200 mm), is placed on the working table. Each circle on the test paper represents a possible waypoint of the trajectory. The experimental setup is displayed in Fig. 12.

During the experiment, five subjects are asked to program a desired trajectory passing through five different waypoints on the paper. These subjects have no expertise in robot programming and are taught to operate our system and UR5's kinesthetic interface to program robot trajectories before the experiment. After the demonstration, the maximum deviation of the executed trajectory compared with the ideal one is measured and recorded for analysis (see Fig. 13).

The experimental results are summarized in the Table 1. Evidently, the developed trajectory programming by demonstration system is more efficient than kinesthetic programming. The average times to complete the programming task by employing the proposed developed system and kinesthetic programming are 30.46 s and 109.35 s, respectively, indicating that the efficiency of the proposed trajectory programming system is 3.5 times that of kinesthetic teaching.

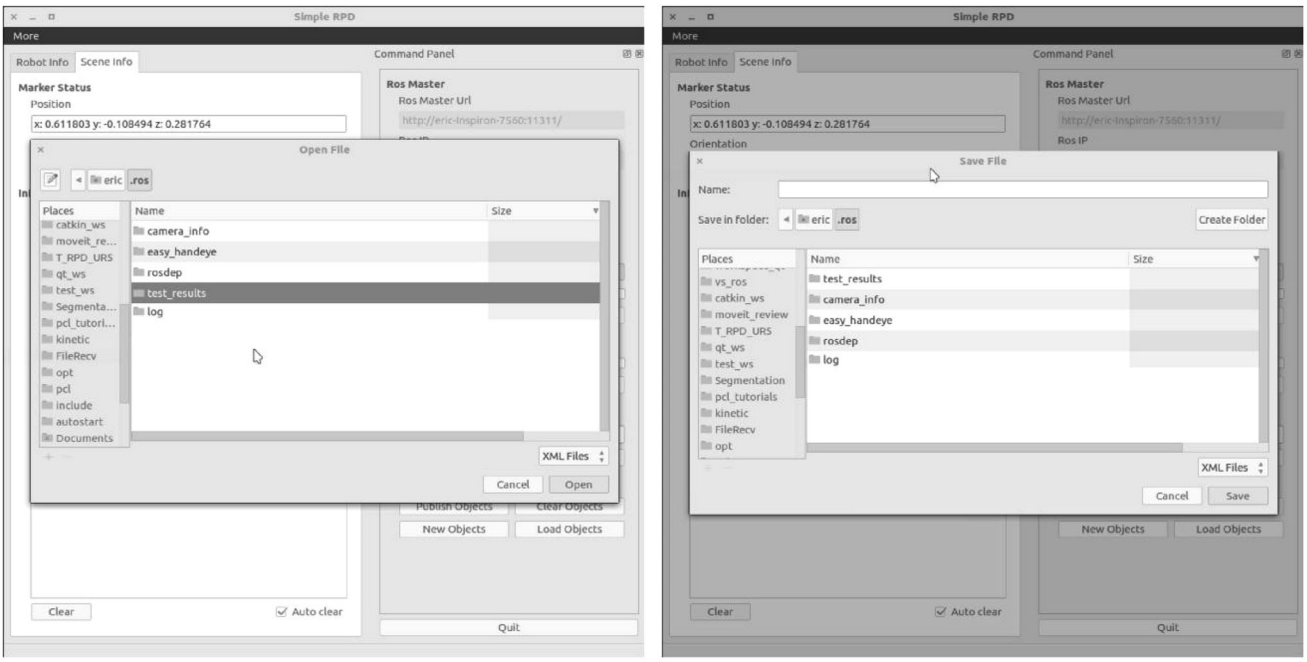
It should be mentioned that the subjects involved in the experiment, especially female subjects, find it laborious to move the robot's arm to desired positions and control its



(a) Robot info tag

(b) Scene info tag

Fig. 9 GUI based on Qt



(a) Load file

(b) Save file

Fig. 10 GUI for trajectory saving and loading operations

orientation. They all claim that compared with the kinesthetic programming interface, the proposed system is easier to learn and more convenient to operate. In addition, the proposed system provides users with visual information of the recorded trajectory in real time, making the programming process more natural and intuitive (see Fig. 14).

