

Color block shape sorting

Before starting this function, you need to close the process of the big program and APP. If you need to start the big program and APP again later, start the terminal,

```
bash ~/dofbot_pro/APP_DOFBOT_PRO/start_app.sh
```

1. Function description

After the program runs, place the same color block. After the camera captures the image, it will perform color recognition based on the HSV value and recognize the shape of the color block. Press the space bar, the robot will grab the color block of the target shape and place it at the set position; after the placement is completed, it returns to the initial recognition posture of the color block.

2. Start and operate

(1) Start command

Enter the following command in the terminal to start,

```
#Start the camera:
ros2 launch orbbec_camera dabai_dcw2.launch.py
#Start the underlying control:
ros2 run dofbot_pro_driver arm_driver
#Start the inverse program:
ros2 run dofbot_pro_info kinematics_dofbot
#Start the robot arm grasping program:
ros2 run dofbot_pro_driver grasp
#Start the color block shape recognition program:
ros2 run dofbot_pro_color shape_recognize
```



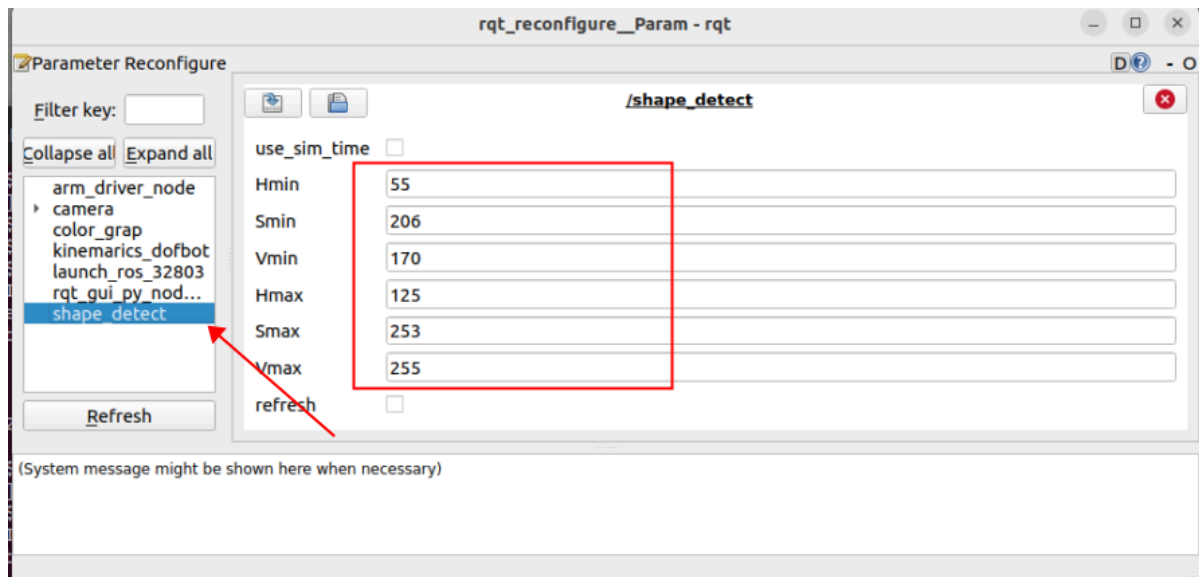
(2) Operation process

After the terminal is started, place a color block of the same color. The camera captures the image. Press [r] or [R] to enter the color selection mode. Use the mouse to select a certain area of the color block, obtain the HSV value of this area, and release the mouse to enter the color recognition mode. After entering the color recognition mode, if the current HSV value still cannot filter out

other colors, you can use the dynamic parameter adjuster to fine-tune the HSV. Enter the following command in the terminal to start the dynamic parameter adjuster.

