Lesson 1 Intelligent Picking

1. Working Principle

Firstly, recognize the color and convert the object color through Lab color. Then, frame the target object with circle after processing the object image.

After recognizing, robotic arm will pick according to the position of block and place it to the specified position.

The source code of program is located in:

/home/ubuntu/armpi_pro/src/intelligent_grasp/scripts/intelligent_grasp_node.py

```
lock = RLock()
ik = ik_transform.ArmIK()

set_visual = 'line'
detect_step = 'color' # 步骤：巡线或者检测色块
line_color = 'yellow' # 巡线颜色
stable = False       # 色块夹取判断变量
place_en = False     # 色块放置判断变量
position_en = False  # 色块夹取前定位判断变量
__isRunning = False  # 玩法控制开关变量
block_clamp = False  # 搬运色块标记变量
chassis_move = False # 底盘移动标记变量

x_dis = 500
y_dis = 0.15
line_width = 0
line_center_x = 0
line_center_y = 0
color_centreX = 320
color_centreY = 410
color_center_x = 0
color_center_y = 0
detect_color = 'None'
target_color = 'None'

img_h, img_w = 480, 640

line_x_pid = PID.PID(P=0.002, I=0.001, D=0)  # pid初始化
color_x_pid = PID.PID(P=0.06, I=0, D=0)
color_y_pid = PID.PID(P=0.00003, I=0, D=0)

range_rgb = {
    'red': (0, 0, 255),
    'blue': (255, 0, 0),
    'green': (0, 255, 0),
    'black': (0, 0, 0),
    'yellow': (0, 255, 255),
    'white': (255, 255, 255),
}
```

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

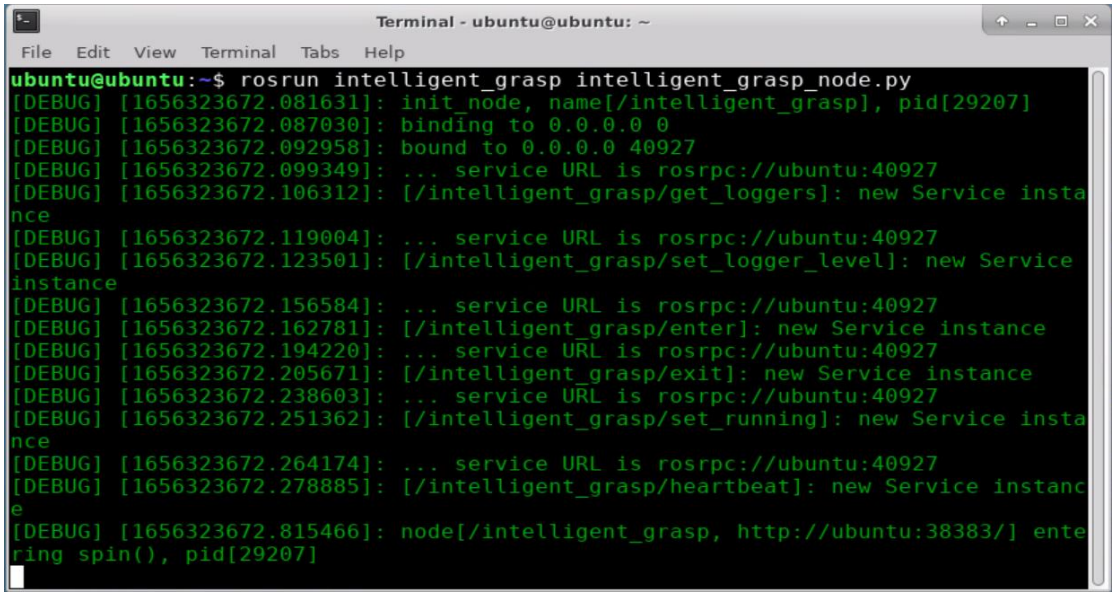
1) Turn on ArmPi Pro and connect to the system desktop via No Machine.

