

# Handbook

## Project Title:

## Secure Shipping Using Robotic Delivery System

Please see Code files in GitHub:

<https://github.com/Jiapei-Yang/Secure-Robotic-Shipping>

## Acknowledgments:

WangBenYan. [https://blog.csdn.net/qq\\_18808965/article/details/90262113](https://blog.csdn.net/qq_18808965/article/details/90262113)

Harshvardhan Gupta. <https://github.com/harveyslash/Facial-Similarity-with-Siamese-Networks-in-Pytorch>

Li, W, Zhao, R, and Wang, X. "Human Reidentification with Transferred Metric Learning." Asian conference on computer vision (2012): 31-44. Web.

## This document is divided into three parts:

1. Explanation of files in project *Secure Shipping Using Robotic Delivery System*.
2. Guidance of simple experiment.
3. Guidance of system implementation (demo).

To get the dataset *CUHK01*, visit

[https://www.ee.cuhk.edu.hk/~xgwang/CUHK\\_identification.html](https://www.ee.cuhk.edu.hk/~xgwang/CUHK_identification.html)

Fig. 1 and Fig. 2 show how the system work. For more details please read the project's paper.

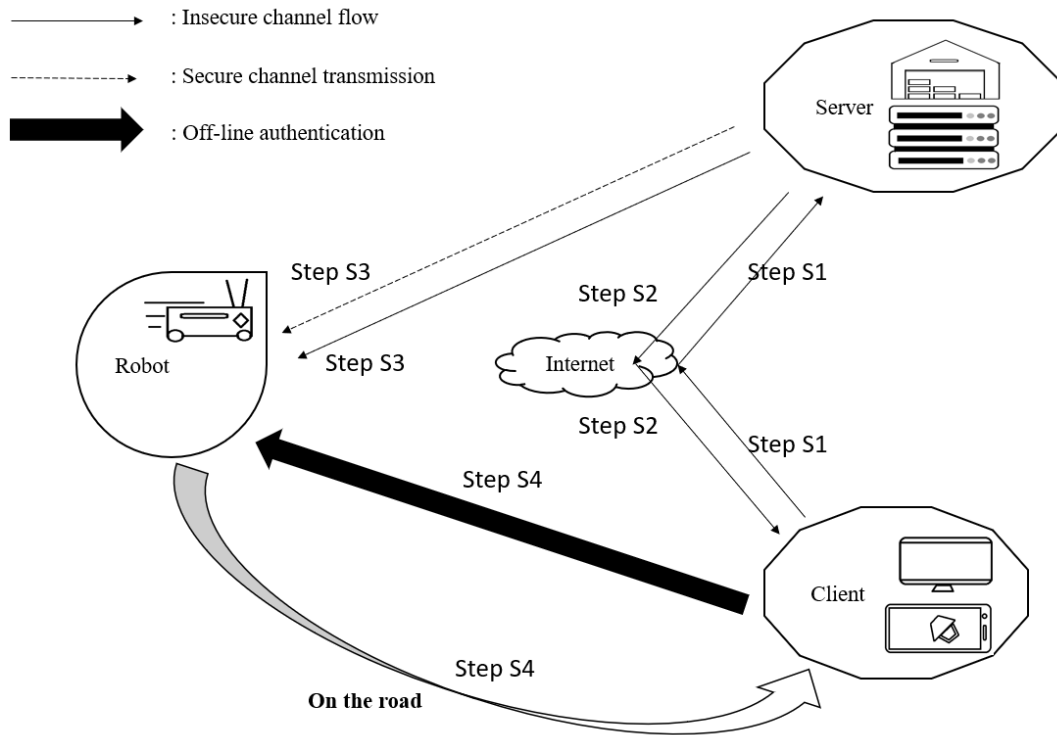


Figure 1. The proposed robotic delivery system

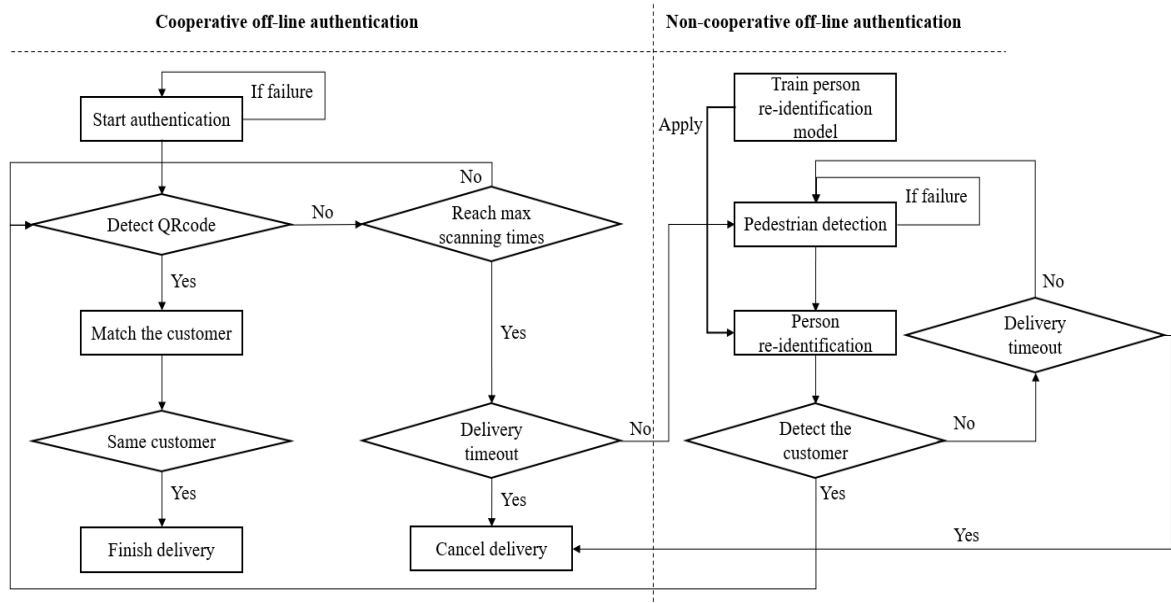


Figure 2. Details of off-line authentication methods

Fig. 3 illustrates the implementation of the system:

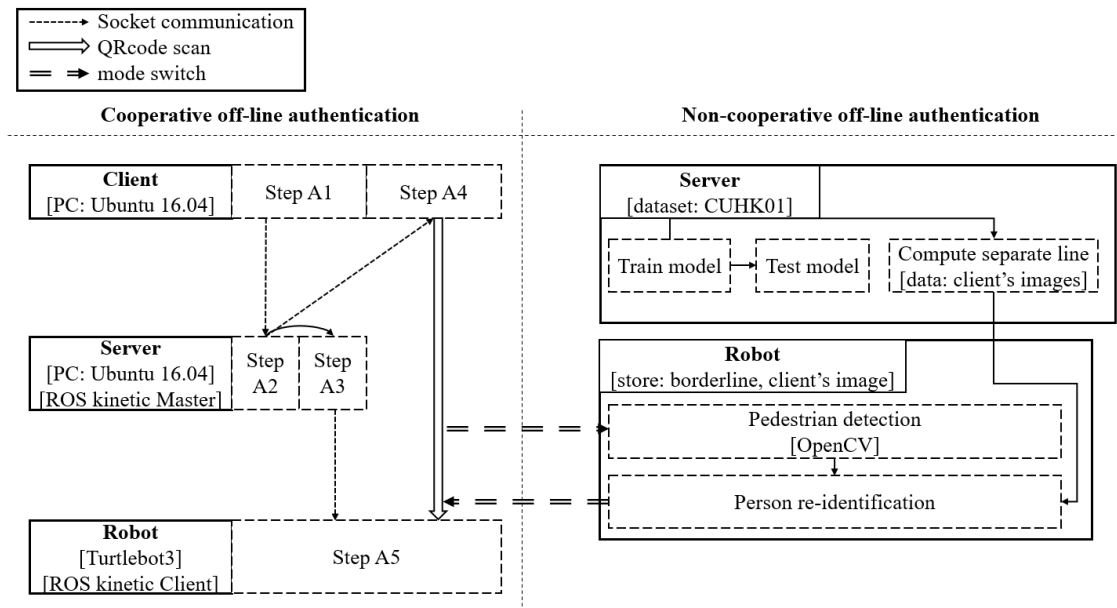


Figure 3. The proposed system implementation

# 1. Explanation of files

## I. Proverif

This folder is for security analysis in the cooperative authentication of our system.

**Verifier.pv:** Use Proverif to analyse security of the communication protocol in our cooperative authentication. This includes analysing data disclosure and forged attacks.

**Result:** Result of verifier.pv.

## II. Noncooperative part in Jupyter

This folder includes implement of our system's noncooperative authentication in Jupyter platform. It is much more convenient to test our noncooperative part's performance in this emulation with Jupyter.

**Data:** Dataset after pre-processing. It includes four sub-folders, and each one is corresponding to a Jupyter file. We can run classify.py to do pre-processing to add the training set folder and the test set folder here.

**classify.py:** Pre-processing code. It splits the original data into a training set and a test set.

**train.ipynb:** Model training code. It stores the trained model and names it as net\_test.pth.

**test\_CMC.ipynb:** Model testing code. It contains the test result (CMC figure).

**re\_identify\_prepare.ipynb:** Calculate the max distance between any two images of the client.

**re\_identify\_robot.ipynb:** Calculate the distance between the captured image of pedestrian and the client's image, and judge whether they are from the same person.

**net\_test.pth:** The saved model. We can generate it in train.ipynb.

### III. Cooperative part

This folder includes implement of our system's cooperative authentication. We utilize three Python code files in Python 2.7 for cooperative authentication.

**true-client.py:** Run this file in the client. It controls behaviours of client in cooperative authentication: Client sends a request by TCP socket protocol to the server, and waits for a reply. Upon receiving the answer from the server, client derives public key and nonce in it. By using these nonce, client generates an authentication information, encrypts it with the public key and generates the QR code. Finally, client shows the QR code in the screen.

**true-server.py:** Run this file in the server. It controls behaviours of server in cooperative authentication: Server firstly waits for the request from clients by TCP socket protocol. Upon receiving it, server authenticates the identity of the client, and generate some nonce and a pair of one-time-use public key and private key. Then, server sends the nonce and public key back to the client. Next, server generates the same authentication information of that in the client, encrypts it with the public key and transfers it to the robot. Finally, server transfers file of the private key and the serial number to the robot, too.

**true-robot.py:** Run this file in the robot. It controls behaviours of robot in cooperative authentication: First of all, robot receives information from the server, and derives the serial number. With the help of it, robot is able to quickly recognize the identity of the client after QR code scanning of him. If the scanning and recognition are successful, robot decrypts two encrypted messages from the server and the client, and compare the plaintexts. As soon as they match, robot finishes the delivery work. If the scanning process fails for 5 times, robot turns to noncooperative mode, and waits for the signal of reidentification of the client, and once again tries to scan the QR code. If robot fails the QR code scanning part again, timeout of the delivery job occurs, and we stop the express.

#### IV. Noncooperative part

This folder includes implement of our system's noncooperative authentication. We utilize 6 Python code files in Python 3.7 for noncooperative authentication. 4 files are in the server and the other 2 runs in the robot.

**classify.py:** Run this file in the server. In dataset pre-processing, it splits the original data into a training set and a test set.

**train.py:** Run this file in the server. This code file trains the model of person reidentification. It randomly reads two images from the same or different person, and makes the distance between them larger for different people and closer for the same identity. Then we store the trained model.