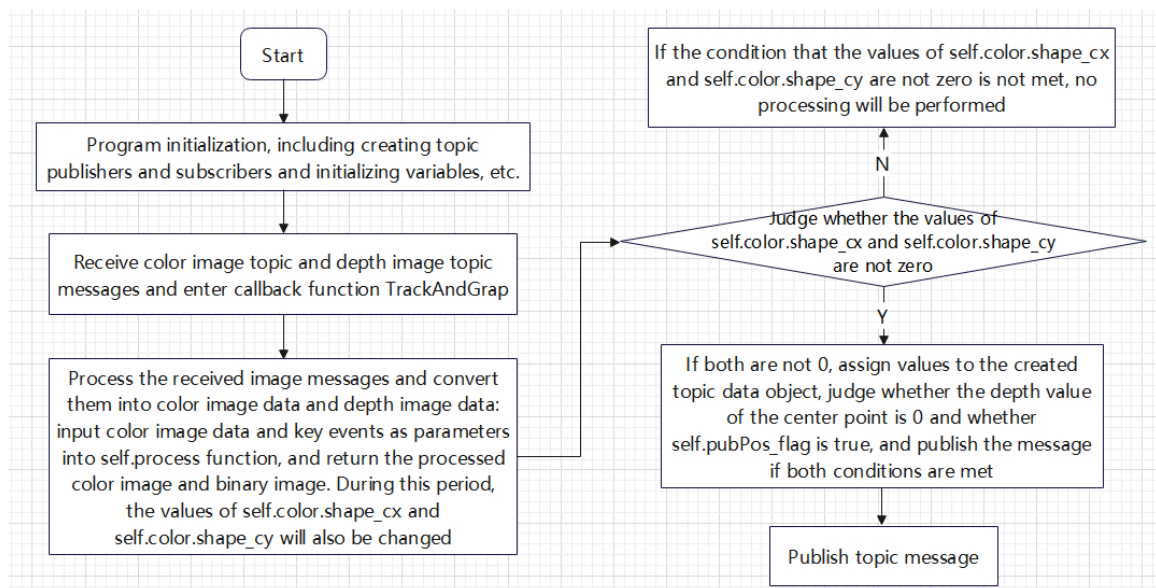
```
ros2 run rqt_reconfigure rqt_reconfigure
```

You can modify the HSV value through the slider. When the image on the left (binarization) only shows the only recognized color, click the color image box and press the space bar. The robot arm will lower its claws to grab the color block of the set target shape. The default target shape is a square.



3. Program flow chart

shape_recognize.py



4. Core code analysis

4.1. shape_recognize.py

Code path:

```
/home/jetson/dofbot_pro_ws/src/dofbot_pro_color/dofbot_pro_color/shape_recognize.py
```

astra_common code library path:

```
/home/jetson/dofbot_pro_ws/src/dofbot_pro_color/dofbot_pro_color/astra_common.py
```

Import necessary libraries,

```
import cv2
import rclpy
from rclpy.node import Node
import numpy as np
from message_filters import ApproximateTimeSynchronizer, Subscriber
from sensor_msgs.msg import Image
from std_msgs.msg import Float32, Bool
from cv_bridge import CvBridge
import cv2 as cv

import time
import math
import os
encoding = ['16UC1', '32FC1']
import tf_transformations as tf
import transforms3d as tfs

from dofbot_pro_color.astra_common import *
from dofbot_pro_interface.msg import *
```

