Lesson 4 Smart Stacking

1. Working Principle

The process of whole game includes three parts: recognize, grip, stack.

Firstly, recognize the block tag within vision range.

Next, through positioning, image segmenting, contour search and other processing, the tag contour is found. Then, the quadrilateral is detected,and the straight line is fitted to form a closed loop by acquiring the four corner points.

Code and decode the tag detected to get corresponding tag ID number.

Determine the gripping sequence by comparing ID number: small ID number will be gripped first.

Finally, pick and stack the block at stacking area. After stacking, the robotic arm will return to the initial position.

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

```
148                 else:
149                     move_time = 20
150                 # 机械臂追踪移动到木块上方
151                 target = ik.setPitchRanges((0, round(y_dis, 4), 0.0), -180
                    , -180, 0)
152                 if target:
153                     servo_data = target[1]
154                     bus_servo_control.set_servos(joints_pub, move_time,((3
                        , servo_data['servo3']),(4, servo_data['servo4']),
155                                                                          (5,
                    servo_data['servo5']), (6, x_dis)))
156                     rospy.sleep(move_time/1000)
157
158                 if dx < 3 and dy < 0.003 and not stack_en: #
                    等待机械臂稳定停在木块上方
159                     count_ += 1
160                     if count_ == 10:
161                         count_ = 0
162                         stack_en = True
163                         angle = object_angle % 90
164                         print(angle)
165                         offset_y = Misc.map(target[2], -180, -150, -0.01,
                            0.02) # 设置位置补偿
166                 else:
167                     count_ = 0
168
169                 if stack_en:
```

2. Operation Steps

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

2.1 Preparation

Prepare three tag blocks (tag 1, tag 2, tag3) and place them within the detected range, The distance between two blocks can not smaller than 3cm.

2.2 Enter Game

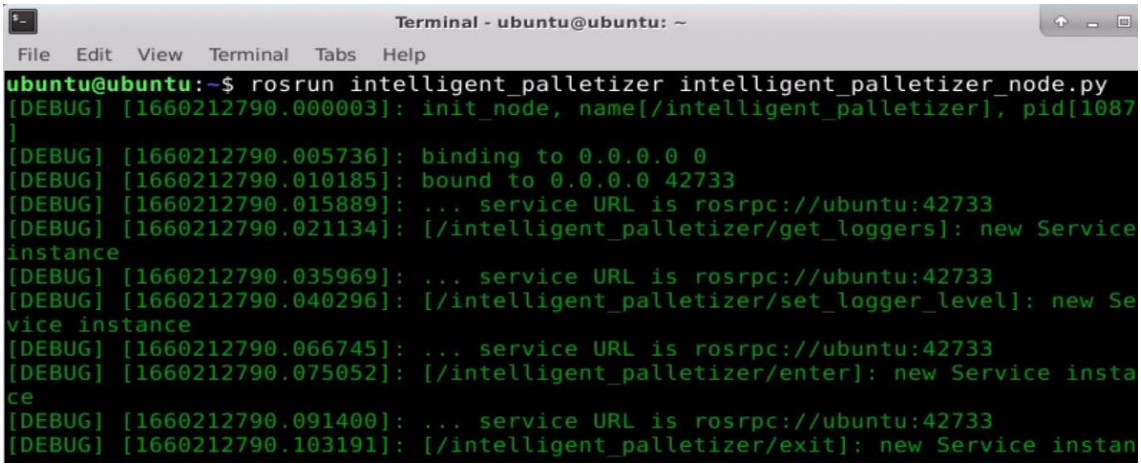
1) Turn on ArmPi Pro and connect to the system desktop via No Machine.