2) Click 

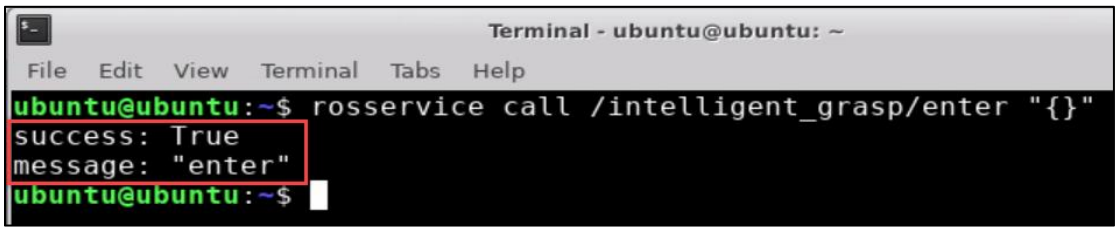
and select 

in pop-up interface to open the terminal.

3) Enter command “**rosrun intelligent_grasp intelligent_grasp_node.py**” and press “Enter” to run the program of intelligent picking.



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



2.2 Start image transmission

2.2.1 Start with browser

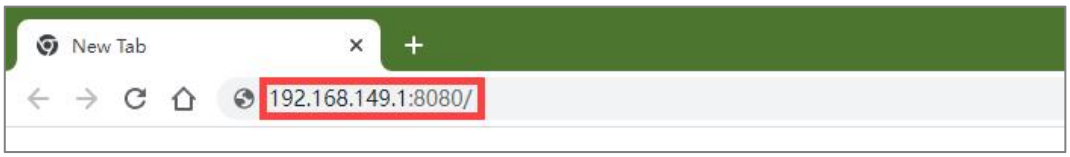
To avoid consuming too much running memory of Raspberry Pi. It is recommended to use an external browser to open the transmitted image.

The specific steps are as follows:

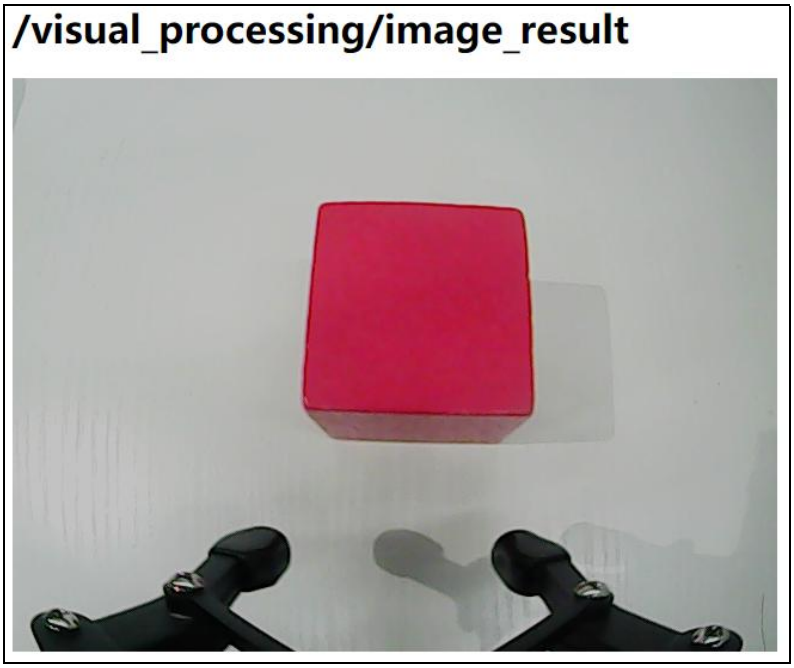
1) Select a browser. Take Google Chrome as example.

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/” for example, “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.



