Implementation of Robot System for Multi-user and Multi-robot Based on Cloud Platform

Lang Yin¹, *, Guanghui Yan¹, *, Yingjian Cao², b, Beng Liu³, c, Jianyong Feng³, d, Meng Zhang³, e and Ting Liu¹, f

¹School of Electronic and Information Engineering, Lanzhou Jiaotong University, Lanzhou, China
²College of Mathematics and Science, Beijing University of Civil Engineering and Architecture, Beijing, China
³College of Electronic and Information Engineering Shandong University of Science and Technology Qingdao, China

*Corresponding author e-mail: yanguanghui@mail.lzjt, a1198414516@qq.com, b1181127937@qq.com, c1454981217@qq.com, d903885274@qq.com, e736336430@qq.com, f844256497@qq.com

Abstract. With the rapid development of the complexity and diversification of robot services, large number of data processing and analysis problems occurred during the robot working process. In this paper, we designed a collaborative system consisted multi-user and multi-robot based on cloud platform. By pre-setting the rights and restrictions owned by each user in different scenarios, the server determines the scene in which the robot is located by the picture taken by the robot, thereby giving each user different permissions. Enable the robot to listen to the user's instructions to protect privacy. The integration of cloud technology and multi-robot systems made it possible to improve energy efficiency, real-time performance and reduce memory cost.

1. Introduction

Professor James Kuffner of Carnegie Mellon University presented the concept of cloud robots at the Humanoids International Conference in 2010. To date, the vast majority of academic and industrial efforts have tackled these challenges by focusing on increase - ing the performance and functionality of isolated robot sys - tems. However, in a trend mirroring the developments of the personal computing (PC) industry [2], recent years have seen first successful examples of augmenting the computational power of individual robot systems with the shared memory of multiple robots. In an industrial context, Kiva Systems successfully uses systematic knowledge sharing among 1,000 individual robots to create a shared world model that allows autonomous navigation and rapid deployment in semistruc - tured environments with high reliability despite economic constraints [3, 4]. Other examples for shared world models include research on multiagent systems, such as RoboCup [5], where sharing sensor information has been shown to increase the success rate of tracking dynamic objects [6], col - lective mapping of autonomous vehicles [7]. As one of the current hotspots, cloud robot [1] can share the data between robots through the Internet and work together to solve the limitations of single robot self-learning. In order to facilitate the task of arranging tasks for the robot,
this article adds the Android side to the cloud robot system [2] when viewing the robot to complete the task anytime and anywhere. In the scenario of multi-user and multi-robot, there are two problems: First, different users have different permissions in different scenarios of the robot; second, when the robot performs the same task of the user intermittently, the other user has no right to control the robot.

2. System analysis and design

2.1. System Architecture

The system needs the user commander publish the task to the server through the Android terminal, the server converts the task into the execution action of the robot, and transmits the command to the corresponding robot, and the robot sends the corresponding feedback result after completing the corresponding task, and finally displays the result to the Android. The end is for the user to view. Based on the above design, the system framework structure is planned as shown in Figure.1.

![Figure 1. Multi-user multi-robot control system architecture.](image)

2.2. Responsibilities of each module

The responsibilities of each module in the system are as follows:

Android: Provide users with the most direct interactive service. The robot state is acquired from the server, the task state is executed, and various data fed back by the robot, a command request is sent to the server, texts, pictures, are displayed for the user to observe.

Server: Registration and management of Android users and robot users; processing of task texts, interpreting strings as tasks that the robot needs to perform, publishing tasks and necessary parameters to specific APIs; target recognition and scene understanding, pictures uploaded to the robot Processing, returning the result of target recognition and scene understanding. The robot returns the task execution result to the Android end after completing the task.

Robot: The robot acts as an executor to communicate with the server through the HTTP protocol to obtain tasks, accepts commands from the console to complete tasks posted by the user, and gives feedback to the server when the task is completed.

The user uses the Android side to operate the process. First, the user login the Android software, selects the corresponding online robot, starts voice recognition, and issues a command. At this time, the task is successfully posted and jumps to the task query interface. If it is a picture task, click to display the picture and long press to save. If you need other functions, you can make a corresponding selection in the menu bar on the left side of the main page.

