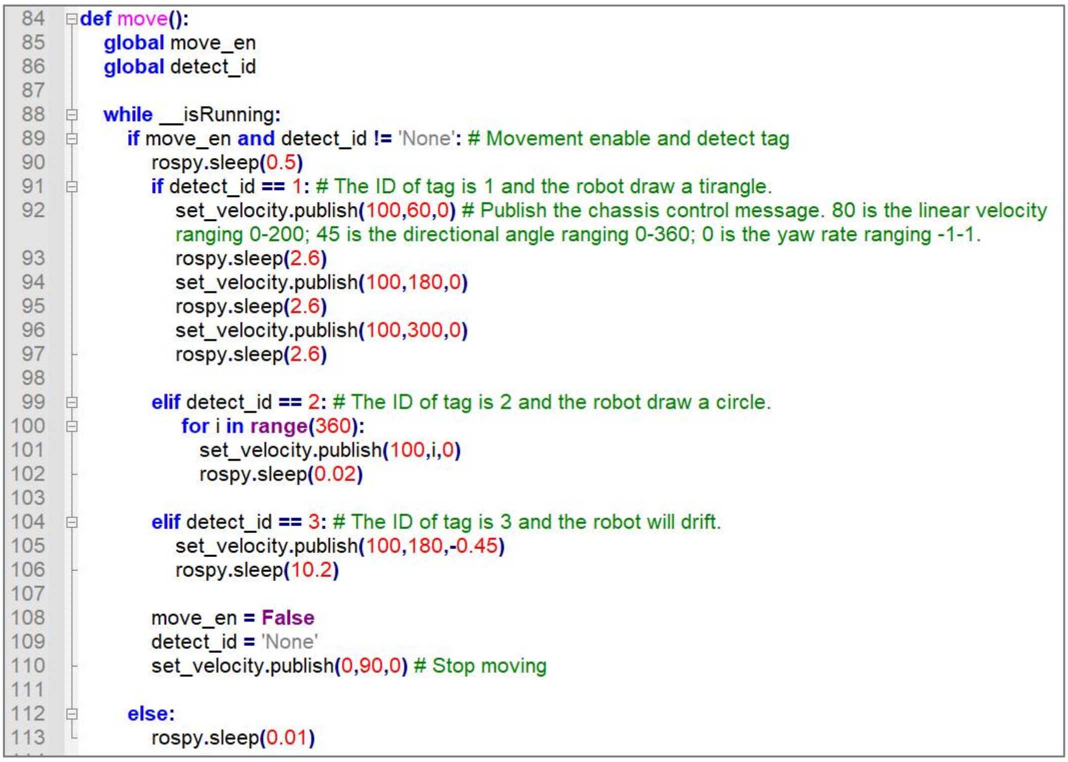
Lesson 4 Tag Recognition

1. Working Principle

AprilTag as a vision location identifier is similar to QR code or bar code, which can detect the tag and calculate its relative position quickly.

We use the trained tag. Firstly, obtain and process image. Then detect the tag and get the information. Finally, frame the recognized tag and perform the corresponding action.

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



```
def move():
    global move_en
    global detect_id

    while __isRunning:
        if move_en and detect_id != 'None': # Movement enable and detect tag
            rospy.sleep(0.5)
            if detect_id == 1: # The ID of tag is 1 and the robot draw a tirangle.
                set_velocity.publish(100,60,0) # Publish the chassis control message. 80 is the linear velocity ranging 0-200; 45 is the directional angle ranging 0-360; 0 is the yaw rate ranging -1-1.
                rospy.sleep(2.6)
                set_velocity.publish(100,180,0)
                rospy.sleep(2.6)
                set_velocity.publish(100,300,0)
                rospy.sleep(2.6)

            elif detect_id == 2: # The ID of tag is 2 and the robot draw a circle.
                for i in range(360):
                    set_velocity.publish(100,i,0)
                    rospy.sleep(0.02)

            elif detect_id == 3: # The ID of tag is 3 and the robot will drift.
                set_velocity.publish(100,180,-0.45)
                rospy.sleep(10.2)

            move_en = False
            detect_id = 'None'
            set_velocity.publish(0,90,0) # Stop moving

        else:
            rospy.sleep(0.01)
```