2) Click 

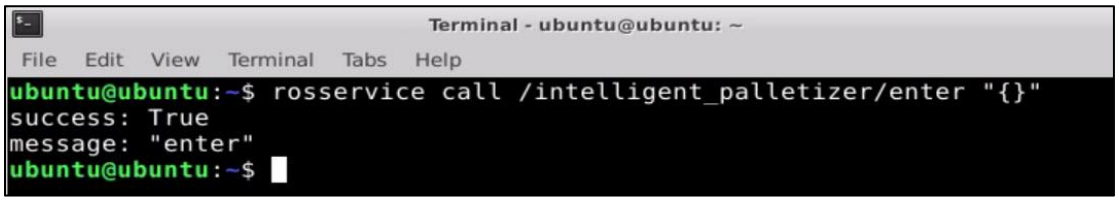
and select 

in pop-up interface to open the terminal.

3) Enter command “**rosrun intelligent_palletizer intelligent_palletizer_node.py**” to tun smart stacking program.



4) Do not close the opened terminal and open a new terminal. Then enter command “**rosservice call /intelligent_transport/enter "{}"**” to enter and press “Enter” to enter this game. After entering, the terminal will print the prompt as the figure shown below:



2.3 Start image transmission

2.3.1 Start with browser

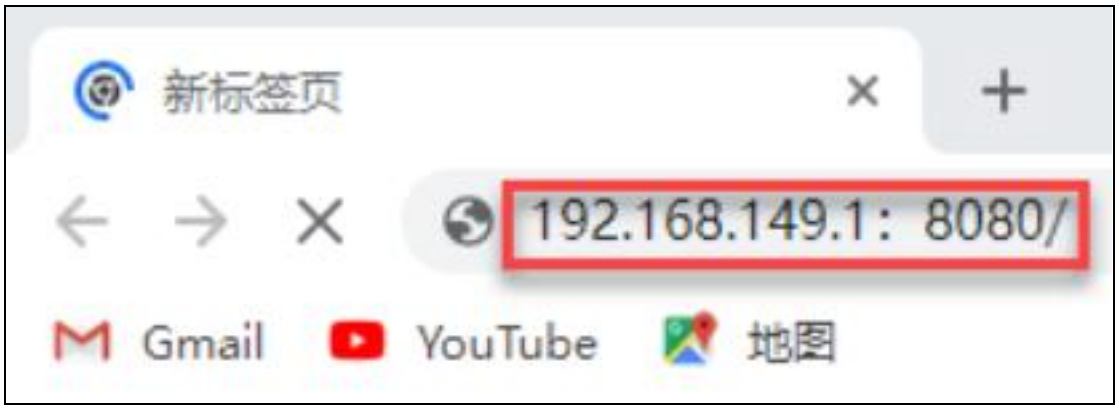
To avoid consuming too much running memory of Raspberry Pi. It is recommended to use an external browser to start image transmission.

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.

