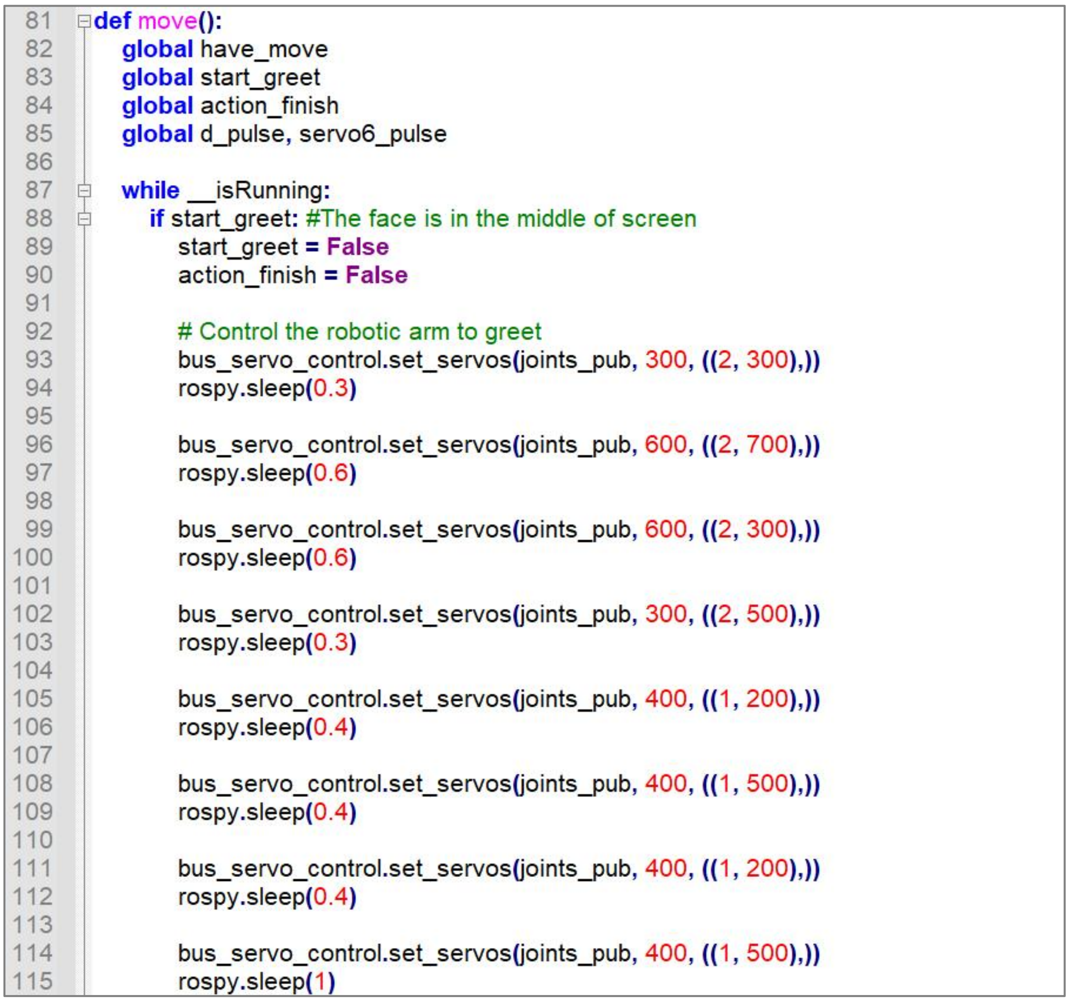
Lesson 3 Face Recognition

1. Working Principle

Firstly, the pan-tilt servo is set to rotate to search the human face. Then use the trained face to detect the face by the scaling screen, and convert the coordinates of the recognized face to the coordinates before scaling. Judge whether it is the largest face, and frame the recognized face.

Finally, control the servo angle to let robot perform the feedback after recognition.