3. System development and implementation

3.1. Android development and implementation

This article is based on the Android system development terminal platform, the device uses the Android-side smart phone. The command module, the query result module, the history module, and the HTTP communication module are four part of its issued.
The main functions of the Android side include login and logout registration, release commands, query results, and history records.

Release command module: This interface is the main interface of the app. The main interface effect is shown after the login is successful, the jump is entered. The integrated Baidu speech recognition is used to convert the user's voice command into text, which makes the command release easier. Remove the tedious typing function. The spinner control is placed in the main interface of the release command, which is refreshed in real time. After clicking, the online robot list is displayed for the user to select. When the robot is selected, it is judged whether the robot is in an idle state, and the problem that different users control the same robot is prevented. TextView is used to display the text of the voice conversion, and the change can be recognized again if it is different from the user's intention. All data selection is complete. Click the Publish button and the process of issuing a command is executed. When the user does not need to use the robot again, he can click the toolbar control in the upper right corner to release the occupied robot, and other users can use the robot. Swipe right to the left of the main page to display the menu bar. Click on the list to jump to the corresponding module.

Query result module: When the task is published, the interface will jump to the query interface. The interface displays 9 text boxes to display specific information about the task. When the task status is "F" to stop refreshing the network request, the notification bar prompts the user that the robot task is completed. If the task request is a task related to the picture, you can view the large picture after obtaining the picture address, and long press the picture to select whether to save to the local.

History module: Entered from the menu bar of the release command module. The module consists of a ListView control. Each item has four TextView controls that display the user, robot, task description, and time. ListView is loaded in batches, loading 20 at a time to increase the loading speed.

HTTP communication module: This module is responsible for obtaining the robot status, task status, and sending data requests to the server from the server. Use the okhttp framework in Android to implement different data requests by using the post, get, and put methods to send the user name at login to each server address.

3.2. User registration login section

The system is developed using the Django and restframework framework.

The Android side registration method is to access the registration url and fill in the registration information in the form of a form, POST to the server side. The server returns the registration result. The information sent by registration is as follows:

"username": "user1",
"password": "mypassword",
"equipment_type": "android",
"register_date": "date",
"accept_value": "picture",
Registration success: return status code 200.
Registration failed: return status code 404.

The Android login method accesses the login url and fills in the account and password in the form of a Json string, POST to the server. After receiving the login request, the server queries the database and verifies the login information. If the login is successful, the user_id is returned. Otherwise, an error message is returned. If the user_id is returned, the Android user needs to have the ID locally, and fill in the url during the task sending process. After the login is successful, an entry is added to the console state table structure the information sent by the login is as follows:

"username": "user1",
"password": "mypassword",
Login successful: return status code 200 and get user_id.
Login failed: return status code 404.
3.3. **ROS robot registration user process**
The same as the Android registration method, registration information:

```json
"group_id": "1",
"username": "user1",
"password": "mypassword",
"equipment_type": "robot",
"function_value": "take_picture",
"function_describe_file": "file",
```

3.4. **ROS robot login process**
Access the login url and fill in the account and password in the form of a form, POST to the server. After receiving the login request, the server queries the database and verifies the login information. If the login is successful, the user_id is returned. Otherwise, an error message is returned. If the user_id is returned, the user of the robot needs to have the ID locally, and the url is filled in the task query process. After the login is successful, an entry is added to the execution state table structure (ExecuteUserStatus).

- Login information: "username": "robotuser1"
- "password": "mypassword",
- login successful:
  - Returns the status code 200 and gets the robot_user_id.
- Login failed:
  - Return status code 404.

3.5. **Task sending process**
After the Android console user logs in, the voice is converted into a string, with its own control terminal ID, the selected execution end ID, the access server assigns the url to the user, and the POST task information in the form of a form. The server returns to receive the command results and decides whether to create a task. The task sends the following information:

```json
"command_id": "1",
"execute_id": "1",
"command": "command_string",
```

The server created the task successfully: return status code 200.
The server creation task failed; return status code 404.

After the Android user posts the task to the server, the task state table of the control terminal is queried. After the task is successfully completed, if the executed task has a camera function, the Android side needs to parse the Json string and access the url of the image to display the result image.

3.6. **ROS robot query task**
The ROS robot accesses the specific url according to the robot_user_id assigned by the server to query its current task ID. The server returns the current state table of the robot in the corresponding url. Analyze the required tasks and parameters.

```json
"function_task": "take_picture",
"parameter": "None",
```

The robot returns the task result:
The ROS robot returns the result of the execution and the result is uploaded via the url.
If the upload is successful, the status_code is 200.