Initialize program parameters, create publishers, subscribers, etc.

```
def __init__(self):
    super().__init__('shape_detect')
    self.declare_param()
    self.target_servox=90
    self.window_name = "depth_image"
    self.target_servoy=45
    self.init_joints = [90.0, 120.0, 0.0, 0.0, 90.0, 90.0]
    #Create a publisher for publishing color block information
    self.pub_ColorInfo = self.create_publisher(AprilTagInfo, "PosInfo", 1)
    #Create a publisher for publishing the target angle of the robot arm
    self.pubPoint = self.create_publisher(ArmJoint, "TargetAngle", 1)
    #Create a subscriber for subscribing to gesture recognition results
    self.grasp_status_sub = self.create_subscription(Bool, 'grasp_done',
self.GraspStatusCallback, 1)
    #Create two subscribers to subscribe to the color image topic and the depth
image topic
    self.depth_image_sub = Subscriber(self, Image, "/camera/color/image_raw",
qos_profile=1)
    self.rgb_image_sub = Subscriber(self, Image, "/camera/depth/image_raw",
qos_profile=1)
    self.TimeSynchronizer = ApproximateTimeSynchronizer([self.depth_image_sub,
self.rgb_image_sub],queue_size=10,slop=0.5)
```

```

self.TimeSynchronizer.registerCallback(self.TrackAndGrap)
#Create a bridge for converting color and depth image topic message data to
image data
self.rgb_bridge = CvBridge()
self.depth_bridge = CvBridge()

#color
#Initialize the region coordinates
self.Roi_init = ()
#Initialize the HSV value
self.hsv_range = ()
#Initialize the information of the color block, which represents the center x
coordinate, center y coordinate and minimum circumscribed circle radius r of the
color block
self.circle = (0, 0, 0)
#Flag for dynamic parameter adjustment, if True, dynamic parameter adjustment
is performed
self.dyn_update = True
#Flag for mouse selection
self.select_flags = False
self.gTracker_state = False
self.windows_name = 'frame'
self.Track_state = 'init'
#Create color detection object
self.color = color_detect()
#Initialize the row and column coordinates of the region coordinates
self.cols, self.rows = 0, 0
#Initialize the xy coordinates of the mouse selection
self.Mouse_XY = (0, 0)
#The path of the default HSV threshold file, which stores the last saved HSV
value
self.hsv_text =
"/home/jetson/dofbot_pro_ws/src/dofbot_pro_color/dofbot_pro_color/colorHSV.text"
if os.path.exists(self.hsv_text): self.roi_hsv_range =
read_HSV(self.hsv_text)
#Publish the flag of machine code information. When it is True, publish the
/PosInfo topic data
self.pubPos_flag = False
#Set the target shape. The default is a cube. You can choose a cuboid or a
cylinder
self.color.target_shape = "Square" # "Rectangle" ,"Cylinder"
print("Target shape: ",self.color.target_shape)

```

Mainly look at the image processing function TrackAndGrap,

```

def TrackAndGrap(self,color_frame,depth_frame):
    #rgb_image
    #Receive a color image topic message and convert the message data into image
data
    rgb_image = self.rgb_bridge.imgmsg_to_cv2(color_frame,'bgr8')
    result_image = np.copy(rgb_image)
    #depth_image
    #Receive a depth image topic message and convert the message data into image
data
    depth_image = self.depth_bridge.imgmsg_to_cv2(depth_frame, encoding[1])

```

```

frame = cv.resize(depth_image, (640, 480))
depth_image_info = frame.astype(np.float32)
action = cv.waitKey(10) & 0xFF
result_image = cv.resize(result_image, (640, 480))
# Pass the obtained color image as a parameter to process, and also pass it
to the keyboard event action
result_frame, binary = self.process(result_image, action)
# Check if self.color.shape_cx and self.color.shape_cy are not 0, indicating
that there is a color block that meets the conditions
if self.color.shape_cx!=0 and self.color.shape_cy!=0:
    pos = AprilTagInfo()
    pos.x = self.color.shape_cx
    pos.y = self.color.shape_cy
    print(self.color.shape_cx, self.color.shape_cy)
    pos.z = depth_image_info[self.color.shape_cy, self.color.shape_cx]/1000
    print("depth: ", pos.z)
    #Judge whether the value of self.pubPos_flag is True and whether the
    depth value of the color block is not 0. If both conditions are met, the message
    can be published
    if self.pubPos_flag == True and pos.z!=0:
        self.pubPos_flag = False
        self.pub_ColorInfo.publish(pos)
    # Check if the binary image exists. If it does, display the color and binary
    images. Otherwise, only display the color image.
    if len(binary) != 0: cv.imshow(self.windows_name, ManyImgs(1, ([result_frame,
    binary])))
    else:
        cv.imshow(self.windows_name, result_frame)