The source code of program is located: in: **/home/ubuntu/armpi_pro/src/face_detect/scripts/face_detect_node.py**



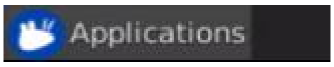
```
def move():
    global have_move
    global start_greet
    global action_finish
    global d_pulse, servo6_pulse

    while __isRunning:
        if start_greet: #The face is in the middle of screen
            start_greet = False
            action_finish = False

            # Control the robotic arm to greet
            bus_servo_control.set_servos(joints_pub, 300, ((2, 300),))
            rospy.sleep(0.3)

            bus_servo_control.set_servos(joints_pub, 600, ((2, 700),))
            rospy.sleep(0.6)

            bus_servo_control.set_servos(joints_pub, 600, ((2, 300),))
            rospy.sleep(0.6)

            bus_servo_control.set_servos(joints_pub, 300, ((2, 500),))
            rospy.sleep(0.3)

            bus_servo_control.set_servos(joints_pub, 400, ((1, 200),))
            rospy.sleep(0.4)

            bus_servo_control.set_servos(joints_pub, 400, ((1, 500),))
            rospy.sleep(0.4)

            bus_servo_control.set_servos(joints_pub, 400, ((1, 200),))
            rospy.sleep(0.4)

            bus_servo_control.set_servos(joints_pub, 400, ((1, 500),))
            rospy.sleep(1)
```

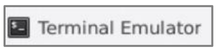
2. Operation Steps

It should be case sensitive when entering command and the “Tab” key can be used to complete the keywords.

2.1 Enter Game

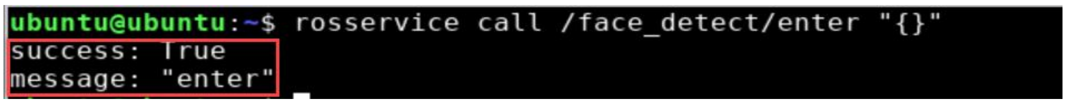
1) Turn on ArmPi Pro and connect to the system desktop via No Machine.

2) Click 

and select 

in pop-up interface to open the terminal.

3) Enter command “**rosservice call /face_detect/enter "{}"**” and press “Enter” to enter this game. After entering, the prompt will be printed, as the figure shown below:



```
ubuntu@ubuntu:~$ rosservice call /face_detect/enter "{}"
success: True
message: "enter"
```

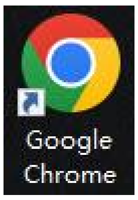
2.2 Start image transmission

2.2.1 Start with browser

To avoid consuming too much running memory of Raspberry Pi. It is recommended to use an external browser to open the transmitted image.

The specific steps are as follows:

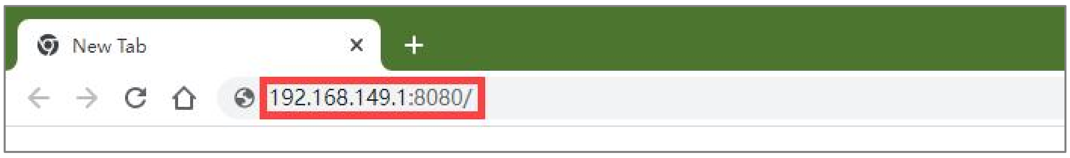
1) Select a browser. Take Google Chrome as example.



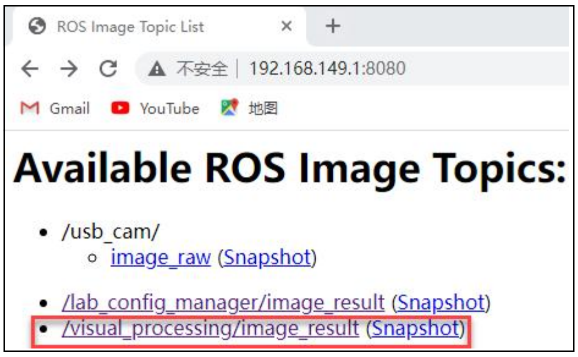
2) Then enter the default IP address “192.168.149.1:8080/” (Note: this IP address is the default IP address for direction connection mode. If it is LAN

mode, please enter “Device IP address+：8080/” such as “192.168.149.1:8080/”) If fail to open, you can try it several times or restart camera.