2. Operation Steps

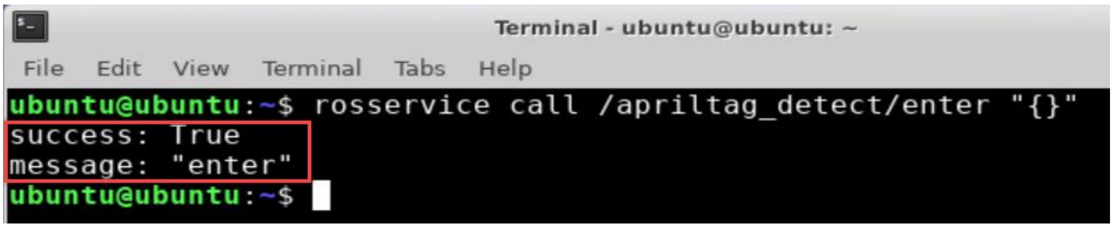
ℹ It should be case sensitive when entering command and the "Tab" key can be used to complete the keywords.

2.1 Enter Game

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

2) Click Applications and select Terminal Emulator in pop-up interface to open the terminal.

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



Terminal - ubuntu@ubuntu: ~
File Edit View Terminal Tabs Help

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

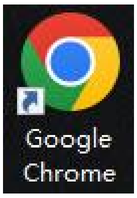
2.2 Start image transmission

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.

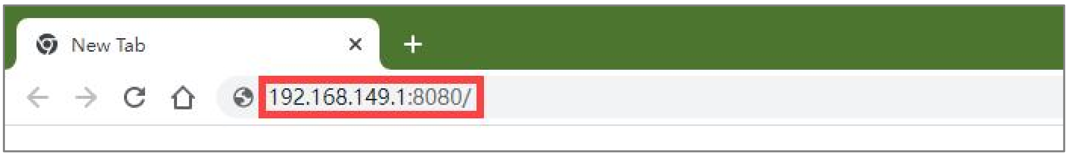
The specific steps are as follows:

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



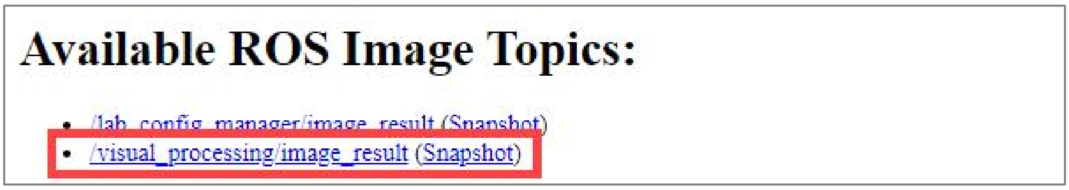
2) Then enter the default IP address “192.168.149.1:8080/” (Note: this IP address is the default IP address for direction connection mode. If it is LAN mode, please enter “Device IP address+：8080/” for example, “192.168.149.1:8080/”) If fail to open, you can try it several times or restart camera.

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.