**test\_CMC.py:** Run this file in the server. It assesses the model by drawing the Cumulative Match Curve (CMC) and comparing with other algorithm's performance.

**re\_identify\_prepare.py:** Run this file in the server. It calculates the max distance between any two images of the client. This value is used as a separate line: if any two pictures' distance is closer than this, they are seen from the same person.

**capture.py:** Run this file in the robot. It directs robot to capture the image, do pedestrian detection, and resize the person's image.

**re\_identify\_robot.py:** Run this file in the robot. Robot calculates the distance between two images: One is the client's image, and the other is the captured image of pedestrian from capture.py. Compare this distance to the max distance from re\_identify\_prepare.py. If this distance is smaller, the two images are from the same person, which means robot has found the client.

## 2. Guidance of simple experiment

### I. Formal Security Analysis in Cooperative Authentication

Put verifier.pv in folder that contains proverif.exe.

Open command (CMD).

Change the current working directory into the folder that has proverif.exe.

Insert the command “*proverif verifier.pv*” and press enter.

### II. Noncooperative authentication’s simulation in Jupyter

Put the CUHK01 dataset under “/data/”, and run classify.py to add the training set and test set open any Jupyter file and run it directly. However, training and testing could be very time-consuming. We can also see the result of previous running directly in Jupyter files.

This part’s work is briefly introduced in the video:

<https://www.youtube.com/watch?v=zgD6tC4vGLM>

### 3. Guidance of system implementation (demo).

We will introduce the environmental requirements, and how to run the demo.

#### I. Environmental requirements:

##### a. Cooperative off-line authentication:

Table 1 shows the implementation environment of cooperative authentication. We can use two laptops as the client and the server, and Turtlebot3 as the robot. Here we install Ubuntu 16.04 in the client and the server and Ubuntu mate 16.04 in the robot. Then utilize ROS kinetic, which is recommended in Turtlebot3 and supports Python 2, in the server and the robot and set the server as the master.

Terminals	Machine	System	Python Version	ROS Node	IP Address	Python third-party libraries
Client	Laptop	Ubuntu 16.04	Python 2.7		192.168.43.183	qrcode Crypto rsa
Server	Laptop	Ubuntu 16.04	Python 2.7	ROS kinetic Master node	192.168.43.97	rsa Crypto
Robot	Robot (Turtlebot3)	Ubuntu mate 16.04	Python 2.7	ROS kinetic Normal node	192.168.43.74	rsa PIL pyzbar

Table 1. Implementation environment of cooperative authentication

##### b. Non-cooperative off-line authentication:

The following table shows the implementation environment of noncooperative authentication. We use the same devices of the cooperative part, but focus on the server and the robot. In addition, virtualenv is utilized for building the Python 3 virtual environment in two devices, and we use Python 3 to fulfil requirement of some libraries in the area of computer vision. To train and test our model we utilize CUHK01 dataset.

Terminals	Machine	System	Python Version	ROS Node	IP Address	Python third-party libraries
Server	Laptop	Ubuntu 16.04	Python 3.7	ROS kinetic Master node	192.168.43.97	virtualenv numpy matplotlib cv2 PIL torch torchvision
Robot	Robot (Turtlebot3)	Ubuntu mate 16.04	Python 3.7	ROS kinetic Normal node	192.168.43.74	virtualenv numpy imutils cv2 PIL torch torchvision

Table 2. Implementation environment of noncooperative authentication



## II. Detailed operation

### a. Cooperative off-line authentication demo:

We can see the implementation of cooperative authentication in the video:

<https://www.youtube.com/watch?v=-cANuZxD9uQ>

#### **Step C1:** Prepare work:

Set the environment based on Python 2 in Table 5.a.

Use ROS to create a map of the workplace.

(see <https://emanual.robotis.com/docs/en/platform/turtlebot3/slam/>)

Change the current working directory to “/home/username/Desktop/project” in three terminals.

Put true-client.py in the client.

Put true-robot.py in the robot.