Note: If it is in LAN mode, the method to obtain device IP address can refer to “10.Advanced Lesson”/ 1.Network Configuration Lesson/ LAN Mode Connection.

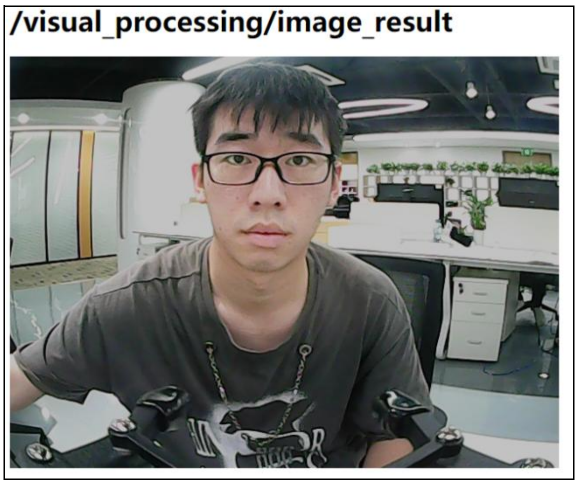


3) Then, click the option shown in the following figure to open the display window of the transmitted image.



Available ROS Image Topics:

- /usb_cam/
  - image_raw (Snapshot)
- /lab_config_manager/image_result (Snapshot)
- /visual_processing/image_result (Snapshot)