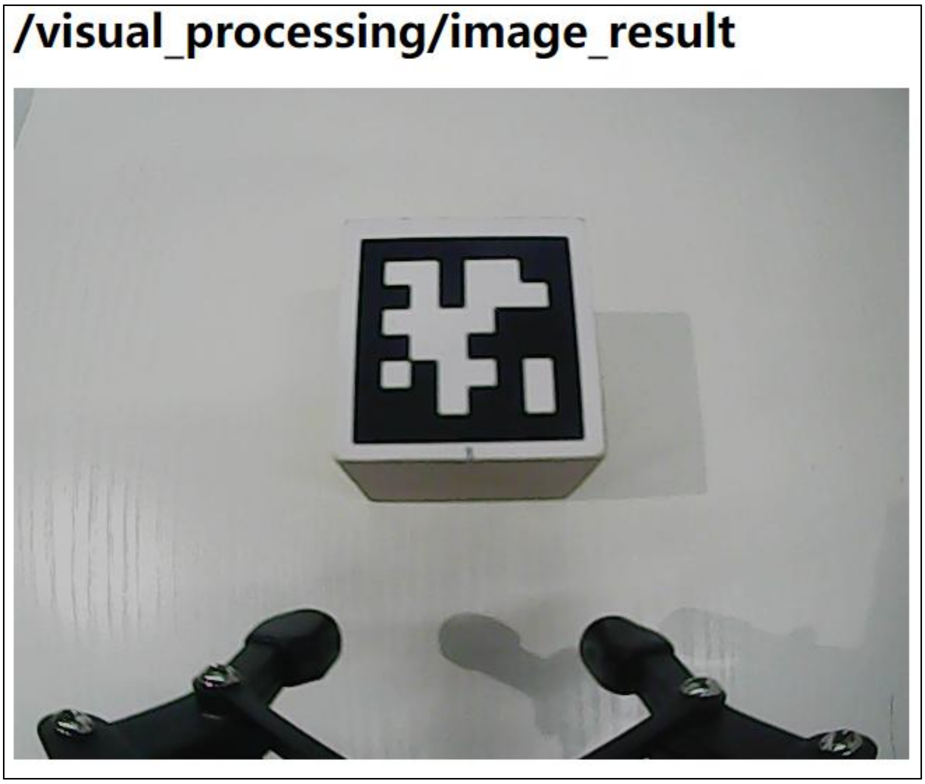
192.168.149.1:8080/

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



Available ROS Image Topics:

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

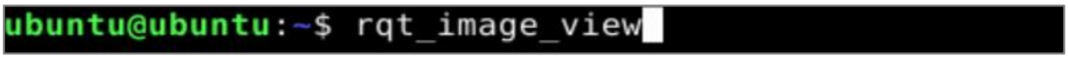


/visual_processing/image_result

2.2.2 Start with rqt

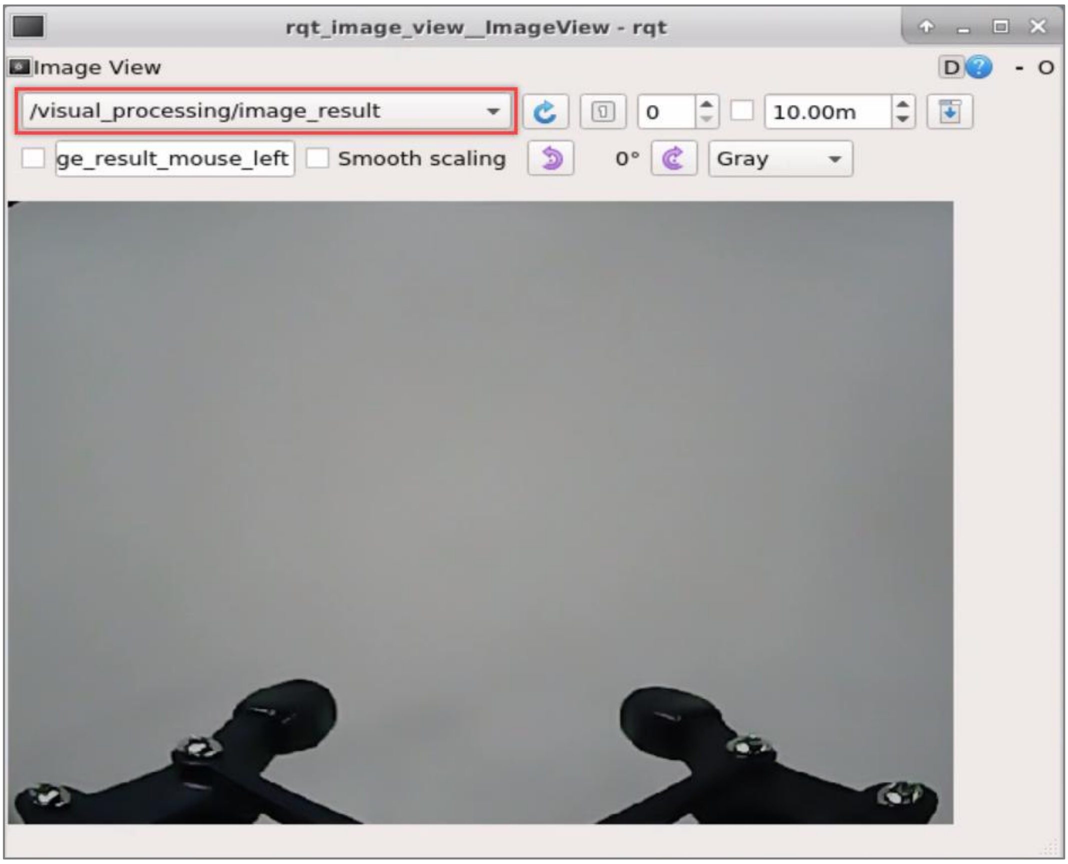
1) After completing the steps of "2.1 Enter Game" and do not exit the terminal, open a new terminal.