Put true-server.py in the server.

#### **Step C2:** Launch ROS and robot equipment:

In the server, launch ROS: *roscore*

In the robot, launch the robot: *roslaunch turtlebot3\_bringup turtlebot3\_robot.launch*

#### **Step C3:** Run scripts:

Server launches a new terminal and start services: *python true-server.py*

Robot launches a new terminal and start services: *python true-robot.py*

Next, client sends the request: *python true-client.py*

As a result, the robot shows “please input Y when robot arrives:”, which means server has received the client’s request and distributed information to the client and the robot.

#### **Step C4:** navigation:

Here we make a manual navigation.

Server launches a new terminal and insert:

```
export TURTLEBOT3_MODEL=waffle_pi
```

Then run the command to launch *Rviz* for navigation:

```
roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:=$HOME/true-map.yaml
```

Press “2D Pose Estimate” to correct start position, and “2D Nav Goal” to select aimed destination. Robot plans the path and go there automatically as shown in Fig. 4.

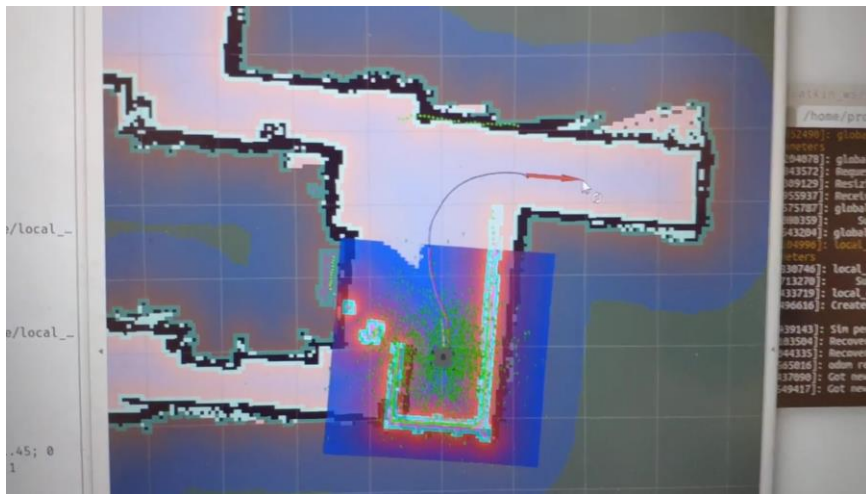


Figure 4. Navigation in the implementation of cooperative authentication

#### **Step C5: QR code scan:**

Upon arriving the destination, insert “Y” in the robot as a signal of completed navigation. Then, the robot automatically tries to scan QR code shown by the client. The scanning work succeeds and the QR code is authenticated, so the robot shows “matched!” and complete the delivery as shown in Fig. 5 and Fig. 6.