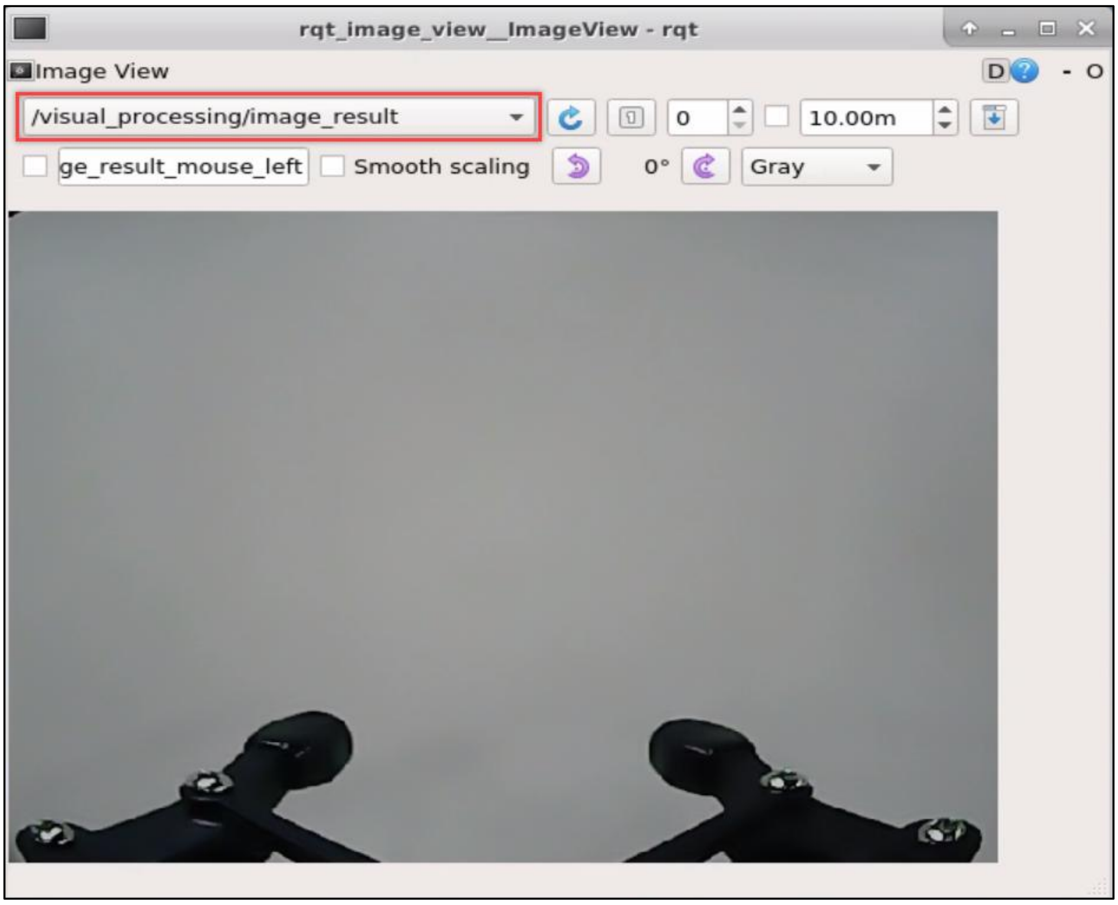
2.2.2 Start with rqt

1) After completing the steps of “2.1 Enter Game” and do not exit the terminal, open a new terminal.

2) Enter command “rqt_image_view” and press “Enter” to open rqt.

```
ubuntu@ubuntu:~$ rqt_image_view
```

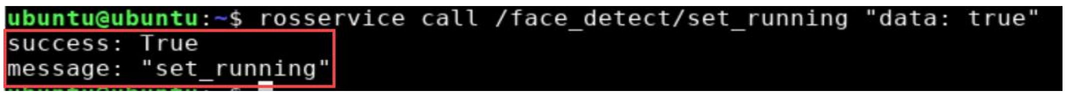
3) Click the red box as the figure shown below, select "/visual_processing/image_result" for the topic of line following and remain other settings unchanged.


rqt_image_view__ImageView - rqt
Image View
/visual_processing/image_result
0
10.00m
ge_result_mouse_left
Smooth scaling
0°
Gray

Note: After opening image, the topic option must be selected. Otherwise, after starting game, the recognition process can not be displayed normally.

2.3 Start Game

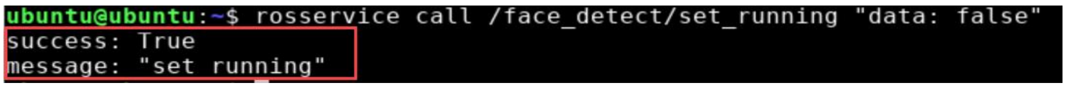
Now, enter the terminal according to the steps in "2.1 Enter Game" and input command "**rosservice call /face_detect/set_running "data: true"**". Then if the prompt shown in the following red box appears, which means game has been started successfully.



```
ubuntu@ubuntu:~$ rosservice call /face_detect/set_running "data: true"
success: True
message: "set_running"
```

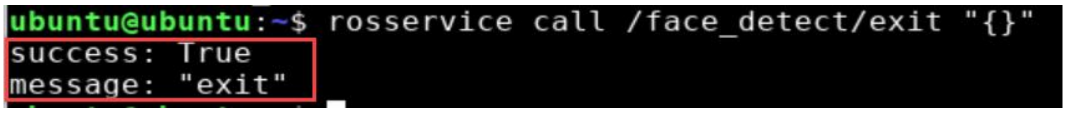
2.4 Stop and Exit

1) If want to stop the game, enter command "**rosservice call /face_detect/set_running "data: false"**".



```
ubuntu@ubuntu:~$ rosservice call /face_detect/set_running "data: false"
success: True
message: "set_running"
```

2) If want to exit the game, enter command "**rosservice call /face_detect/exit "{}"**" to exit.