The deviation of the executed trajectory of the proposed system, however, is not as good as that of kinesthetic teaching, and certain factors affect its accuracy. For

```
vim test_trajectory.xml
File Edit View Search Terminal Help
<?xml version="1.0" encoding="UTF-8"?>
<trajectory>
    <point>
        <position>
            <x>0.626423</x>
            <y>0.133727</y>
            <z>0.165477</z>
        </position>
        <orientation>
            <qw>-0.012652</qw>
            <qx>0.431545</qx>
            <qy>0.899083</qy>
            <qz>0.0725147</qz>
        </orientation>
    </point>
    <point>
        <position>
            <x>0.777504</x>
            <y>-0.0243896</y>
            <z>0.155479</z>
        </position>
        <orientation>
            <qw>0.0173936</qw>
"test_trajectory.xml" 68L, 1644C                                  1,1           Top
```

Fig. 11 Saved trajectory

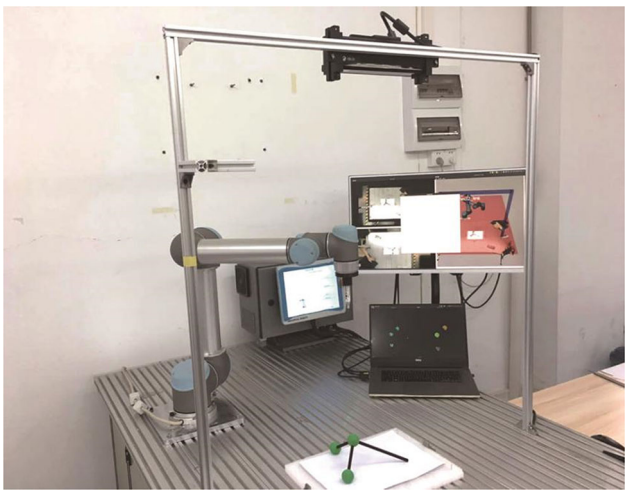


Fig. 12 Hardware setup for the validation experiment

example, the error in the extrinsic parameters of the camera after calibration can affect the accuracy of the tracking results in the robot's frame. Inadequate intrinsic calibration results and the random error of depth measurements of the Kinect sensor can also cause errors in computing the position of the colored sphere in the camera frame, thus resulting in trajectory deviation [29].

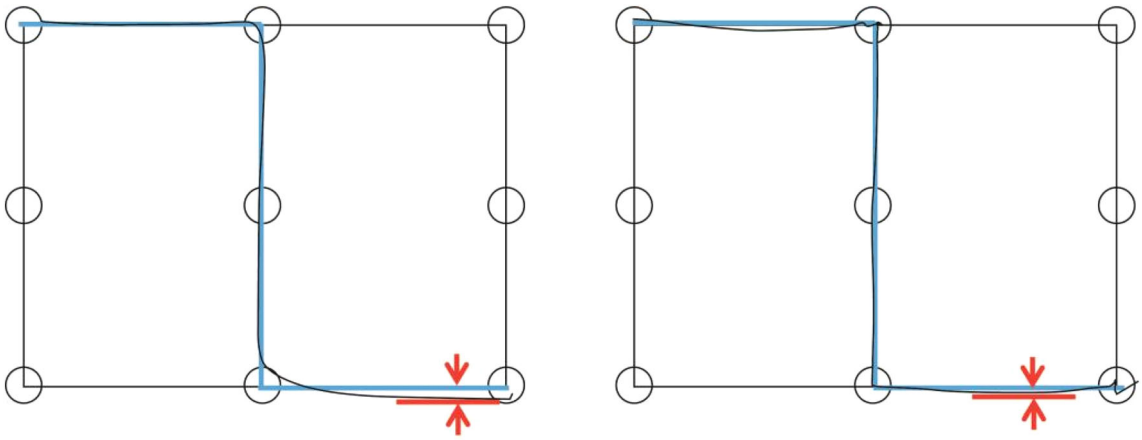
5 Further developments

In the experiments, it has been noticed that during task demonstration, it is difficult to signal the system that demonstration has begun or completed using input devices (e.g., mouse or keyboard). To further improve system friendliness, a hand gesture recognition (HGR) module (see Fig. 15) is developed based on a convolutional neural network (CNN), whose structure is designed based on a classic neural network model, LetNet-5. The HGR module is capable of recognizing seven different hand gesture commands corresponding to the functional buttons on the graphical interface. Videos on performing robot trajectory demonstration using the hand gesture recognition interface is uploaded to https://www.youtube.com/watch?v=zrDPwKBzDR4.