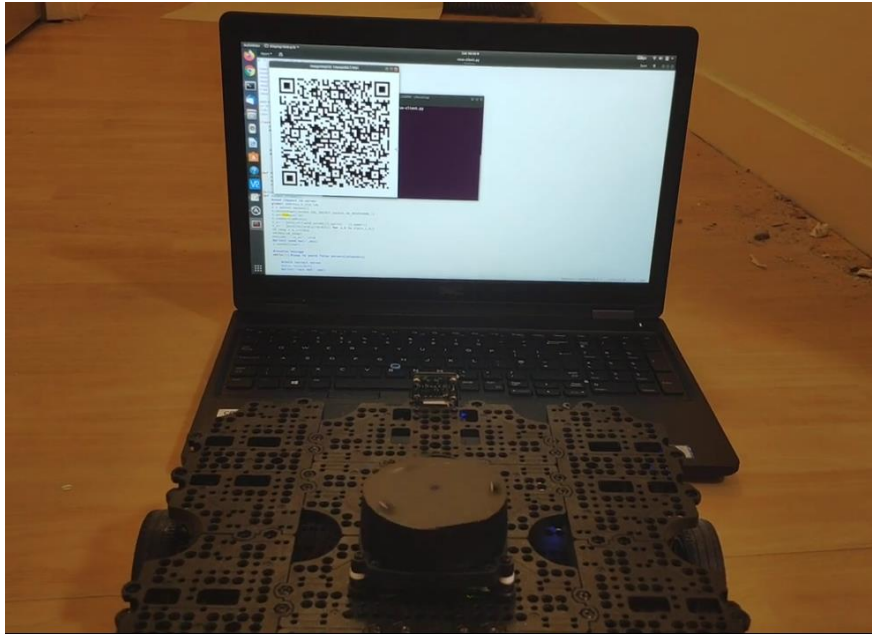


Figure 5. QR code scanning in the implementation of cooperative authentication

```

ubuntu@ubiquityRobot: ~/Desktop
/home/ubuntu/catkin_ws/src... x ubuntu@ubiquityRobot: ~/D... x
Preview window 0,0,1024,768
Opacity 255
Sharpness 0, Contrast 0, Brightness 50
Saturation 0, ISO 0, Video Stabilisation No, Exposure compensation 0
Exposure Mode 'auto', AWB Mode 'auto', Image Effect 'none'
Flicker Avoid Mode 'off'
Metering Mode 'average', Colour Effect Enabled No with U = 128, V = 128
Rotation 0, hflip No, vflip No
ROI x 0.000000, y 0.000000, w 1.000000 h 1.000000
Camera component done
Encoder component done
Starting component connection stage
Connecting camera preview port to video render.
Connecting camera stills port to encoder input port
Opening output file 0.jpg
Enabling encoder output port
Starting capture -1
Finished capture -1
Closing down
Close down completed, all components disconnected, disabled and destroyed
ET, get QRcode
matched!

```

Figure 6. Result of QR code scanning in cooperative authentication

b. Cooperative off-line authentication demo:

This part executes cooperative authentication twice: A failed one before shifting to the noncooperative part, and the other successful one when the robot recognizes the client and shifts the mode back.

We can see a video of the implementation above:

<https://www.youtube.com/watch?v=IN5tngrdmds>

**Step N1: Prepare work**

Do preparing work in Step C1 in the implementation of cooperative authentication.

Set the virtual environment based on Python 3 in Table 5.b.

(see <https://code-maven.com/slides/python/virtualev-python3>)

Put capture.py and re\_identify\_robot.py in the robot.

Put classify.py, train.py, test\_CMC.py, re\_identify\_prepare.py in the server.

Put dataset (CUHK01) as “/data/CUHK01/campus”.

Next, server activates python3 environment and does pre-processing:

```
source ~/venv3/bin/activate
```

```
python classify.py
```

To finish the pre-processing work in the server, put images of the client into “/data/reid\_prepare”, and copy one of them into “/data/reid\_robot” in the robot as the comparison image. Therefore, we have four folders to store dataset:

/data/training\_set: Training set. Stored in the server.

/data/test\_set: Testing set. Stored in the server.

/data/reid\_prepare: All images of the client. Stored in the server.

/data/reid\_robot: An image of the client (a captured photo of pedestrian will be added here later).

**Step N2: Model training**

Then, train a model in the server:

*python train.py*

The model is stored as “/net\_test.pth”, and copy it to the robot.

### **Step N3: Optional Model testing**

Run the code in the server to plot CMC in test1.jpg:

*python test\_CMC.py*

To get the separate line used in the final comparison, run the code in the server:

*python re\_identify\_prepare.py*

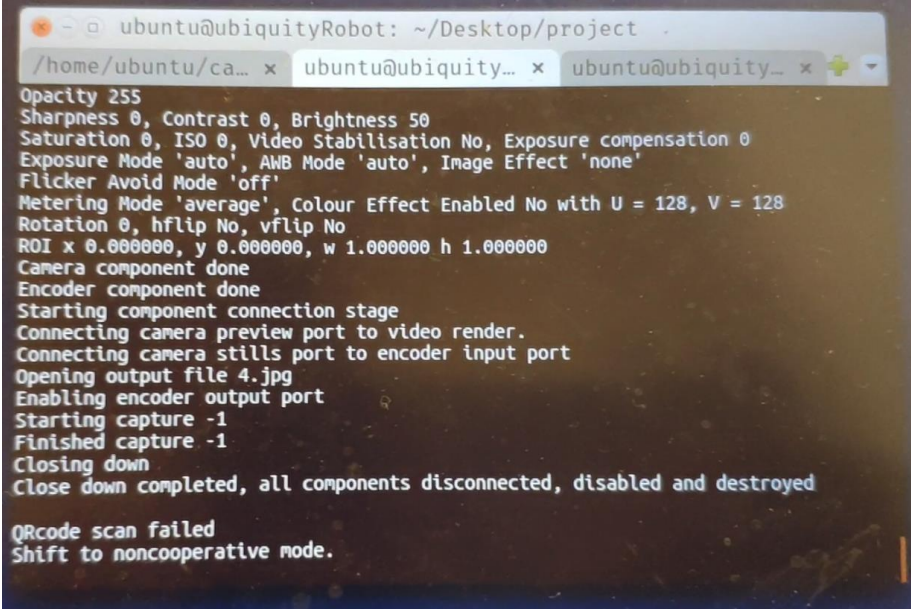
### **Step N4: Capturing images**

Do the cooperative authentication and make the QR code scanning unit fail as shown in Fig. 7

Robots runs:

*python capture.py*

In this way, we successfully captured an image of pedestrian, did pedestrian detection in it, resize the person’s image and stored it in “/data/reid\_robot”.

A terminal window titled 'ubuntu@ubiquityRobot: ~/Desktop/project' with three tabs. The output shows the initialization of camera and encoder components, followed by an attempt to connect to a video render and encoder input port. The process then opens an output file '4.jpg' and starts capturing. After finishing capture, it closes down, but the QR code scan fails, leading to a shift to noncooperative mode.

```
ubuntu@ubiquityRobot: ~/Desktop/project
/home/ubuntu/ca... x  ubuntu@ubiquity... x  ubuntu@ubiquity... x
Opacity 255
Sharpness 0, Contrast 0, Brightness 50
Saturation 0, ISO 0, Video Stabilisation No, Exposure compensation 0
Exposure Mode 'auto', AWB Mode 'auto', Image Effect 'none'
Flicker Avoid Mode 'off'
Metering Mode 'average', Colour Effect Enabled No with U = 128, V = 128
Rotation 0, hflip No, vflip No
ROI x 0.000000, y 0.000000, w 1.000000 h 1.000000
Camera component done
Encoder component done
Starting component connection stage
Connecting camera preview port to video render.
Connecting camera stills port to encoder input port
Opening output file 4.jpg
Enabling encoder output port
Starting capture -1
Finished capture -1
Closing down
Close down completed, all components disconnected, disabled and destroyed

QRcode scan failed
Shift to noncooperative mode.
```

Figure 7. Failed QR code scanning and shifting to noncooperative mode

### Step N5: Person reidentification

Next, robot runs the command to compare the similarity of the captured image and client's image in “/data/reid\_robot”.

```
python re_identify_robot.py
```

If the output is “same person”, robot successfully recognized the client, and the system switches back to cooperative mode as shown in Fig. 8 and Fig. 9.

### Step N6: QR code scanning again

Finally, input any character in the robot's terminal that runs true-robot.py so it can start to scan the QR code again and match as shown in Fig. 10 and Fig. 11.