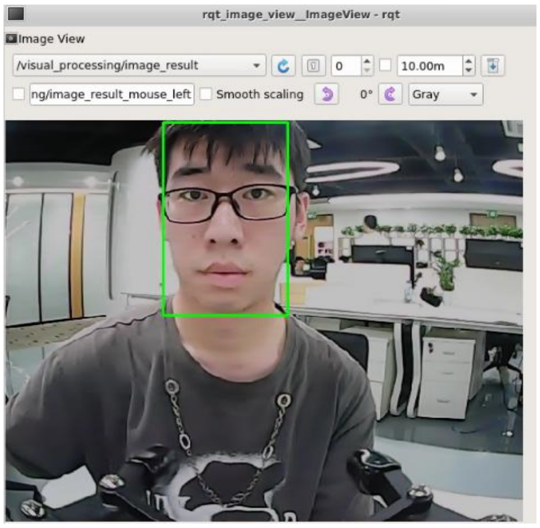
```
ubuntu@ubuntu:~$ rosservice call /face_detect/exit "{}"
success: True
message: "exit"
```

Note: Before exiting the game, it will keep running when Raspberry Pi is powered on. To avoid consume too much running memory of Raspberry Pi, you need to exit the game first according to the operation steps above before performing other AI vision games.

3) If want to exit the image transmission, press "Ctrl+C" to return and open the terminal of rqt. If fail to exit, please keep trying several times.

3. Project Outcome

After starting the game, the robotic arm will search for human face from side to side. Then, the human face will be framed in rqt tool after recognition and the gripper of robotic arm will rotate left and right before opening and closing.

