Block Color Sorting

Before starting this function, you need to close the large program and APP processes. If you need to restart the large program and APP later, start them from the terminal:

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

1. Function Description

After the program starts, the robotic arm will perform color recognition based on the set HSV values. After pressing the spacebar, the robotic arm will lower the gripper to grasp the recognized robotic arm, and place it at the set position after grasping; after placement is complete, it returns to the recognition posture.

2. Startup and Operation

2.1. Startup Commands

Enter the following commands in the terminal to start:

```
#Start camera:
ros2 launch orbbec_camera dabai_dcw2.launch.py
#Start underlying control:
ros2 run dofbot_pro_driver arm_driver
#Start inverse kinematics program:
ros2 run dofbot_pro_info kinemarics_dofbot
#Start color recognition program:
ros2 run dofbot_pro_color color_detect
#Start robotic arm grasping program:
ros2 run dofbot_pro_driver grasp
```

2.2. Operation Process

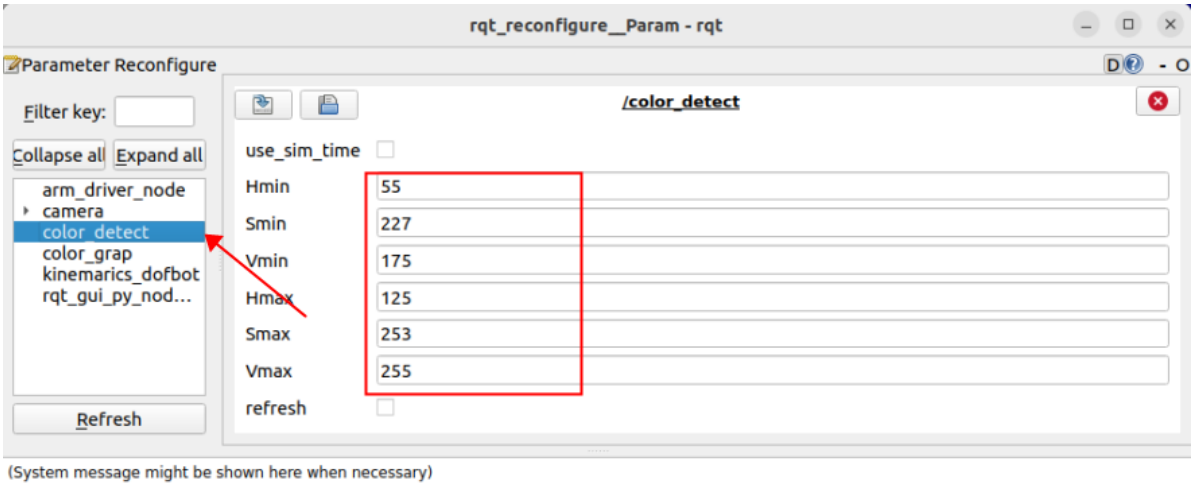
After starting in the terminal, place color blocks of different colors. The camera captures the image. Process the image with the following keys:

- [i] or [I]: Enter color recognition mode, directly load the HSV values calibrated by the program last time for recognition;
- [r] or [R]: Reset program parameters, enter color selection mode, use the mouse to select a certain area of the color block, obtain the HSV values of this area, release the mouse to enter color recognition mode
- Spacebar: Start grasping the recognized color block

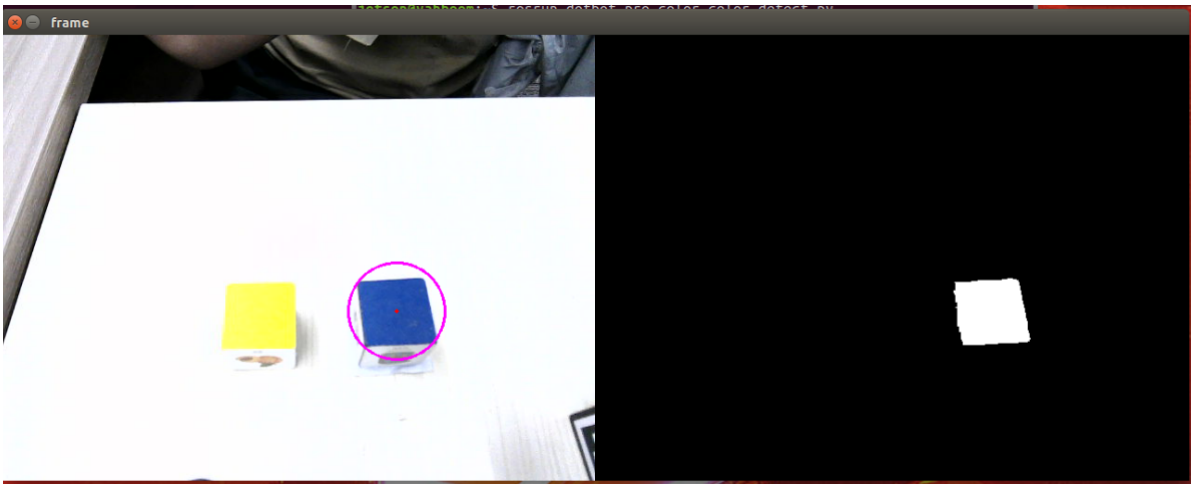
After entering color recognition mode, if the current HSV values still cannot filter out other colors, you can fine-tune the HSV values through the dynamic parameter tuner. Enter the following command in the terminal to start the dynamic parameter tuner:

```
ros2 run rqt_reconfigure rqt_reconfigure
```