Figure 8. Person detection and reidentification

```
ubuntu@ubiquityRobot: ~/Desktop/project
/home/ubuntu/ca... x ubuntu@ubiquity... x ubuntu@ubiquity... x
Saturation 0, ISO 0, Video Stabilisation No, Exposure compensation 0
Exposure Mode 'auto', AMB Mode 'auto', Image Effect 'none'
Flicker Avoid Mode 'off'
Metering Mode 'average', Colour Effect Enabled No with U = 128, V = 128
Rotation 0, hflip No, vflip No
ROI x 0.000000, y 0.000000, w 1.000000 h 1.000000
Camera component done
Encoder component done
Starting component connection stage
Connecting camera preview port to video render.
Connecting camera stills port to encoder input port
Opening output file capture.jpg
Enabling encoder output port
Starting capture -1
Finished capture -1
Closing down
Close down completed, all components disconnected, disabled and destroyed

Image Captured
(venv3) ubuntu@ubiquityRobot:~/Desktop/project$ python re_identify_robot.py
distance:7.256
same person
(venv3) ubuntu@ubiquityRobot:~/Desktop/project$
```

Figure 9. Result of person detection and reidentification



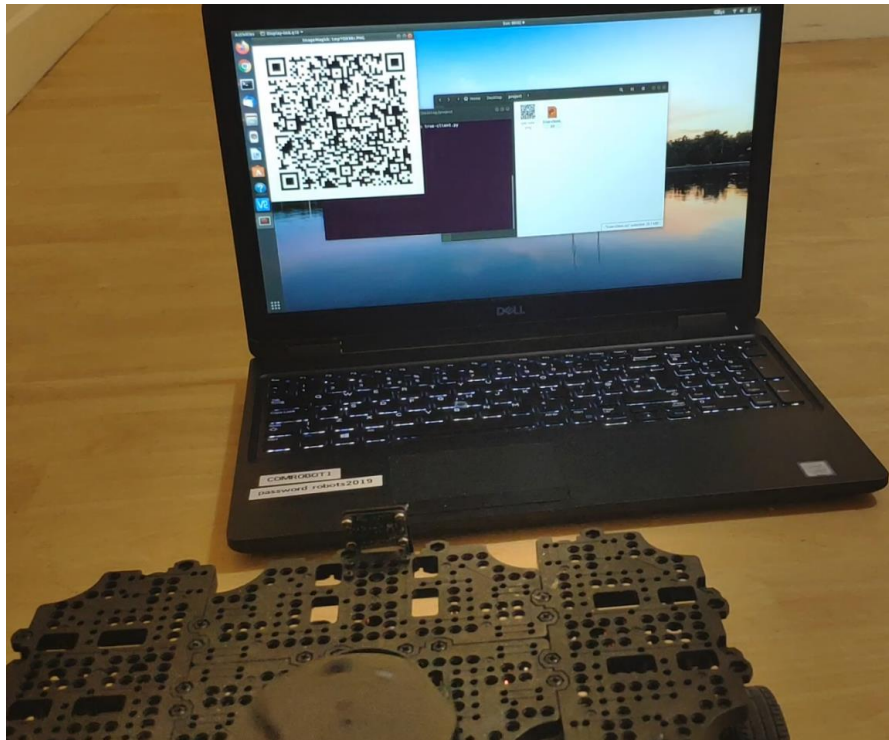


Figure 10. QR code scanning again

```

Opacity 255
Sharpness 0, Contrast 0, Brightness 50
Saturation 0, ISO 0, Video Stabilisation No, Exposure compensation 0
Exposure Mode 'auto', AWB Mode 'auto', Image Effect 'none'
Flicker Avoid Mode 'off'
Metering Mode 'average', Colour Effect Enabled No with U = 128, V = 128
Rotation 0, hflip No, vflip No
ROI x 0.000000, y 0.000000, w 1.000000 h 1.000000
Camera component done
Encoder component done
Starting component connection stage
Connecting camera preview port to video render.
Connecting camera stills port to encoder input port
Opening output file 1.jpg
Enabling encoder output port
Starting capture -1
Finished capture -1
Closing down
Close down completed, all components disconnected, disabled and destroyed

get QRcode
matched!

```

Figure 11. Result of QR code scanning again