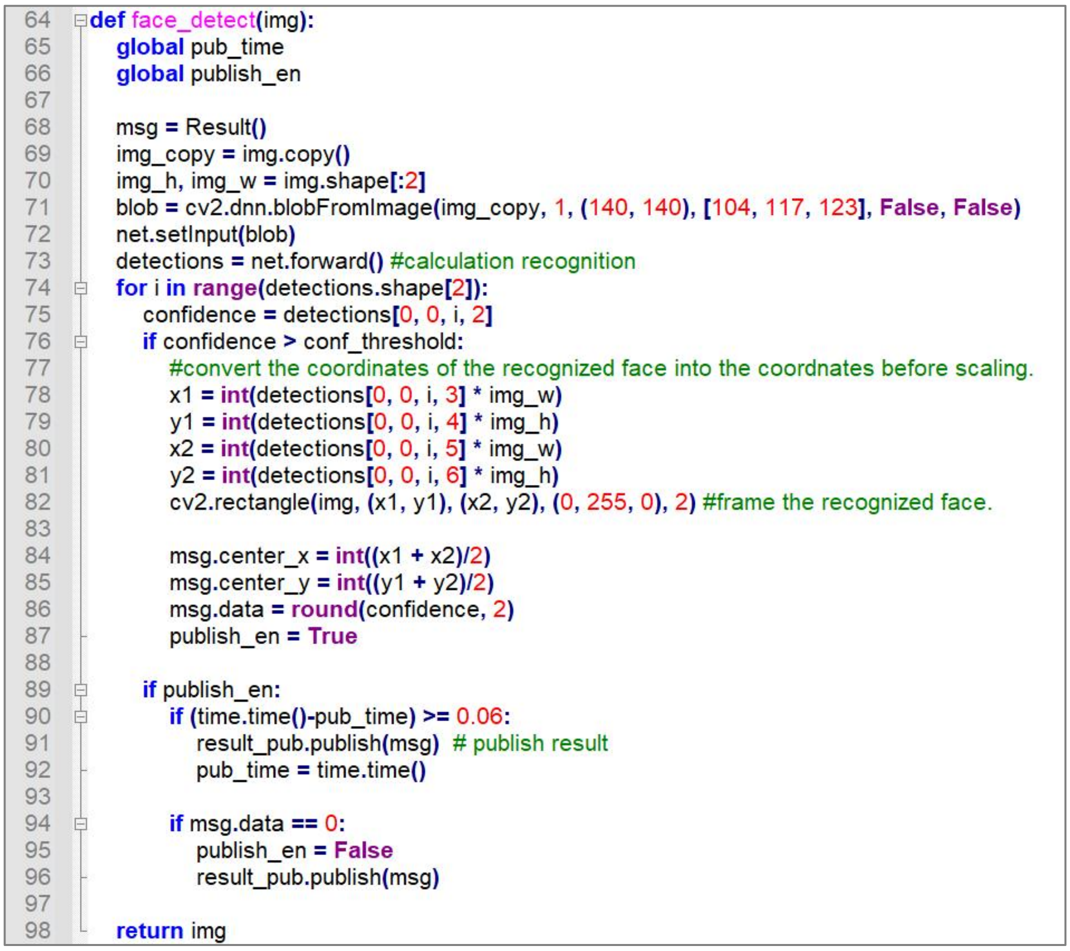

rqt_image_view__ImageView - rqt
Image View
/visual_processing/image_result
0
10.00m
ng/image_result_mouse_left
Smooth scaling
0°
Gray

4. Function Extension

4.1 Image Process

The source code of image process program is located in:

/home/ubuntu/armpi_pro/src/visual_processing/scripts/visual_processing_node.py



```python
def face_detect(img):
    global pub_time
    global publish_en

    msg = Result()
    img_copy = img.copy()
    img_h, img_w = img.shape[:2]
    blob = cv2.dnn.blobFromImage(img_copy, 1, (140, 140), [104, 117, 123], False, False)
    net.setInput(blob)
    detections = net.forward() #calculation recognition
    for i in range(detections.shape[2]):
        confidence = detections[0, 0, i, 2]
        if confidence > conf_threshold:
            #convert the coordinates of the recognized face into the coordnates before scaling.
            x1 = int(detections[0, 0, i, 3] * img_w)
            y1 = int(detections[0, 0, i, 4] * img_h)
            x2 = int(detections[0, 0, i, 5] * img_w)
            y2 = int(detections[0, 0, i, 6] * img_h)
            cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2) #frame the recognized face.

            msg.center_x = int((x1 + x2)/2)
            msg.center_y = int((y1 + y2)/2)
            msg.data = round(confidence, 2)
            publish_en = True

        if publish_en:
            if (time.time()-pub_time) >= 0.06:
                result_pub.publish(msg)  # publish result
                pub_time = time.time()

            if msg.data == 0:
                publish_en = False
                result_pub.publish(msg)

    return img
```

4.1.1 Image Preprocessing

Use the inRange () function in the cv2 library to binarize the image

```python
    blob = cv2.dnn.blobFromImage(img_copy, 1, (140, 140), [104, 117, 123
    ], False, False)
```

The first parameter “img_copy” is the input image.

The second parameter “1” is scale of the image after subtracting the mean.

The third parameter “(140, 140)” is the spatial size of the output size. The parameters means that the width is 150 and the height is 150.

The fourth parameter "[104, 117, 123]" is the subtraction value of each channel. The image channel order of OpenCV is B, G, R. The value here represents the value of the B channel minus 104, the value of the G channel minus 117, and the value of the R channel minus 123;

The fifth parameter "False" is used to decide whether to exchange the R and B channels. The default is "False", that is, the R and B channels are not exchanged. When the order of subtracting the mean value is assumed to be R, G, B, the R and B channels need to be exchanged, that is, fill in "True".

The sixth parameter “False” is used to determine whether to crop the image. The default is “False”, that is, the image is not cropped, and its size is directly adjusted, and the aspect ratio is preserved. When the value is “True”, the image is first scaled proportionally, and then cropped from the center according to the size set by the third parameter.