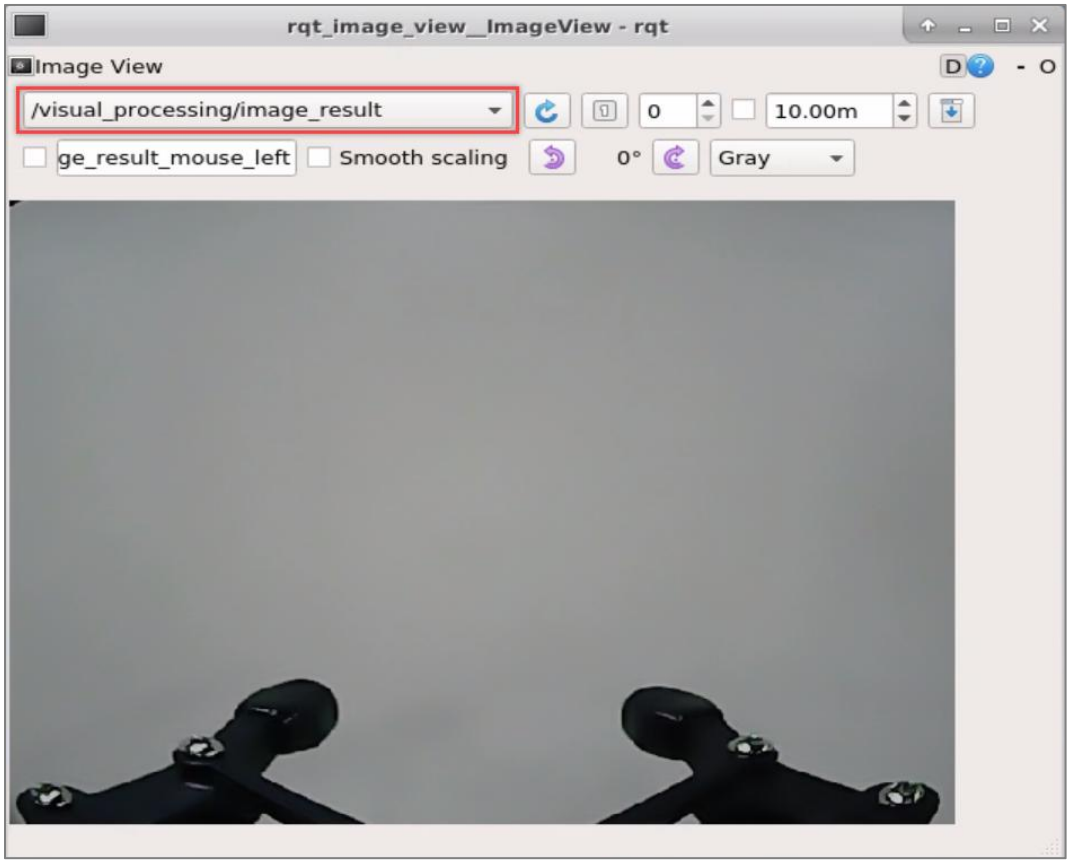
2.2.2 Start with rqt

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.



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 /visual_patrol/set_running "data: true"”. Then if the prompt shown in the following red box appears, which means game has been started successfully.

2.4 Stop and Exit

1) If want to stop the game, enter command “**rosservice call /intelligent_grasp/set_running**” to exit.

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

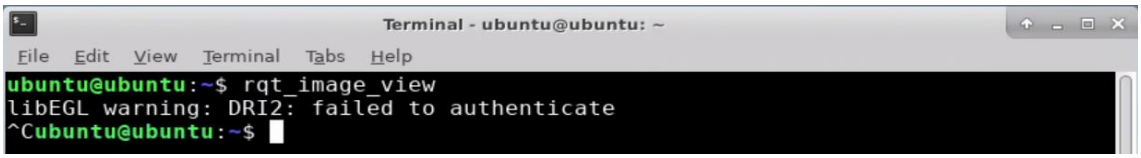
2) If want to exit the game, enter command "**rosservice call /intelligent_grasp/exit** "{}"" to exit.