After the commander issues the task, query the commander_status to get the task_id, query the progress of the current task according to the task_id, and view the contents of the task to update the commander_status.

The executor queries whether the executor_status has a task dispatch. If the task_id is not empty, it indicates that there is a task to be executed. The task is queried according to the task_id, and the
related function and parameter are called. During the execution, the task is continuously queried and the task (Update) is updated.

4. Robot side implementation
The hardware platform mainly adopts the test development platform built by hardware such as Xiaoqiang robot, IMU (gyroscope) and depth camera. The software platform uses the robot operating system ROS, etc.

![Figure 2. Hardware connection simulation diagram.](image)

The hardware components required for this article include Xiaoqiang Robot, IMU (Gyro), Deep Camera Kinect, etc. Figure 2 shows a hardware connection simulation diagram.

The ROS end communicates with the server using the HTTP protocol. In the development of ROS, the ROS-based function packages usb_cam and nav_test are used as drivers for the camera and mobile functions for secondary development.

The ROS end communicates with the server: the ROS end communicates with the server by using the Http protocol, and the task is queried by polling. The task flow is as follows: Figure 3: After the automatic login, the executor's user name is obtained and the task ID (task_id) is obtained. When there is no task, the task ID (task_id) is empty, the server accepts the task will generate the task ID (task_id); after obtaining the task_id through the get request, it will enter the task page through the task_id, and obtain the task status through the get request on the task page, the task status Execute the task for Executing and use the data required to upload the server task with the put request to feed back the task (listen to the upload status).

![Figure 3. Task flow chart.](image)

Robot ROS end communication: The total communication architecture of the robot is shown in Figure 3. Communication between nodes and nodes in ROS: including topics and services. In the case where communication that does not require timeliness can be used, the robot is required to synchronize with the control during the movement of the robot.

Image acquisition module: The image acquisition is based on the usb_cam function package, in which the video device of the camera is opened by using the V4L2 technology, and the video data is recorded in the memory and stored in the memory by the data structure and the underlying V4L2 drive interface according to me,. The video data node / usb_cam / image_raw publishes the video data. Image_view uses the image_transport, cv_bridge, and opencv to obtain the image information of the
message (sensor_msg) by subscribing to the video data node / usb_cam / image_raw. I save the image and upload the image to the server through network communication.

APP control mobile module: Our robot design is based on the APP-side control. The robot is remotely controlled by the server, so that the robot's moving moment can be controlled by people. It can increase the controllability of the robot and facilitate the angle of the image acquisition. Control, the robot and the server are required to maintain communication at all times. I use the polling method to update the movement state of the robot (UP stands for forward, DOWN stands for backward, LEFT stands for left turn, RIGHT stands for right turn, STOP stands for stop), and the control release node is used. The linear speed (linear_speed) and angular velocity (angular_speed) of cmd_vel reach the defined state; when the APP side updates the moving state, the moving state of the ROS terminal changes.

Autonomous obstacle avoidance of the mobile module: Autonomous obstacle avoidance utilizes the point cloud generated by kinect's depth camera, and the released point cloud is converted into an obstacle grid map by image_pipeline. After the nav_test package starts the chassis navigation program, it will automatically process the analysis of the obstacle distribution map, and then move autonomously according to the target target navigation point as shown in Figure 4.

![Figure 4. Robotic autonomous obstacle avoidance navigation.](image)

Scene recognition module: As an important basis of the system, the role of the scene recognition module is to identify the scene in the picture uploaded by the robot and understand the scene in which the robot is located. Model architecture for scene recognition:

System architecture: This system mainly uses the scene recognition algorithm as the basis for judging the environment of the robot. The scene recognition design diagram is as follows

Flowchart Figure.5 for scene recognition:

![Figure 5. Scene Recognition.](image)
The scene recognition algorithm can be used to analyze the scene of the robot according to the photos taken by the robot. When the user uploads the task, the server will query the permissions owned by the user in the current scene according to the user's commander_id, and if there is no permission, the request will be rejected.