```

Image processing function self.process,

```

def process(self, rgb_img, action):
    rgb_img = cv.resize(rgb_img, (640, 480))
    binary = []
    #Judge key events. When the space bar is pressed, change the information
    release status. Self.pubPos_flag is True, indicating that the information topic
    can be released.
    if action == 32: self.pubPos_flag = True
    #Judge key events. When i or I is pressed, change the status to
    identification mode.
    elif action == ord('i') or action == ord('I'): self.Track_state = "identify"
    #Judge key events. When r or R is pressed, reset all parameters and enter
    color selection mode.
    elif action == ord('r') or action == ord('R'): self.Reset()
    #Judge the status value. If it is init, it means the initial status value. At
    this time, you can use the mouse to select the area.
    if self.Track_state == 'init':
        cv.namedWindow(self.windows_name, cv.WINDOW_AUTOSIZE)
        #Select the color of an area in the specified window
        cv.setMouseCallback(self.windows_name, self.onMouse, 0)
        #Judge the color selection flag, true means that the color can be
        selected
        if self.select_flags == True:
            cv.line(rgb_img, self.cols, self.rows, (255, 0, 0), 2)
            cv.rectangle(rgb_img, self.cols, self.rows, (0, 255, 0), 2)

```

```

        # Check if the selected area exists
        if self.Roi_init[0] != self.Roi_init[2] and self.Roi_init[1] !=
self.Roi_init[3]:
            #Call the Roi_hsv function in the created color detection object
self.color, and return the processed color image and HSV value
            rgb_img, self.hsv_range = self.color.Roi_hsv(rgb_img,
self.Roi_init)
            self.gTracker_state = True
            self.dyn_update = True
        else: self.Track_state = 'init'
        #Judge the status value. If it is "identify", it means that color recognition
can be performed.
        elif self.Track_state == "identify":
            # Check if there is an HSV threshold file. If it exists, read the value
in it and assign it to hsv_range
            if os.path.exists(self.hsv_text): self.hsv_range =
read_HSV(self.hsv_text)
            # If it does not exist, change the state to init to select the color
            else: self.Track_state = 'init'
            if self.Track_state != 'init':
                #Judge the length of the self.hsv_range value, that is, whether the value
exists. When the length is not 0, enter the color detection function
                if len(self.hsv_range) != 0:
                    #Call the ShapeRecognition function in the created color detection
object self.color, pass in the color image and self.hsv_range, which is the hsv
threshold, and return the processed color image, the binary image, and the
information of the graphics that meet the hsv threshold, including the center
point coordinates and the radius of its minimum circumscribed circle
                    rgb_img, binary, self.circle = self.color.ShapeRecognition(rgb_img,
self.hsv_range)
                    #Judge the flag of dynamic parameter update. True means that the
hsv_text file can be updated and the value on the parameter server can be
modified
                    if self.dyn_update == True:
                        write_HSV(self.hsv_text, self.hsv_range)
                        params = {'Hmin': self.hsv_range[0][0], 'Hmax': self.hsv_range[1]
[0],
                                'Smin': self.hsv_range[0][1], 'Smax': self.hsv_range[1]
[1],
                                'Vmin': self.hsv_range[0][2], 'Vmax': self.hsv_range[1]
[2]}
                        self.dyn_client.update_configuration(params)
                        self.dyn_update = False
                    return rgb_img, binary

```

4.2、grasp.py

Please refer to the content of [grasp.py] in section 4.2 of the tutorial [Three-dimensional space sorting and gripping\1. Machine code ID sorting].