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

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.

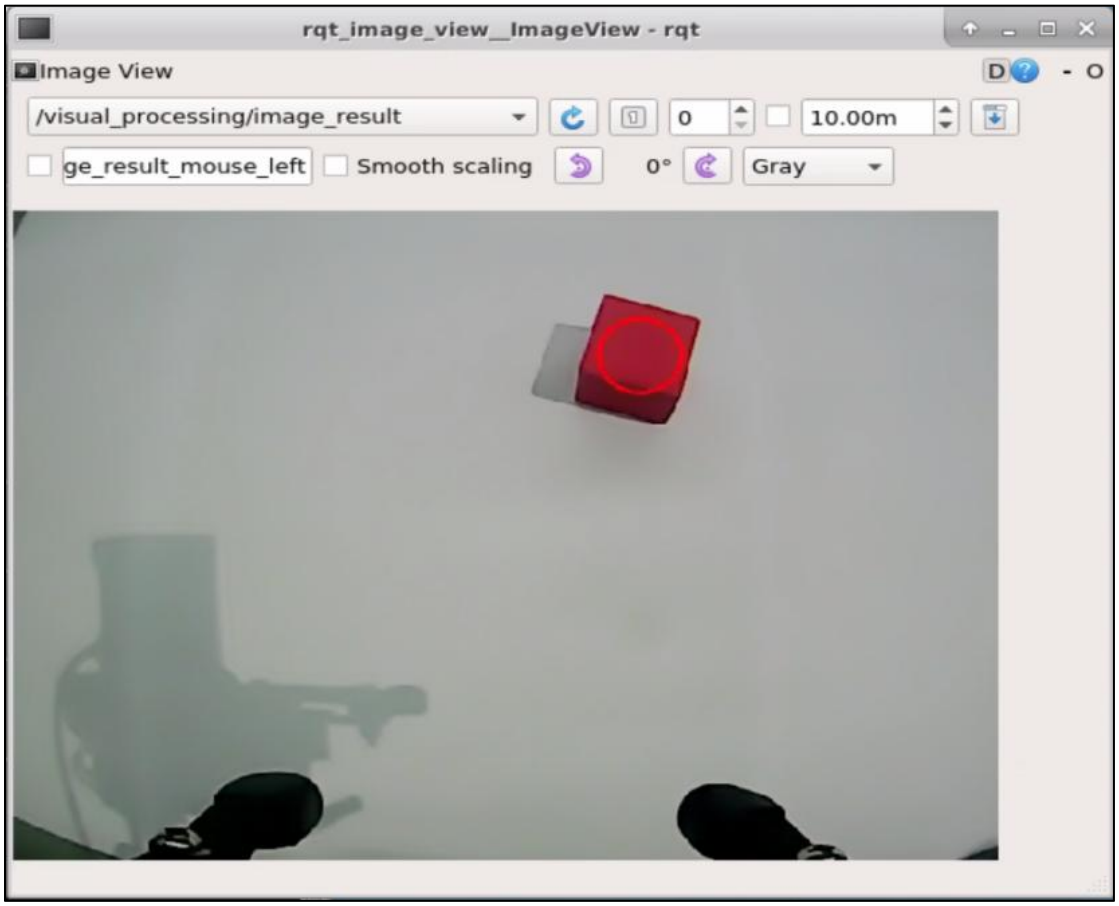
Note: Before exiting the game, it will keep running when Raspberry Pi is powered on. To avoid consume too much running memory of Raspberry Pi, you need to exit the game first according to the operation steps above before performing other AI vision games.

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 the game, the robotic arm will Rotate to search the block. We can see that that target block is framed in rqt tool after recognition . Then the robotic arm will slowly move to the block, grip and place it to the specified position.