Some advanced functions have also been developed towards planning scene collision object manipulation. By using the system, users can simply draw a collision object with standard geometrical shapes using the pen-shaped marker and adjust its position and orientation in the planning scene. This can provide a quick approach of setting up the robot's working environment for users who are unfamiliar with ROS and MoveIt. A module that provides users with some simple task-based robot programming by demonstration options, such as a pick-and-place task, is developed (see Fig. 16). The corresponding video is uploaded to https://www.youtube.com/watch?v=maANrNGWW3w.

6 Conclusions and future work

In this paper, a low-cost and easy-to-use robot trajectory programming by the demonstration method is presented. The system allows users who have no experience in robot programming to efficiently and intuitively program robot trajectories with the aid of a pen-shaped marker and Kinect



(a) A result trajectory of the system

(b) A result trajectory of kinesthetic teaching

Fig. 13 Maximum deviation between ideal trajectories and actual ones

Table 1 Result of the validation experiment

Subject	Time/s		Error/mm	
	RPD	Kinesthetic	RPD	Kinesthetic
1	33.58	137.89	9.00	2.50
2	31.89	78.28	5.00	3.00
3	28.23	90.30	7.50	3.20
4	24.86	139.44	5.10	3.00
5	33.72	100.84	7.50	3.00
Average	30.46	109.35	6.82	2.94

sensor. A graphical user interface, which allows users to conveniently control the system's running states, is developed. The system also provides intuitive visual feedback that allows users to visualize the demonstrated trajectory dynamically. The system is developed based on the ROS; thus, it can be integrated with different types of robots provided they are ROS compatible. A comprehensive instruction that guides the implementation and operation of the developed system can be found online (https://github.com/Zhang-Hongda). Users can download the ROS packages and build the system on their local machines according to the guide.