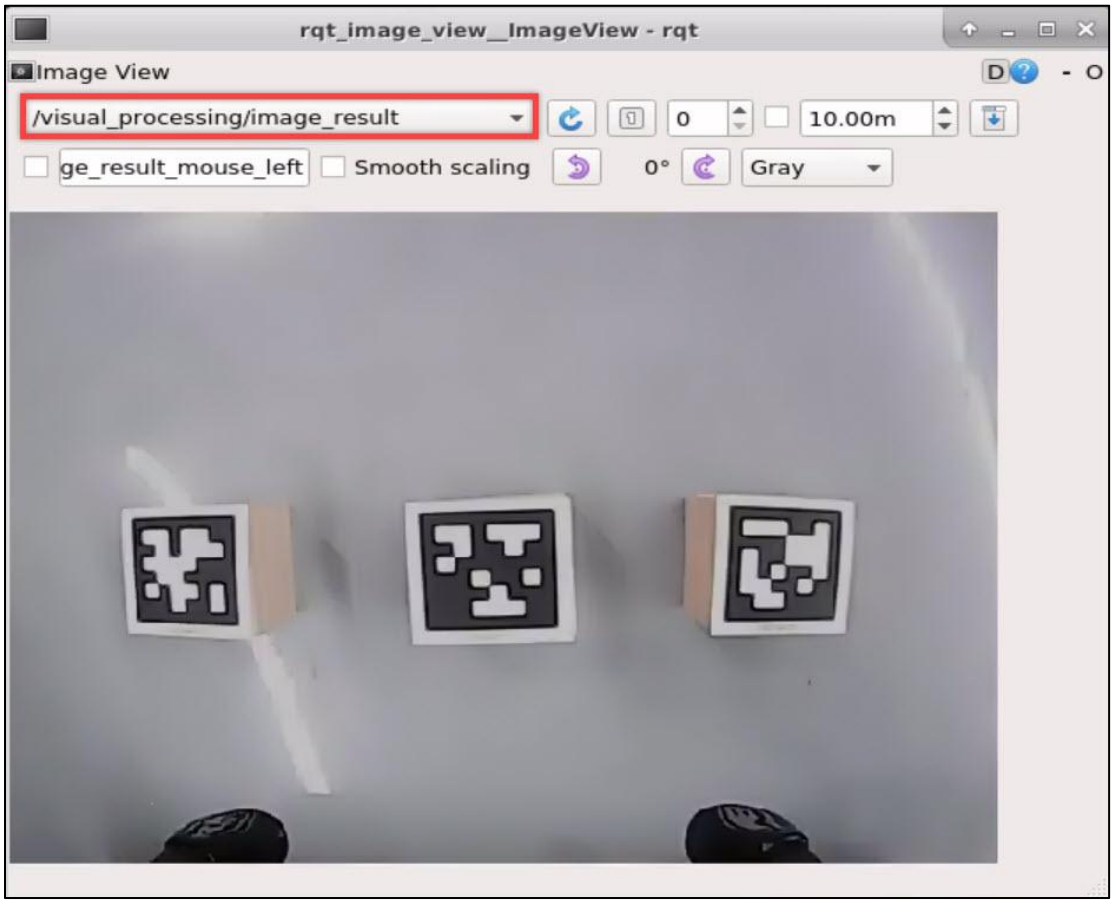
2.3.2 Start with rqt

1) After completing the steps of “2.2 Enter Game” and do not exit the terminal, open a new terminal.

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

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, as the figure shown below:

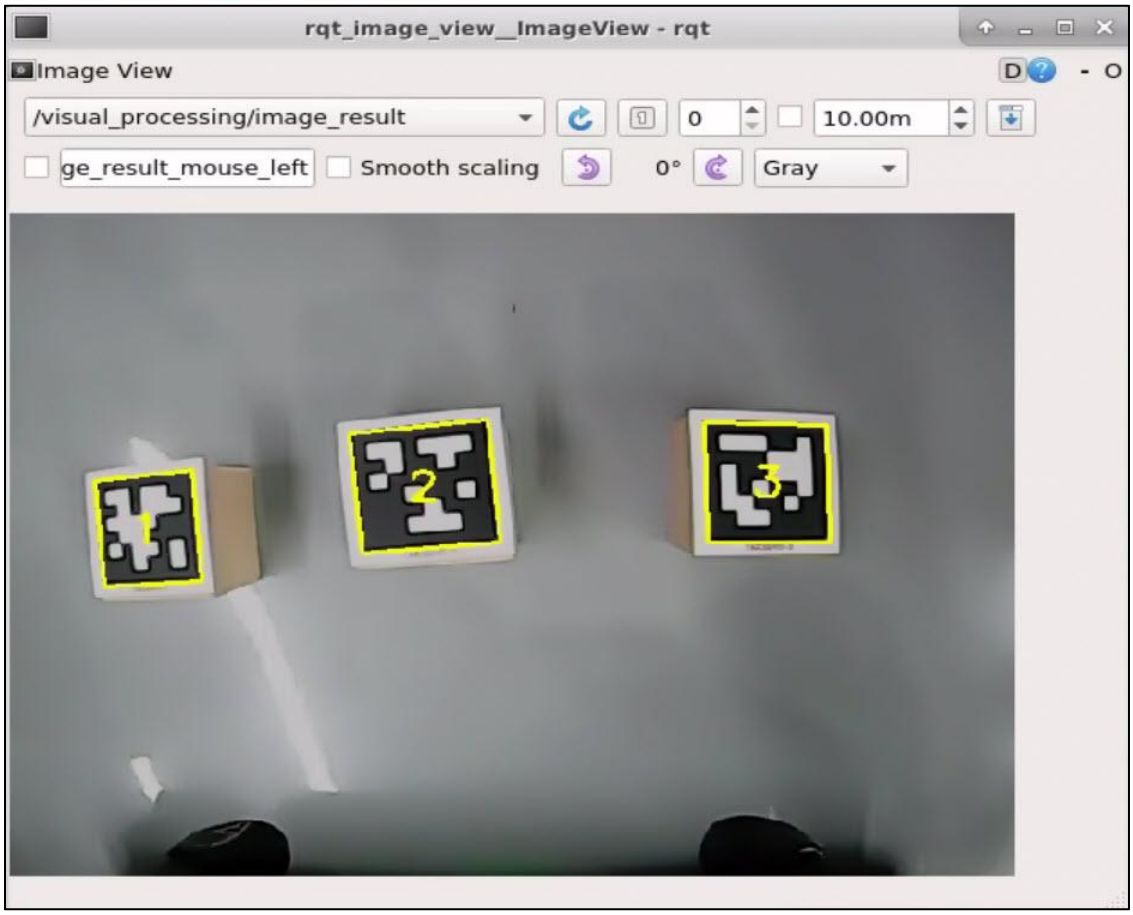


Note: After opening image, the topic option must be selected. Otherwise, after starting game, the recognition process can not be displayed normally.

2.4 Start Game

Now, enter the terminal according to the steps in "2.2 Enter Game" and input command "**rosservice call /intelligent_palletizer/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 /intelligent_palletizer/set_running "data: true"
success: True
message: "set_running"
ubuntu@ubuntu:~$
```



2.5 Stop and Exit

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

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

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

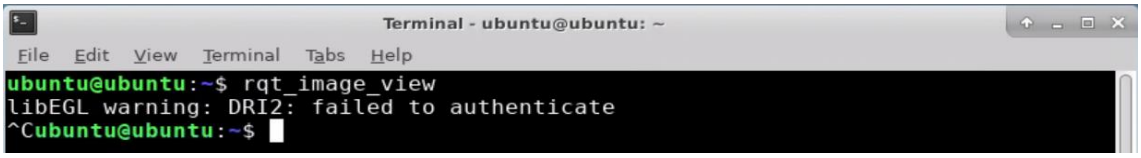
```
ubuntu@ubuntu:~$ rosservice call /intelligent_palletizer/exit "{}"
success: True
message: "exit"
ubuntu@ubuntu:~$
```

3) To avoid consume too much running memory of Raspberry Pi, after exiting the game and returning to the terminal of running game programmings, press "Ctrl+C" to exit the program. If fail to exit, please keep trying several times.

```
[INFO] [1660267868.468256]: enter intelligent palletizer
[INFO] [1660267868.474775]: intelligent palletizer Init
[DEBUG] [1660267870.491558]: connecting to ubuntu 39075
[DEBUG] [1660267870.577300]: connecting to ubuntu 39075
[INFO] [1660267933.886385]: start running intelligent palletizer
[DEBUG] [1660267933.902881]: connecting to ubuntu 39075
[INFO] [1660269439.282748]: stop running intelligent palletizer
[DEBUG] [1660269439.335897]: connecting to ubuntu 39075
[INFO] [1660270162.443987]: exit intelligent palletizer
[DEBUG] [1660270162.477862]: connecting to ubuntu 39075
[DEBUG] [1660270162.533879]: [/visual_processing/result] failed to receive incom
ing message : unable to receive data from sender, check sender's logs for detail
s
^C[INFO] [1660270222.048972]: shutdown
ubuntu@ubuntu:~$
```

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.

4) If want to close 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 game, ArmPi Pro will recognize the block tag within the detected range. Then the robotic arm will grip and stack the block at the tacking area.

4. Code Analyse

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**

