Voice Control Color Sorting and Stacking

Before running the function, you need to close the App and large programs. For the closing method, refer to [4.Preparation] - [1. Manage APP control services].

1. Function Description

Through voice commands for color block sorting order, the robotic arm will sort color blocks according to the given order.

2. Startup and Operation

2.1. Startup

2.1.1. Orin Board

First start the ROS node service. Open the terminal on the main board and enter the following:

```
ros2 run dofbot_pro_info kinemarics_dofbot
```

Then enter the following command in the terminal to start:

```
python ~/dofbot_voice/scripts/voice_color_sorting.py
```

2.1.2. Jetson-Nano Board

First start roscore. Open the terminal on the main board and enter the following:

```
roscore
```

Then start the ROS node service. Open the terminal on the main board and enter the following:

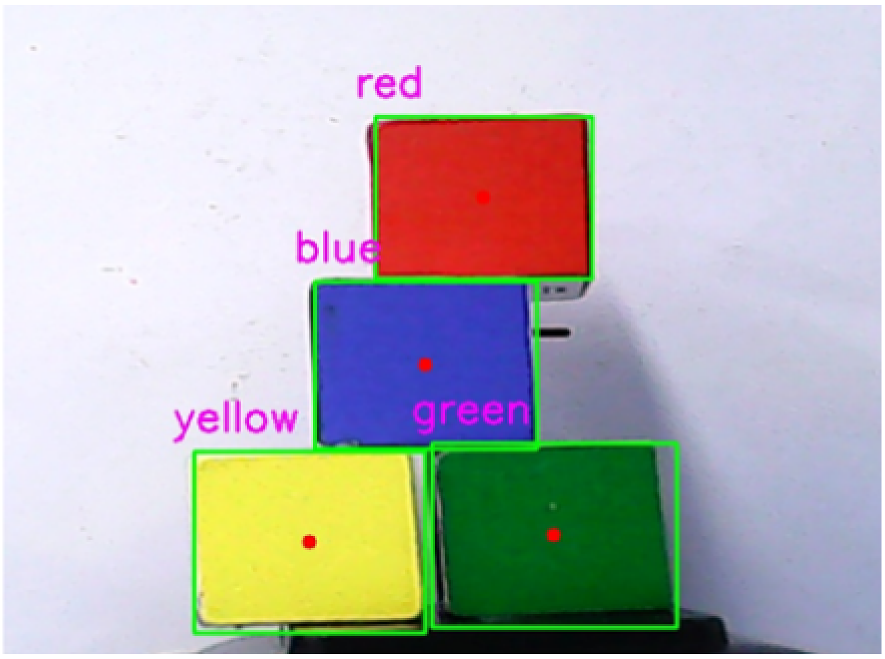
```
rosrun dofbot_pro_info kinemarics_dofbot_pro
```

Finally, open the terminal and enter the following command to start:

```
python ~/dofbot_voice/scripts/voice_color_sorting.py
```

2.2. Operation Steps

After the program runs, the voice broadcast module will broadcast the colors of all blocks in the box. For example, in the figure below, it will broadcast "I see red, blue, yellow, green".



Then it will broadcast "The first color to sort is:". At this time, wake up the voice module with the keyword "Hello, yahboom" and say the first color to sort, for example "red". The program will record the first color to sort, then broadcast "The second color to sort is:", then continue to wake up with the keyword and say the color. If there is a color that doesn't need sorting, you can say "No". After sorting is completed, it will broadcast "Sorting completed, whether to start sorting". At this time, say "Start sorting", it will broadcast "ok", then execute the sorting action.

Note: If colors cannot be recognized, you may need to modify the HSV values in the program or adjust the ambient lighting.

3. Core Code Analysis

`Source code path: ~/dofbot_voice/scripts/voice_color_sorting.py`

```
#Mainly look at the imported library file, speech_identify_target library, this
library is located at
~/dofbot_pro/dofbot_color_sorting/scripts/identify_target.py
from speech_identify_target import identify_GetTarget
## Create target acquisition instance
target      = identify_GetTarget()
#Call select_color function to select, determine color and color block position
img, msg, coo1, coo2, coo3, coo4 = target.select_color(img, color_hsv,
color_list)
#Start thread, start grabbing program
threading.Thread(target=target.target_run, args=(msg,xy)).start()
```

In the speech_identify_target library:

```
#Select recognition color, the returned msg includes the selected color and the
position of that color block
def select_color(self, image, color_hsv, color_list):
    '''
    Choose a recognition color
    :param image: input image
    :param color_hsv: Range threshold for HSV
    :param color_list: Color sequence: ['0': None '1': Red '2': Green '3': Blue
'4': Yellow]
```

```
        :return: Output the processed image, (color, position)
        '''
        # canonical input image size
        self.image = image.copy()
        msg = {}
        # Get information about the first color sequence
        if len(color_list)==0:return self.image,msg
        if '1' in color_list:
            self.color_name=color_list['1']
            pos = self.get_Sqaure(color_hsv[self.color_name])
            if pos != None: msg[self.color_name] = pos
        if '2' in color_list:
            self.color_name=color_list['2']
            pos = self.get_Sqaure(color_hsv[self.color_name])
            if pos != None: msg[self.color_name] = pos
        if '3' in color_list:
            self.color_name=color_list['3']
            pos = self.get_Sqaure(color_hsv[self.color_name])
            if pos != None: msg[self.color_name] = pos
        if '4' in color_list:
            self.color_name=color_list['4']
            pos = self.get_Sqaure(color_hsv[self.color_name])
            if pos != None: msg[self.color_name] = pos
        return self.image, msg

    #Grab function
    def target_run(self, msg, xy=None):
        '''
        grab function
        :param msg: (color, position)
        '''
        if xy != None: self.xy = xy
        num = 1
        move_status=0
        for i in msg.values():
            if i !=None: move_status=1
        if move_status==1:
            self.arm.Arm_Buzzer_On(1)
            sleep(0.5)
        for name, pos in msg.items():
            # Here, ROS inversely solves the communication to obtain the rotation
    angle of each joint
            joints = self.server_joint(pos)
            self.grap.identify_move(str(name), joints)
            num += 1

        if move_status==1:
            # set up
            joints_uu = [90, 80, 50, 50, 90, 30]
            # Move over the object's position
            self.arm.Arm_serial_servo_write6_array(joints_uu, 1000)
            sleep(1.5)
            # initial position
            joints_0 = [self.xy[0], self.xy[1], 0, 0, 90, 30]
            # move to initial position
            self.arm.Arm_serial_servo_write6_array(joints_0, 1000)
            sleep(1.5)
```

```
#Call inverse solution service, provide xy to calculate rotation angle of each
joint
def server_joint(self, posxy):
    '''
    Post position request, get joint rotation angle
    :param posxy: Location point x,y coordinates
    :return: Rotation angle of each joint
    '''
    # Create a message pack
    request = Kinemarics.Request()
    request.tar_x = posxy[0]
    request.tar_y = posxy[1]
    request.tar_z = 0.02
    request.roll = -1.57
    request.pitch = 0.0
    request.yaw = 0.0
    request.kin_name = "ik"
    future = self.client.call_async(request)
    rclpy.spin_until_future_complete(self, future)
    response = future.result()
    if response is not None:
        # Get the inverse solution response result
        joints = [0.0, 0.0, 0.0, 0.0, 0.0]
        joints[0] = response.joint1
        joints[1] = response.joint2
        joints[2] = response.joint3
        joints[3] = response.joint4
        joints[4] = posxy[2] + 90
        return joints
    else:
        self.get_logger().info('Service call failed')
```