4. Program Parameter Instruction

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

```
def colors_detect(img, color_list):
    global pub_time
    global publish_en
    global color_range_list

    if color_list == 'RGB' or color_list == 'rgb':
        color_list = ('red','green','blue')
    else:
        return img

    msg = Result()
    msg.data = 0
    color_num = 0
    max_area = 0
    color_area_max = None
    areaMaxContour_max = 0

    img_copy = img.copy()
    img_h, img_w = img.shape[:2]
    frame_resize = cv2.resize(img_copy, size_m, interpolation=cv2.INTER_NEAREST)
    frame_lab = cv2.cvtColor(frame_resize, cv2.COLOR_BGR2LAB)  # convert image into LAB space

    for color in color_list:
        if color in color_range_list:
            color_range = color_range_list[color]
            frame_mask = cv2.inRange(frame_lab, tuple(color_range['min']), tuple(color_range['max']))  # Bitwise operation
            operates on the original image and mask.
            eroded = cv2.erode(frame_mask, cv2.getStructuringElement(cv2.MORPH_RECT, (2, 2)))      # erode
            dilated = cv2.dilate(eroded, cv2.getStructuringElement(cv2.MORPH_RECT, (2, 2)))        # dilate
            contours = cv2.findContours(dilated, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)[-2]  # find contour
            areaMaxContour, area_max = getAreaMaxContour(contours)                                 # find the biggest contour
            if areaMaxContour is not None:
                if area_max > max_area:#find the biggest area
                    max_area = area_max
                    color_area_max = color
                    areaMaxContour_max = areaMaxContour
```

4.1.1Binarization

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

```
297         frame_mask = cv2.inRange(frame_lab, tuple(color_range['min']), tuple(color_range['max']))
            # Bitwise operation operates on the original image and mask.
```

The first parameter “frame_lab” is the input image.

The second parameter “**tuple(color_range['min'])**” is the lower limit of threshold.

The third parameter “**tuple(color_range['max'])**” is the upper lower of threshold.

4.1.2 Dilation and Erosion

To lower interference and make image smoother, the image needs to be dilated and eroded.

```
298         eroded = cv2.erode(frame_mask, cv2.getStructuringElement(cv2.MORPH_RECT, (2, 2))) # erode
299         dilated = cv2.dilate(eroded, cv2.getStructuringElement(cv2.MORPH_RECT, (2, 2)))        # dilate
```

erode() function is applied to erode image. Take code “**eroded = cv2.erode(frame_mask, cv2.getStructuringElement(cv2.MORPH_RECT, (2, 2)))**” as example. The meaning of parameters in parentheses are as follow:

The first parameter "**frame_mask**" is the input image.

The second parameter "**cv2.getStructuringElement(cv2.MORPH_RECT, (2, 2))**" is the structural elements and kernel that determines the nature of operation. The first parameter in parentheses is the shape of kernel and the second parameter is the size of kernel.

dilate() function is applied to dilate image. The meaning of parameters in parentheses is the same as the parameters of erode() function.

4.1.3 Obtain the contour of the maximum area

After processing the above image, obtain the contour of the recognition target. The findContours() function in cv2 library is involved in this process.

```
300         contours = cv2.findContours(dilated, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)[-2]
            # find contour
```

The erode() function is applied to erode. Take code "**contours = cv2.findContours(dilated, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)[-2]**" as example.

The first parameter "**dilated**" is the input image.

The second parameter "**cv2.RETR_EXTERNAL**" is the contour retrieval mode.

The third parameter "**cv2.CHAIN_APPROX_NONE)[-2]**" is the approximate method of contour.