4.1.2 Coordinate Conversion

During pre-processing, the image is scaled and the coordinates of the face obtained are not matched with the actual screen. Therefore, the coordinates need to be converted after the image pre-processing is completed.

```
            x1 = int(detections[0, 0, i, 3] * img_w)
            y1 = int(detections[0, 0, i, 4] * img_h)
            x2 = int(detections[0, 0, i, 5] * img_w)
            y2 = int(detections[0, 0, i, 6] * img_h)
```

4.1.3 Information Feedback

Frame the face in returned image with rectangle by calling rectangle function in cv2 library.

```
            cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2)
            #frame the recognized face.
```

The meaning of parameters in parentheses are as follow:

The first parameter “img” is the input image.

The second parameter “(x1, y1)” is the starting coordinate of rectangle.

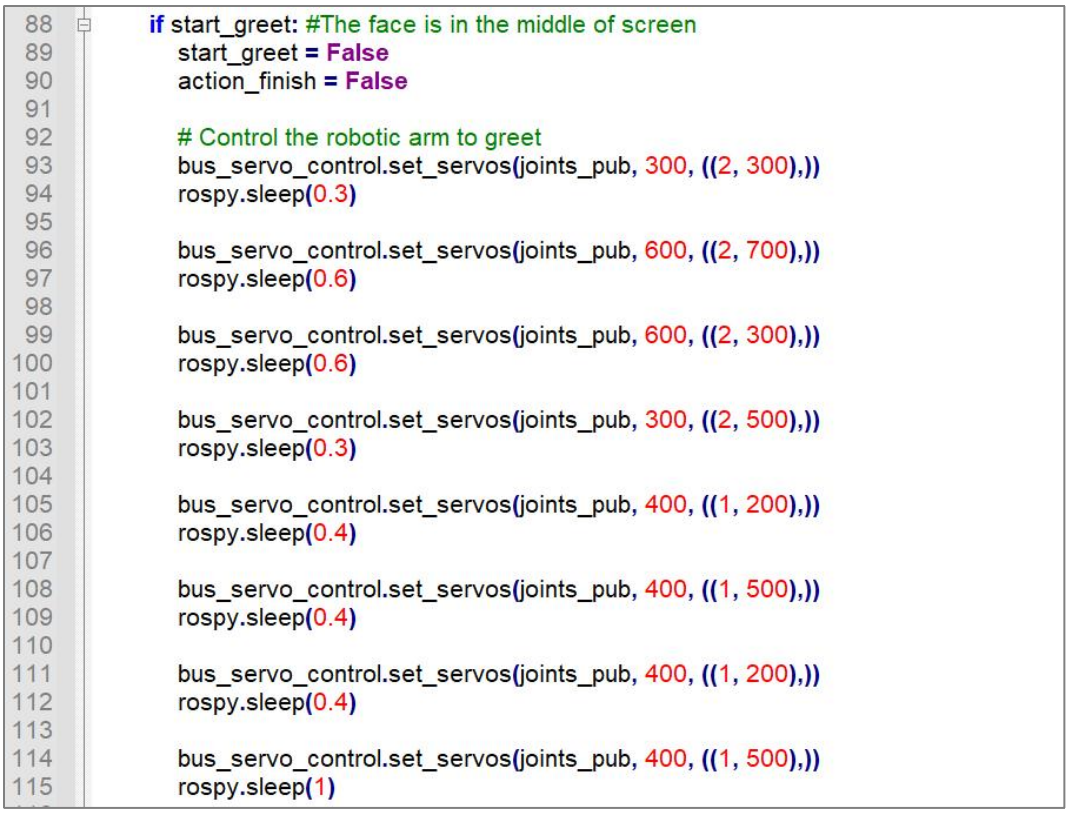
The third parameter “(x2, y2)” is the end coordinate of rectangle.