```
# 检测apriltag函数
detector = apriltag.Detector(searchpath=apriltag._get_demo_searchpath())
def apriltag_Detect(img):
    global pub_time
    global publish_en
    global id_smallest

    msg = Result()
    img_copy = img.copy()
    img_h, img_w = img.shape[:2]
    frame_resize = cv2.resize(img_copy, size_m, interpolation=cv2.
    INTER_NEAREST)
    gray = cv2.cvtColor(frame_resize, cv2.COLOR_BGR2GRAY)
    detections = detector.detect(gray, return_image=False)

    if len(detections) != 0:
        for i, detection in enumerate(detections):
            tag_id = int(detection.tag_id)          # 获取tag_id
            corners = np.rint(detection.corners)  # 获取四个角点
            for i in range(4):
                corners[i][0] = int(Misc.map(corners[i][0], 0, size_m[0],
```

4.1.1 Obtain Tag ID

Firstly, get tag ID through int（）function

```
        for i, detection in enumerate(detections):
            tag_id = int(detection.tag_id)          # 获取tag_id
```

4.1.2 Obtain the information of corner point.

Obtain the four corner points through np.rint().

```
            corners = np.rint(detection.corners)  # 获取四个角点
            for i in range(4):
                corners[i][0] = int(Misc.map(corners[i][0], 0, size_m[0],
                0, img_w))
                corners[i][1] = int(Misc.map(corners[i][1], 0, size_m[1],
                0, img_h))
```

4.1.3 Detect Tag

1) After getting the corner point information of tag, recognize tag by calling drawContours() function in cv2 library.

```
            cv2.drawContours(img, [np.array(corners, np.int)], -1, (0, 255
            , 255), 2)
```

The meaning of parameter in parentheses is as follow:

The first parameter “img” is the input image.

The second parameter “**[np.array(corners, np.int)]**” is the contour which is list in

Python.

The third parameter “-1” is the index of contour. The value here represents all contour in drawing contour list.

The fourth parameter “**(0, 255, 255)**” is the contour color. The value sequence is B, G, and R. The color here is yellow.

The fifth parameter “2” is the width of contour.

1) By calling putText() function in cv2 library, print the tag ID and type in the transmitted image.

```
127                 cv2.putText(img, str(tag_id), (object_center_x - 10,
                    object_center_y + 10), cv2.FONT_HERSHEY_SIMPLEX, 1, [0, 255,
                    255], 2)
```

The meaning of parameters in parentheses is as follow:

The first parameter “img” is the input image.

The second parameter “**str(tag_id)**” is the displayed content.

The third parameter “**(object_center_x - 10, object_center_y + 10)**” is the display position.

The fourth parameter “**cv2.FONT_HERSHEY_SIMPLEX**” is the font type.

The fifth parameter “**1**” is the size of font.

The sixth parameter “**[0, 255, 255]**” is the font color and its sequence is B, G, R. The value here is yellow.

The seventh parameter “**2**” is the thickness of font.

4.2 Control Action

After the image is processed, get the position of tag black through inverse kinematics. Then grip and tack the blocks,

The source code of program is located in:

/home/ubuntu/armpi_pro/src/intelligent_palletizer/scripts/intelligent_palletizer_node.py

```
# 初始位置
def initMove(delay=True):
    with lock:
        target = ik.setPitchRanges((0, 0.15, 0.0), -180, -180, 0) # 逆运动学求解
        if target:
            servo_data = target[1]
            bus_servo_control.set_servos(joints_pub, 1800, ((1, 200), (2, 500), (3, servo_data['servo3']), (4, servo_data['servo4']), (5, servo_data['servo5']),(6, servo_data['servo6'])))
    if delay:
        rospy.sleep(2)

def turn_off_rgb():
    led = Led()
    led.index = 0
    led.rgb.r = 0
    led.rgb.g = 0
    led.rgb.b = 0
    rgb_pub.publish(led)
```

1) By determining whether the coordinate of tag block is change, then we can determine if the black is stable. If it meets the conditions, robotic arm will grip the block.