The model as a whole can be divided into five parts, the first four parts are convolutional layer groups, each group contains several convolutional layers, and the fifth part is a fully connected layer. The input size of the picture is 224×224. After the first layer, the output size is 112×112. The output of the current layer is also the input of the next layer. The specific conversion formula is as shown in equations (1) and (2). Where out_length indicates the length of the output picture, in_length indicates the length of the input picture, length indicates the length of the convolution kernel, length-padding_length indicates the length of the full 0 padding, when the output width is solved, the same as this, will not be described here. In fact, in order to ensure that the size of the matrix after convolution is unchanged, each convolutional layer will be filled with all 0s, and the output size is as shown in equations (3) and (4).

\[
\text{out}_{\text{length}} = \frac{\text{in}_{\text{length}} - \text{filter}_{\text{length}} + 2 \times \text{zero} - \text{padding}_{\text{length}}}{\text{stride}_{\text{length}}} + 1 \quad (1)
\]

\[
\text{out}_{\text{length}} = \frac{\text{in}_{\text{length}} - \text{filter}_{\text{length}} + 2 \times \text{zero} - \text{padding}_{\text{length}}}{\text{stride}_{\text{length}}} + 1 \quad (2)
\]

\[
\text{out}_{\text{length}} = \frac{\text{in}_{\text{length}}}{\text{stride}_{\text{length}}} \quad (3)
\]

\[
\text{out}_{\text{length}} = \frac{\text{in}_{\text{length}}}{\text{stride}_{\text{length}}} \quad (4)
\]

In the table.1, Conv1 is a convolutional layer with a convolution kernel of 7×7, an input dimension of 3, an output dimension of 64, and a step value of 2. The data in Conv2_x first passes through a size of 3×3 and a step value of 2. The largest pooling layer is then composed of three convolutional layer groups, each consisting of two convolutional kernels with a size of 3×3 and an output dimension of 64. Conv3_x, Conv4_x, Conv5_x are similar. In the last part, the data goes through an average pooling layer and a fully connected layer, outputting a confidence level of 80 categories.

| name   | output | model (ResNet34) |
|--------|--------|------------------|
| Conv1  | 112 × 112 | 7 × 7, 64, stride 2 |
| Conv2_x | 56 × 56 | 3 × 3 maxpool, stride 2 |
| Conv3_x | 28 × 28 | 3 × 3, 64 |
| Conv4_x | 14 × 14 | 3 × 3, 128 |
| Conv5_x | 7 × 7 | 3 × 3, 256 |
|       | 1 × 1 | average pool, 80d fc |

5. System implementation effects
After opening the software, the speech recognition is completed, select the robot you want to use, and issue the command. The Android side jump display task query interface is as shown in Figure.5, the task status is "E", and the picture address is "null". When the robot performs the task and the server
side modifies the data, the Android end receives the corresponding change. As shown in Figure 6, the task status is “F”, the image address is displayed, and the image address can be clicked, as shown in Figure 6. As shown, long press can save the picture taken by the robot. Prove that the system can provide users with services for publishing tasks and viewing results.

Figure 6. Result of the software.

Acknowledgments
We thank Ye Jian for his helpful comments on a previous version of this article.

References
[1] “World Robotics 2010”, Int. Federation Robot. Statist. Dept., 2010.
[2] B. Gates, “A robot in every home”, Scientific Amer. Mag., vol. 296, no. 1, pp. 58–65, 2007.
[3] P. Wurman, R. D’Andrea, and M. Mountz, “Coordinating hundreds of cooperative, autonomous vehicles in warehouses”, AI Mag., vol. 29, no. 1, p. 9, 2008.
[4] E. Guizzo, “Three engineers, hundreds of robots, one warehouse”, IEEE Spectr., vol. 45, no. 7, pp. 26–34, 2008.
[5] T. Buchheim, G. Kindermann, and R. Lafrenz, “A dynamic environment modelling framework for selective attention”, in Proc. IJCAI Workshop: Issues in Designing Physical Agents for Dynamic Real-Time Environments: World Modeling, Planning, Learning, and Communicating, 2003, pp. 91–98.
[6] M. Roth, D. Vail, and M. Veloso, “A real-time world model for multi-robot teams with high-latency communication”, in Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems 2003(IROS’03), vol. 3, pp. 2494–2499.
[7] J. Fenwick, P. Newman, and J. Leonard, “Cooperative concurrent mapping and localization”, in Proc. IEEE Int. Conf. Robotics and Automation 2002 (ICRA’02), vol. 2, pp. 1810–1817.