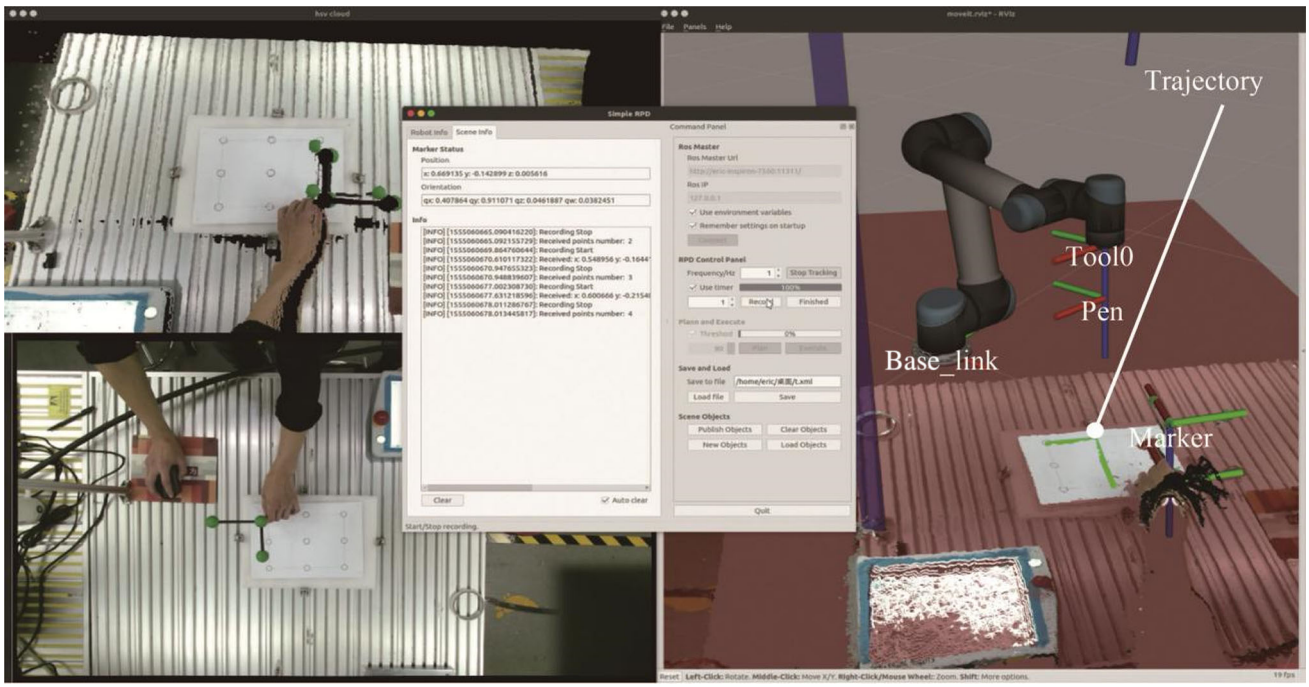
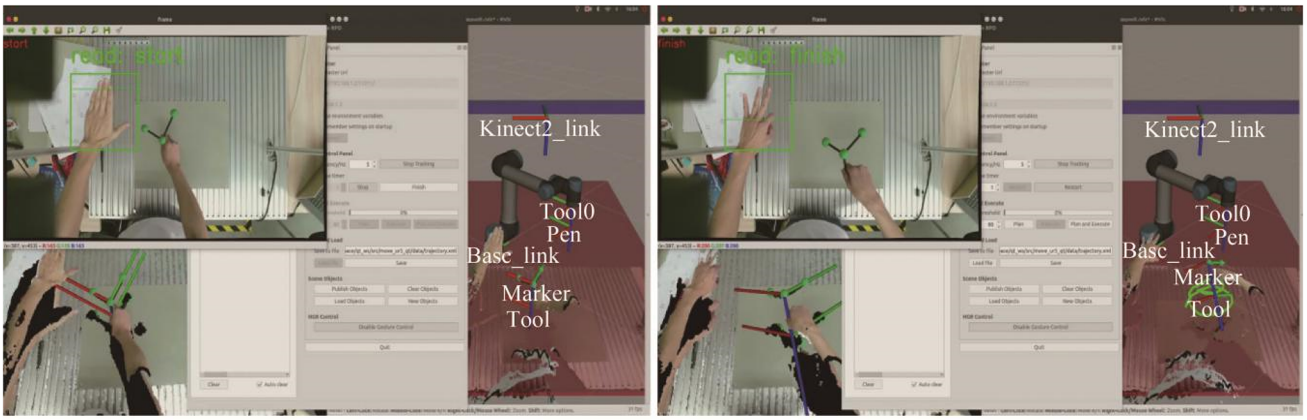
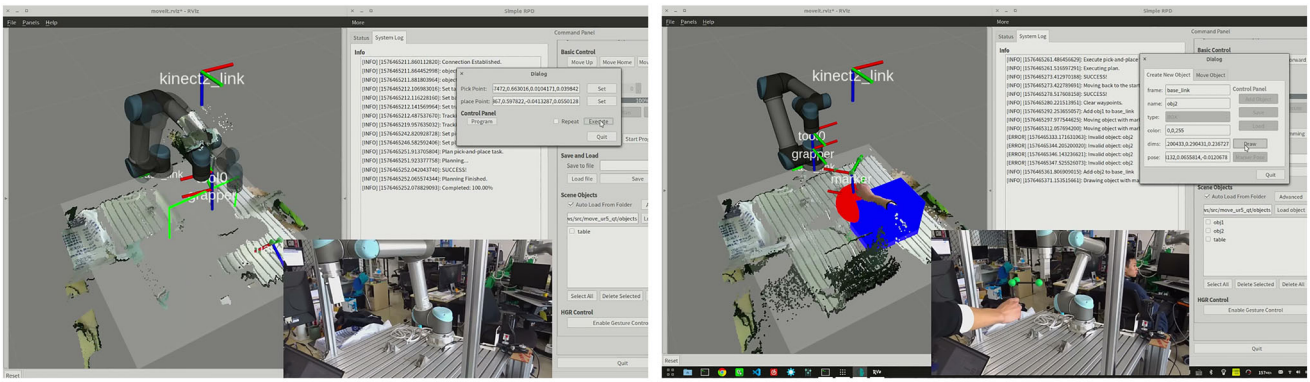


Fig. 14 Real-time display of demonstrated trajectories during programming



(a) **(b)**

Fig. 15 Trajectory demonstration using hand gestures



(a) Programming pick and place task **(b)** Drawing a collision object in the workspace

Fig. 16 Collision objects editing and task-based programming

A set of experiments is conducted using a UR5 robot to evaluate the system performance. The experimental results indicate that our system can boot the speed of robot trajectory programming by approximately 3.5 times faster than the traditional kinesthetic programming method. The results indicate that the developed system can potentially reduce the ergonomic stress and workload of users. The accuracy of the system can be further improved by employing high-precision sensors and more accurate camera calibration. Videos of system implementation and operation are available at https://www.youtube.com/watch?v=F18YgKlDluw, which also includes some of the experimental processes and results.