```
    while __isRunning:
        if steadier and object_center_x > 0 and object_center_y > 0:
            # 木块已经放稳，进行追踪夹取
```

2) Robotic arm moves to above the block.

```
            # 机械臂追踪移动到木块上方
            target = ik.setPitchRanges((0, round(y_dis, 4), 0.0), -180, -180, 0)
```

The analysis of code above is as follow:

The first parameter “0” is the position in x-axis.

The second parameter “**round(y_dis, 4), 0.0)**” is the position in y-axis.

The third parameter “-180” is the pitch angle.

The fourth and fifth parameter “-180”, “0” is the range of pitch angle.

3) When the determination conditions are met, robotic arm will stop above the

block and adjust the angle of robotic arm.

```
158             if abs(dx) < 3 and abs(dy) < 0.003 and not stack_en: #
                等待机械臂稳定停在木块上方
159                 count_ += 1
160                 if count_ == 10:
161                     count_ = 0
162                     stack_en = True
163                     angle = object_angle % 90
164                     print(angle)
165                     offset_y = Misc.map(target[2], -180, -150, -0.01, 0.02
                        ) # 设置位置补偿
```

4) Then the robotic arm is controlled to grip and raise the block through bus_servo_control.set_servos（）function.

```
181                 bus_servo_control.set_servos(joints_pub, 500, ((1, 450),))
                     # 闭合机械爪
182                 rospy.sleep(0.8)
183
184                 bus_servo_control.set_servos(joints_pub, 1500, ((1, 450),
                    (2, 500), (3, 80), (4, 825), (5, 625), (6, 500))) #
                    机械臂抬起来
185                 rospy.sleep(1.5)
```

Then the inverse kinematics controls the robotic arm to transport and put down the block.

```
198                 target = ik.setPitchRanges(place_coord[stack_num], -180, -
                    180, 0) # 机械臂移动到色块放置位置
199                 if target:
200                     servo_data = target[1]
201                     bus_servo_control.set_servos(joints_pub, 1000, ((3,
                        servo_data['servo3']), (4, servo_data['servo4']), (5,
                        servo_data['servo5']))) # 再放下了
202                 rospy.sleep(1)
```

Take “target = ik.setPitchRanges(place_coord[stack_num], -180, -180, 0)” as example. Among them, the first parameter “place_coord[stack_num]” represent s the coordinate position of tag block. The following image is the position information of corresponding ID.

```
113 place_coord = {1:(0.18, 0.0, -0.09),
114                2:(0.18, 0.0, -0.05),
115                3:(0.18, 0.0, -0.02)}
```

5) Control each servo by bus_servo_control.set_servos（） and let gripper put down and release the block.

```
                if target:
                    servo_data = target[1]
                    bus_servo_control.set_servos(joints_pub, 1000, ((3,
                    servo_data['servo3']), (4, servo_data['servo4']), (5,
                    servo_data['servo5']))) # 再放下了
                rospy.sleep(1)

                bus_servo_control.set_servos(joints_pub, 500, ((1, 150),))
                  #张开机械爪
                rospy.sleep(0.8)
```

6) When stacking action is executed three time, it will starts from scratch.

```
                if stack_num >= 3:
                    stack_num = 0
```

7) Finally, robotic arm returns to the initial posture.

```
                #机械臂复位
                target = ik.setPitchRanges((0, 0.15, 0.0), -180, -180, 0)
                if target:
                    servo_data = target[1]
                    bus_servo_control.set_servos(joints_pub, 1000, ((1,
                    200), (2, 500), (3, servo_data['servo3']),
                                                                    (4,
servo_data['servo4']), (5, servo_data['servo5'])))
                    rospy.sleep(1)
                    bus_servo_control.set_servos(joints_pub, 1500, ((6,
                    servo_data['servo6']),))
                    rospy.sleep(1.5)

                start_en = True
                reset()  # 变量重置
```