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.

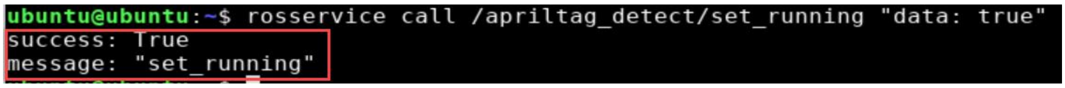


rqt_image_view__ImageView - rqt
Image View
/visual_processing/image_result
0
10.00m
ge_result_mouse_left
Smooth scaling
0°
Gray

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

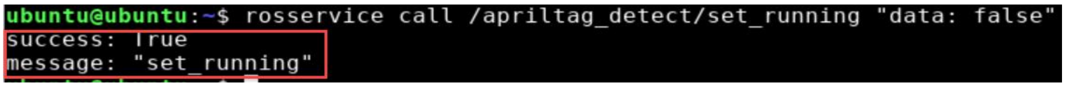
Now, enter the terminal according to the steps in "2.1 Enter Game" and input command "**rosservice call /apriltag_detect/set_running "data: true"**". Then if the prompt shown in the following red box appears, which means game has been started successfully.



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

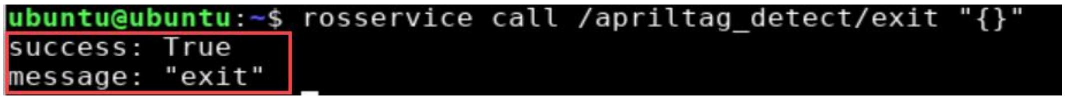
2.4 Stop and Exit

1) If want to stop the game, enter command “**rosservice call /apriltag_detect/set_running "data: false"** ”. After stopping , you can refer to the content of “2.3 Start Game” to change line color and start following again.