The fourth parameter “(0, 255, 0)” is the color of the edge of rectangle. Its order is B, G and R. Here is green.

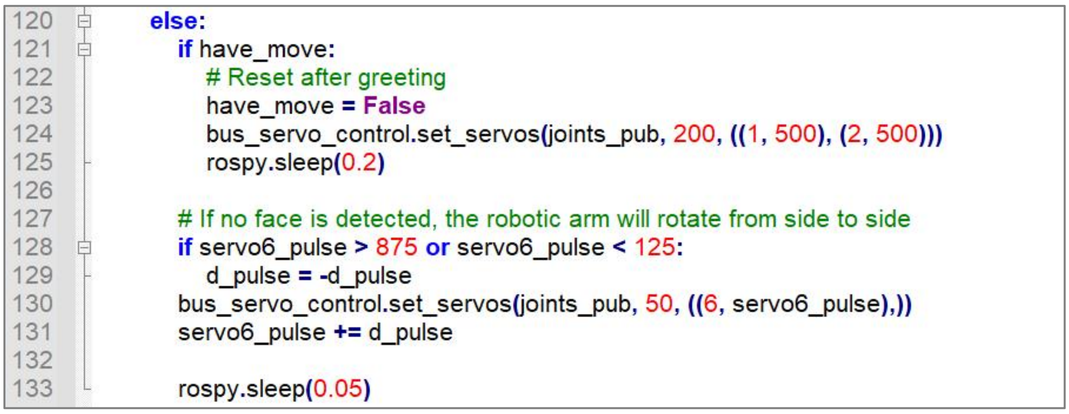
The fifth parameter “2” is the width of rectangle edge. When the value is “-1”, it means the rectangle is filled with the color specified by the fourth parameter.

4.2 Control Action

When detecting human face, ArmPi Pro is controller to perform the corresponding action by calling bus_servo_control.set_servos() function in hiwonder_servo_msgs.msg library.



```
        if start_greet: #The face is in the middle of screen
            start_greet = False
            action_finish = False

            # Control the robotic arm to greet
            bus_servo_control.set_servos(joints_pub, 300, ((2, 300),))
            rospy.sleep(0.3)

            bus_servo_control.set_servos(joints_pub, 600, ((2, 700),))
            rospy.sleep(0.6)

            bus_servo_control.set_servos(joints_pub, 600, ((2, 300),))
            rospy.sleep(0.6)

            bus_servo_control.set_servos(joints_pub, 300, ((2, 500),))
            rospy.sleep(0.3)

            bus_servo_control.set_servos(joints_pub, 400, ((1, 200),))
            rospy.sleep(0.4)

            bus_servo_control.set_servos(joints_pub, 400, ((1, 500),))
            rospy.sleep(0.4)

            bus_servo_control.set_servos(joints_pub, 400, ((1, 200),))
            rospy.sleep(0.4)

            bus_servo_control.set_servos(joints_pub, 400, ((1, 500),))
            rospy.sleep(1)
```



```
        else:
            if have_move:
                # Reset after greeting
                have_move = False
                bus_servo_control.set_servos(joints_pub, 200, ((1, 500), (2, 500)))
                rospy.sleep(0.2)

            # If no face is detected, the robotic arm will rotate from side to side
            if servo6_pulse > 875 or servo6_pulse < 125:
                d_pulse = -d_pulse
            bus_servo_control.set_servos(joints_pub, 50, ((6, servo6_pulse),))
            servo6_pulse += d_pulse

            rospy.sleep(0.05)
```

Take the code “bus_servo_control.set_servos(joints_pub, 300, ((2, 300),))” as example. The meaning of parameters in parentheses are is follow:

The first parameter “**(joints_pub)**” is to publish the message of servo control node.

The second parameter “**300**” is the running time.

The third parameter is “**((2, 300),)**”. Among them, “2” is the servo number and “300” is the servo angle.