Further developments have been performed to improve the system, including a hand gesture recognition module and a scene object manipulation module. Users can intuitively control the system through hand gestures and edit or create new collision objects in MoveIt's planning scene with the pen-shaped marker. A task-based trajectory programming module is still under development. Currently, the module supports only the pick-and-place task. To program a pick-and-place task, users should only select pick and place points using the pen-shaped marker. Thereafter, the system will automatically generate a trajectory according to initial points.

In the future, the authors intend provide users with more powerful functions, such as the dynamic editing of the trajectory's waypoints, and modify their speed and acceleration. More programming tasks will be supported by the task-based trajectory programming module, whose performance will be evaluated in a specific working scenario. Additionally, advanced modules, such as object recognition and tracking, will be developed to further process and generalize the demonstrated trajectories to cope with dynamic working scenarios.

Acknowledgments This research was supported by the Major Projects of Guangzhou City of China (Grant Nos. 201907010012, 201704030091 and 201607010041), the Guangdong Innovative and Entrepreneurial Research Team Program (Grant No. 2014ZT05G132), Shenzhen Peacock Plan (Grant No.

KQTD2015033117354154), the Major Projects of Guangdong Province of China (Grant No. 2015B010919002), the Major Projects of Dongguan City of China (Grant No. 2017215102008), and the Nansha District International Science and Technology Cooperation Project of Guangzhou City of China (Grant No. 2016GJ004).

References

1. Biggs G, MacDonald B (2003) A survey of robot programming systems. In: Proceedings of the Australasian conference on robotics and automation, Brisbane, Australia, pp 1–3
2. Ng CL, Ng TC, Nguyen TAN et al (2010) Intuitive robot tool path teaching using laser and camera in augmented reality environment. In: 2010 IEEE 11th international conference on control automation robotics and vision, Singapore, pp 114–119
3. Schaal S, Ijspeert A, Billard A (2003) Computational approaches to motor learning by imitation. Philos Trans R Soc Lond Ser B Biol Sci 358(1431):537–547
4. BillardAG Calinon S, Dillmann R (2016) Learning from humans. In: Siciliano Bruno, Khatib Oussama (eds) Springer handbook of robotics. Springer, Cham, pp 1995–2014
5. Helander MG, Landauer TK, Prabhu PV (2014) Handbook of human-computer interaction, 2nd edn. Elsevier, Amsterdam
6. ROS.org (2018) Powering the world's robots. http://www.ros.org. Accessed 15 May 2019
7. Billard A, Calinon S, Dillmann R et al (2008) Robot programming by demonstration. Springer handbook of robotics. Springer, Berlin, pp 1371–1394
8. Landa-Hurtado LR, Mamani-Macaya FA, Fuentes-Maya M et al (2014) Kinect-based trajectory teaching for industrial robots. In: Pan-American congress of applied mechanics (PACAM), Santiago, Chile
9. Moe S, Schjølberg I (2013) Real-time hand guiding of industrial manipulator in 5 d of using microsoft kinect and accelerometer. In: 2013 IEEE RO-MAN, Gyeongju, Korea, pp 644–649
10. Lambrecht J, KleinsorgeM, KrügerJ (2011) Markerless gesture-based motion control and programming of industrial robots. In: International conference on emerging technologies and factory automation, Toulouse, France, pp 1–4
11. Zhang X, Zhou H, Cheng H et al (2015) Teaching-playback of robot manipulator based on human gesture recognition and motion tracking. In: 2015 IEEE international conference on robotics and biomimetics (ROBIO), Zhuhai, China, pp 1183–1188
12. Lambrecht J, Walzel H, Krüger J (2013) Robust finger gesture recognition on handheld devices for spatial programming of industrial robots. In: IEEE international workshop on robot and human communication, Gyeongju, South Korea, pp 99–106
13. Lambrecht J, Krüger J (2012) Spatial programming for industrial robots based on gestures and augmented reality. In: IEEE/RSJ international conference on intelligent robots and systems, Vilamoura, Portuga, pp 466–472
14. Ng CL, Ng TC, Nguyen TAN et al (2010) Intuitive robot tool path teaching using laser and camera in augmented reality environment. In: 11th International conference on control automation robotics and vision, Singapore, pp 114–119
15. Ong SK, Yuan ML, Nee AYC (2008) Augmented reality applications in manufacturing: a survey. Int J Prod Res 46(10):2707–2742
16. Pettersen T, Pretlove J, Skourup C et al (2003) Augmented reality for programming industrial robots. In: The second IEEE and ACM international symposium on mixed and augmented reality, Tokyo, Japan, pp 319–320
17. Pan Z, Polden J, Larkin N et al (2010) Recent progress on programming methods for industrial robots. Robot Comput Integr Manuf 28(2):87–94
18. Nguyen A, Le B (2013) 3D point cloud segmentation: a survey. In: 2013 6th IEEE conference on robotics, automation and mechatronics (RAM), Manila, Philippines, pp 225–230
19. Cheng HD, Jiang XH, Sun Y et al (2001) Color image segmentation: advances and prospects. Pattern Recognit 34(12):2259–2281
20. PCL - Point Cloud Library (PCL) (2019). http://pointclouds.org. Accessed 18 May 2019
21. Meer P, Mintz D, Rosenfeld A et al (1991) Robust regression methods for computer vision: a review. Int J Comput Vis 6(1):59–70
22. Rousseeuw PJ (1984) Least median of squares regression. J Am Stat Assoc 79(388):871–880
23. Bolles RC, Fischler MA (1981) A RANSAC-based approach to model fitting and its application to finding cylinders in range data. In: International joint conference on artificial intelligence, Vancouver, Canada, pp 637–643
24. Derpanis KG (2010) Overview of the RANSAC algorithm. Image Rochester NY 4(1):2–3
25. Grilli E, Menna F, Remondino F (2017) A review of point clouds segmentation and classification algorithms. In: International archives of photogrammetry, remote sensing and spatial information sciences, Nafplio, Greece
26. Qt (2019) Cross-platform software development for embedded & desktop. https://www.qt.io. Accessed 19 May 2019
27. MoveIt! Motion Planning Framework (2018). http://moveit.ros.org. Accessed 19 May 2019
28. rviz - ROS Wiki (2019). http://wiki.ros.org/rviz. Accessed 20 May 2019
29. Khoshelham K (2011) Accuracy analysis of kinect depth data. In: Proceedings of the ISPRS workshop laser scanning, Calgary, Canada, pp 133–138