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

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



```
ubuntu@ubuntu:~$ rosservice call /apriltag_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 recognize the tag ID. Then, we can see that the tag ID will be framed in rqt tool after recognition and the robotic arm will perform the corresponding action.

Tag ID	Corresponding action
1	Drawing a triangle.
2	Drawing a circle
3	Drifting performance

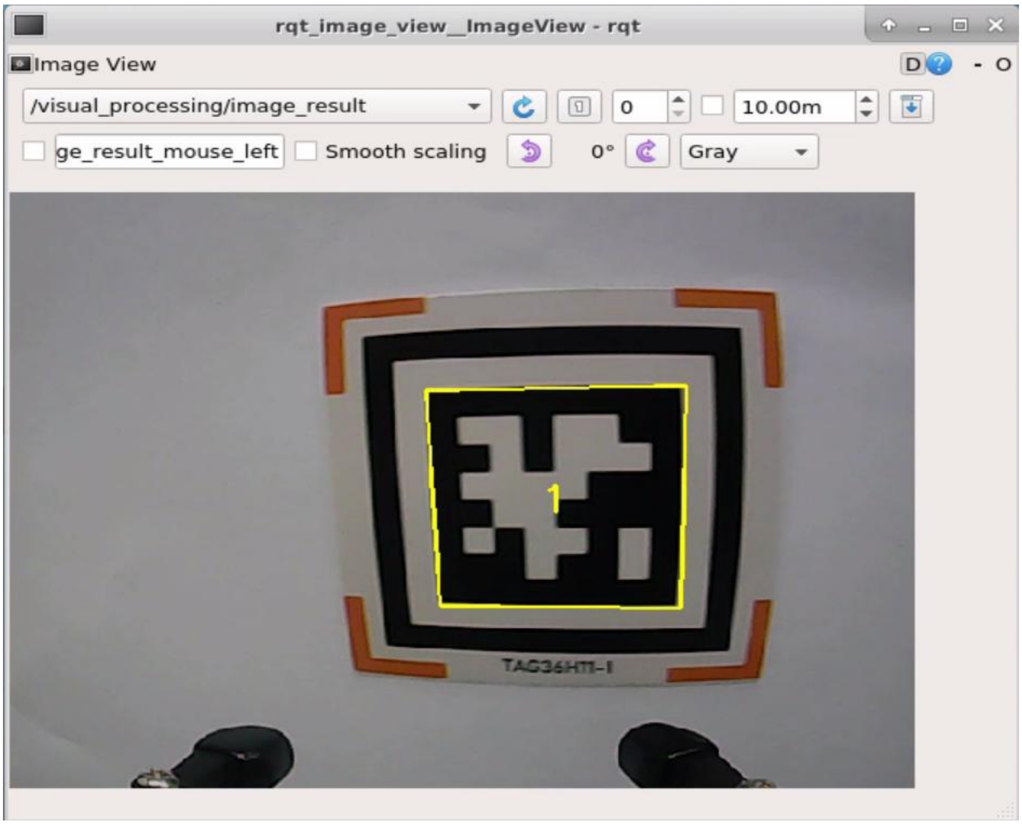

rqt_image_view__ImageView - rqt

Image View

/visual_processing/image_result

0

10.00m

□ ge_result_mouse_left □ Smooth scaling

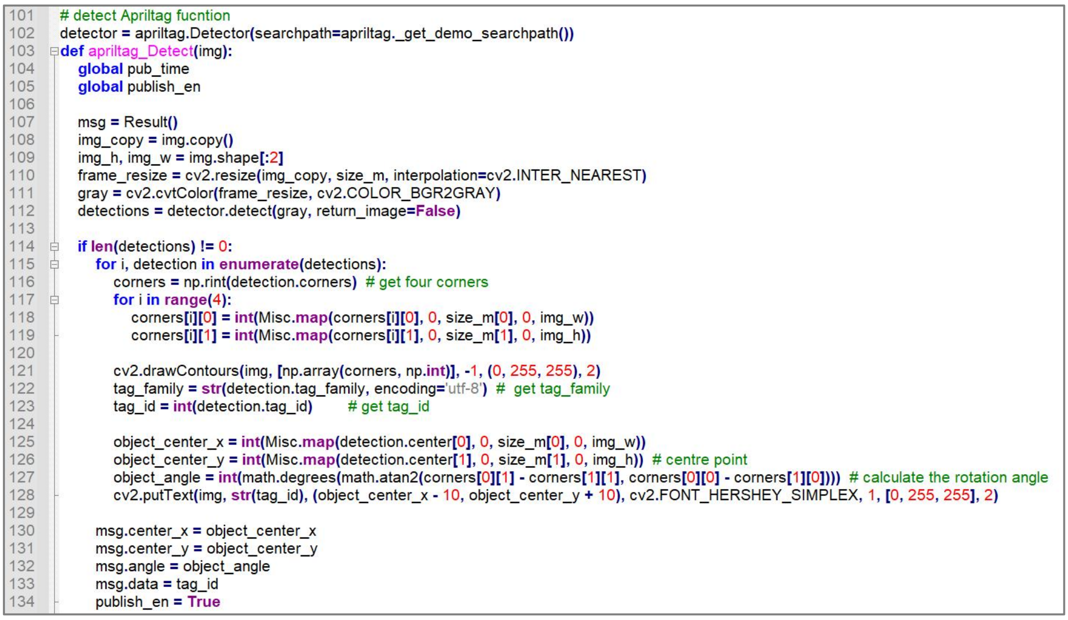
0°

Gray

4. Program Parameter Instruction

4.1 Image Process

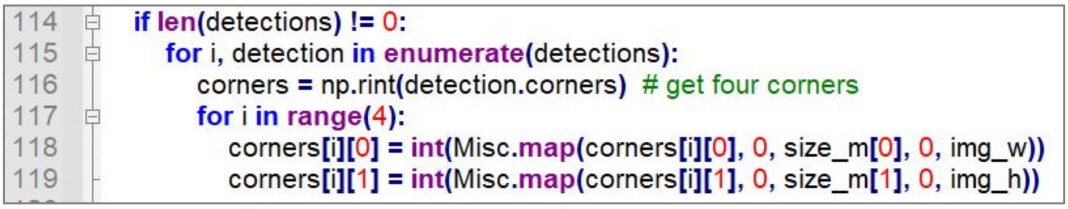
The source code of image process program is located in: **/home/ubuntu/armpi_pro/src/visual_processing/scripts/visual_processing_node.py**