Find the maximum contour from the obtained contours. To avoid interference, set a minimum value. Only when the area is greater than this minimum value, the target contour will take effect. The minimum value here is "50".

```
301         areaMaxContour, area_max = getAreaMaxContour(contours)  # find the biggest contour
302         if areaMaxContour is not None:
303             if area_max > max_area:#find the biggest area
```

4.1.4 Obtain Position Information

The minAreaRect() function in cv2 library is used to obtain the minimum

external rectangle of the target contour, and the coordinates of its four vertices are obtained through the boxPoints() function. Then, the coordinates of the center point of the rectangle can be calculated from the coordinates of the vertexes of the rectangle.

```
309         (centerx, centery), radius = cv2.minEnclosingCircle(areaMaxContour_max)  # Get the smallest circumcircle
310         msg.center_x = int(Misc.map(centerx, 0, size_m[0], 0, img_w))
311         msg.center_y = int(Misc.map(centery, 0, size_m[1], 0, img_h))
312         radius = int(Misc.map(radius, 0, size_m[0], 0, img_w))
313         rect = cv2.minAreaRect(areaMaxContour_max)   #Minimum circumscribed rectangle
314         box = np.int0(cv2.boxPoints(rect))                #The four vertices of the smallest circumscribed rectangle
315         msg.angle = int(math.degrees(math.atan2(box[1][1] - box[0][1], box[1][0] - box[0][0])))
316
317         cv2.circle(img, (msg.center_x, msg.center_y), radius+5, range_rgb[color_area_max], 2)
```

Determine the color block with the largest area.

```
319         if color_area_max == 'red':  #the maximum color area is red
320             msg.data = 1
321         elif color_area_max == 'green':  #the maximum color area is green
322             msg.data = 2
323         elif color_area_max == 'blue':  #the maximum color area is blue
324             msg.data = 3
```

4.2 Control Action

The position of the target on x, y and z axes are obtained after processing image. Then get the target position calculated by inverse kinematics and grip it.

```
112 while __isRunning:
113     if arm_move and detect_color != 'None': # wait for grasping
114         target_color = detect_color  # save the target color
115         set_rgb(target_color)    # set RGB light color
116         rospy.sleep(0.1)
117         buzzer_pub.publish(0.1) # The buzzer sounds once
118         bus_servo_control.set_servos(joints_pub, 500, ((1, 120),)) #open gripper
119         rospy.sleep(0.5)
120         target = ik.setPitchRanges((0, round(y_dis + offset_y, 4), -0.08), -180, -180, 0) #the robotic arm moves downward.
121         if target:
122             servo_data = target[1]
123             bus_servo_control.set_servos(joints_pub, 1000, ((3, servo_data['servo3']), (4, servo_data['servo4']),
124                                          (5, servo_data['servo5']), (6, x_dis)))
125         rospy.sleep(1.5)
126         bus_servo_control.set_servos(joints_pub, 500, ((1, 450),)) # close gripper
127         rospy.sleep(0.8)
```

The inverse kinematics takes "**ik.setPitchRanges((0, round(y_dis + offset_y, 4), -0.08), -180, -180, 0)**" as example and the meaning of parameters in parentheses are as follow:

The first parameter is "**(0, round(y_dis + offset_y, 4)**". "0" is the position of the target

on x-axis. "**round(y_dis, 4)**" is the position of the target on y-axis. "**round(z_dis, 4)**" is the position of the target on z-axis.

The second parameter "-180" is the angle of x-axis.

The third parameter "-180" is the range of the pitch angle.

The fourth parameter "0" is the range of pitch angle.

The servo control takes the code "bus_servo_control.set_servos(joints_pub, 20, ((3, servo_data['servo3']), (4, servo_data['servo4']), (5, servo_data['servo5']), (6, x_dis)))" as example and the meaning of parameters in parentheses is as follow:

The first parameter "joints_pub" is to publish the message of servo control node.

The second parameter "20" is the running time.

The third parameter is "**((3, servo_data['servo3']), (4, servo_data['servo4']), (5, servo_data['servo5']), (6, x_dis)**". Among them, "3" is the servo number. "**servo_data['servo3']**" and the rest of parameters are the servo angle.