Hong-Da Zhang is a guest master student at Guangzhou Institute of Advanced Technology (GIAT), Chinese Academy of Sciences. He is a graduate student in the school of mechanical engineering and automation at Harbin Institute of Technology. His research interests include intelligent robots, robot vision and human-robot interaction.

Shou-Bin Liu works as an associate professor at the school of Mechanical Engineering and Automation at Harbin Institute of Technology, Shenzhen. He received his Doctoral Degree in 2000 from City University of Hong Kong. His research interests include Optomechatronics machine, Precision instrument and intelligent robots.

Qu-Jiang Lei works as an Associate Professor at the Intelligent Robot and Equipment Center, Guangzhou Institute of Advanced Technology (GIAT), Chinese Academy of Sciences, Guangzhou, China. He received the Doctoral Degree in 2018 from Delft University of Technology, Delft, The Netherlands. His research interests include intelligent robots, collaborative robots, robotic systems integration, robot vision, artificial intelligence, machine learning and human-computer interaction.

Yue He is a guest master student at Guangzhou Institute of Advanced Technology (GIAT), Chinese Academy of Sciences. She received the B.S. degree in Communication Engineering from University of Northeast Electric Power University, China, in 2013. She is currently working toward the M.S. degree with the Institute of Communication Engineering, Department of Big Data and Information Engineering, Guizhou University, China. Her latest research interests include robotics vision and machine learning.

Yang Yang is a guest master student at Guangzhou Institute of Advanced Technology (GIAT), Chinese Academy of Sciences. She is a graduate student in the college of Big Data and Information Engineering at Guizhou University. Her research interests include computer vision and image processing.

Yang Bai is a guest master student at Guangzhou Institute of Advanced Technology (GIAT), Chinese Academy of Sciences. He is a graduate student and currently pursuing the M.E. degree in Intelligent Building at Xi'an University of Architecture and Technology, Xi'an, China. His research interests include motion recognition, human robot interaction, collision detection and obstacle avoidance.