```python
# detect Apriltag fucntion
detector = apriltag.Detector(searchpath=apriltag._get_demo_searchpath())
def apriltag_Detect(img):
    global pub_time
    global publish_en

    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):
            corners = np.rint(detection.corners)  # get four corners
            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))

            cv2.drawContours(img, [np.array(corners, np.int)], -1, (0, 255, 255), 2)
            tag_family = str(detection.tag_family, encoding='utf-8')  #  get tag_family
            tag_id = int(detection.tag_id)        # get tag_id

            object_center_x = int(Misc.map(detection.center[0], 0, size_m[0], 0, img_w))
            object_center_y = int(Misc.map(detection.center[1], 0, size_m[1], 0, img_h))  # centre point
            object_angle = int(math.degrees(math.atan2(corners[0][1] - corners[1][1], corners[0][0] - corners[1][0])))  # calculate the rotation angle
            cv2.putText(img, str(tag_id), (object_center_x - 10, object_center_y + 10), cv2.FONT_HERSHEY_SIMPLEX, 1, [0, 255, 255], 2)

        msg.center_x = object_center_x
        msg.center_y = object_center_y
        msg.angle = object_angle
        msg.data = tag_id
        publish_en = True
```

4.1.1 Obtain the vertices information

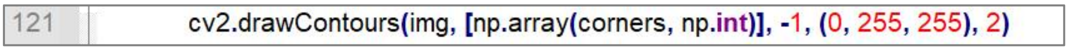
Get the four vertices of tag with np.rint() function.



```python
    if len(detections) != 0:
        for i, detection in enumerate(detections):
            corners = np.rint(detection.corners)  # get four corners
            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))
```

4.1.2 Detect tag

1) After obtaining the vertices information, the tag is recognized by calling drawContours() function in cv2 library.



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

The meaning of parameters in parentheses is as follow:

The first parameter “img” is the input image.

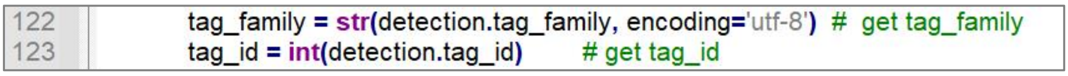
The second parameter “[np.array(corners, np.int)]” is the contour which is list in Python.

The third parameter “-1” is the index of the contour, where the value represents all contours in the drawn contour list.

The fourth parameter “(0, 255, 255)” is the color of contour and its order is B, G and R.

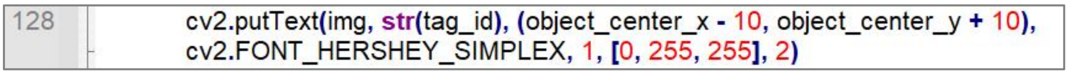
The fifth parameter “2” is the width of contour.

1) The type of the obtained tag (tag_family) and ID (tag_id).



```
            tag_family = str(detection.tag_family, encoding='utf-8')  #  get tag_family
            tag_id = int(detection.tag_id)        # get tag_id
```

2) Print the tag ID and type in the transmitted image by calling putText() function in cv2 library.



```
            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 display 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 type of font.

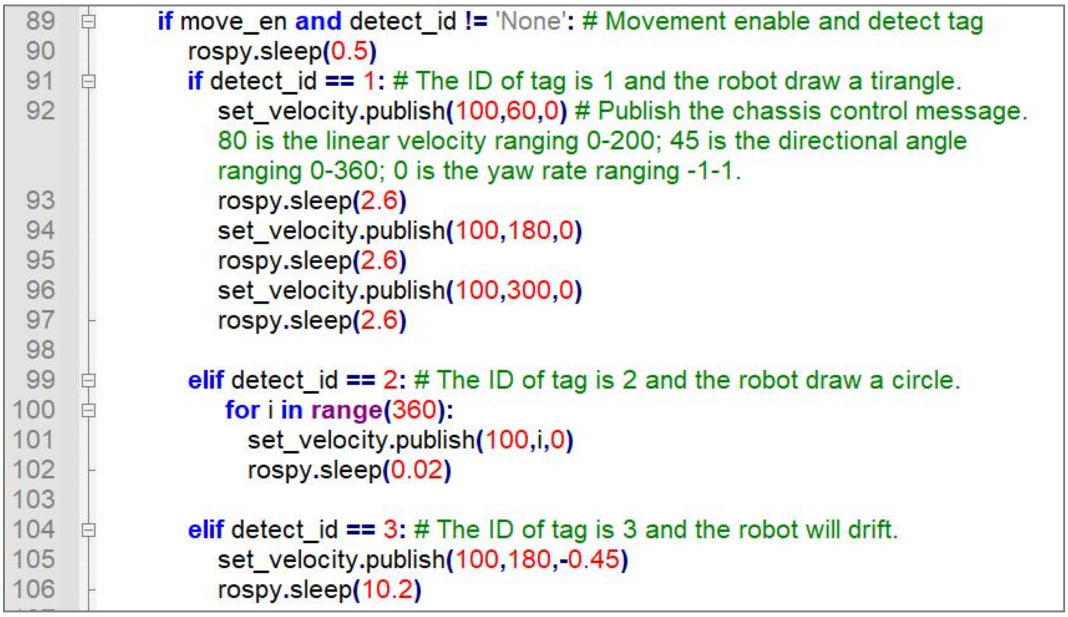
The fifth parameter “1” is the size of font.

The sixth parameter “[0, 255, 255]” is the color of font and its color is B, G and R. Here is yellow.

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

4.2 Control action

After obtaining ID, control ArmPi Pro to perform corresponding action by calling set_velocity.publish() function in hiwonder_servo_msgs.msg library.



```
if move_en and detect_id != 'None': # Movement enable and detect tag
    rospy.sleep(0.5)
    if detect_id == 1: # The ID of tag is 1 and the robot draw a tirangle.
        set_velocity.publish(100,60,0) # Publish the chassis control message. 80 is the linear velocity ranging 0-200; 45 is the directional angle ranging 0-360; 0 is the yaw rate ranging -1-1.
        rospy.sleep(2.6)
        set_velocity.publish(100,180,0)
        rospy.sleep(2.6)
        set_velocity.publish(100,300,0)
        rospy.sleep(2.6)

    elif detect_id == 2: # The ID of tag is 2 and the robot draw a circle.
        for i in range(360):
            set_velocity.publish(100,i,0)
            rospy.sleep(0.02)

    elif detect_id == 3: # The ID of tag is 3 and the robot will drift.
        set_velocity.publish(100,180,-0.45)
        rospy.sleep(10.2)
```

Take code “set_velocity.publish(100,60,0)” as example. The meaning of parameters in parentheses is as follow:

The first parameter “100” is the linear velocity.

The second parameter “60” is the direction angle.

The third parameter “0” is the yaw rate.