You can modify the HSV values through sliders

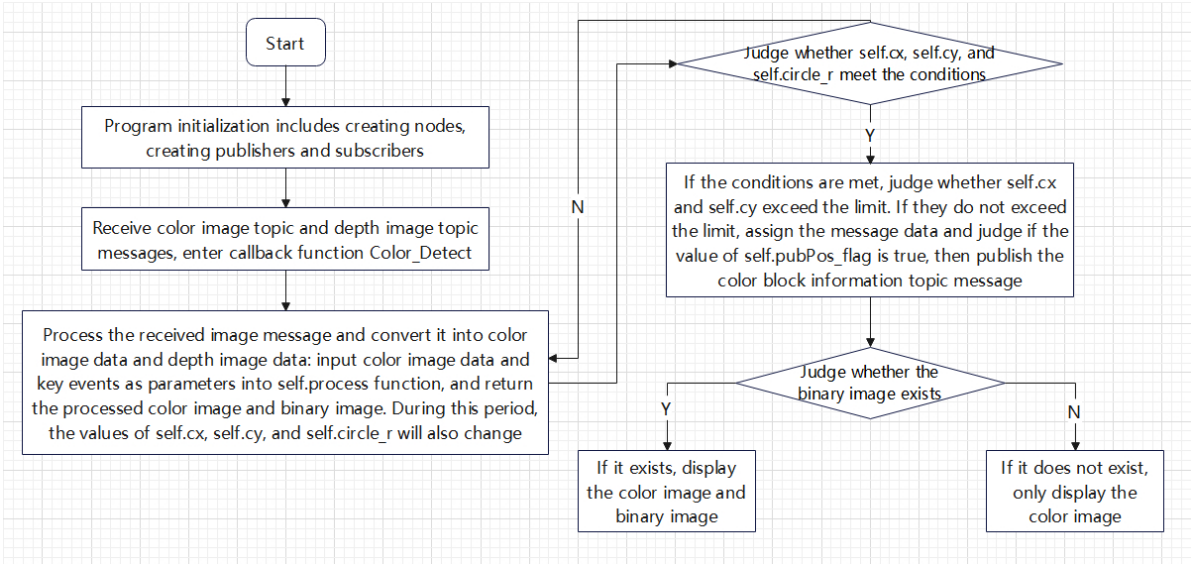


As shown in the figure below, when the left image (binary) only displays the uniquely recognized color, click the color image frame and press the spacebar, the robotic arm will lower the gripper to grasp the recognized color block.



3. Program Flowchart

color_detect.py



4. Core Code Analysis

4.1. color_detect.py

Code path:

```
/home/jetson/dofbot_pro_ws/src/dofbot_pro_color/dofbot_pro_color/color_detect.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
#Import custom image processing library
from dofbot_pro_color.astra_common import *
from dofbot_pro_interface.msg import *
```

Program parameter initialization, create publishers, subscribers, etc.

```
def __init__(self):
    super().__init__('color_detect')
    self.declare_param()
    #Robotic arm color block recognition posture
    self.init_joints = [90.0, 120, 0.0, 0.0, 90, 90]
    #Create publisher for color block information
    self.pub_ColorInfo = self.create_publisher(AprilTagInfo, "PosInfo", 1)
    self.pubPoint = self.create_publisher(ArmJoint, "TargetAngle", 1)
    #Create subscriber for gesture recognition results
    self.grasp_status_sub = self.create_subscription(Bool, 'grasp_done',
self.GraspStatusCallback, 1)
    #Create two subscribers, subscribe to color image topic and 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)
    #Time-synchronize color and depth image subscription messages
```

```
        self.TimeSynchronizer = ApproximateTimeSynchronizer([self.depth_image_sub,
self.rgb_image_sub],queue_size=10,slop=0.5)
        self.TimeSynchronizer.registerCallback(self.Color_Detect)
    #Save xy coordinates of color block center
    self.y = 0
    self.x = 0
    #Create bridges for converting color and depth image topic message data to
image data
    self.rgb_bridge = CvBridge()
    self.depth_bridge = CvBridge()
    #Initialize region coordinates
    self.Roi_init = ()
    #Initialize HSV values
    self.hsv_range = ()
    #Initialize recognized color block information, here representing color block
center x coordinate, center y coordinate and minimum enclosing circle radius r
    self.circle = (0, 0, 0)
    #Dynamic parameter adjustment flag, True means perform dynamic parameter
adjustment
    self.dyn_update = True
    #Mouse selection flag
    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 row and column coordinates of region coordinates
    self.cols, self.rows = 0, 0
    #Initialize mouse selected xy coordinates
    self.Mouse_XY = (0, 0)
    #Image processed color block center coordinates xy
    self.cx = 0
    self.cy = 0
    #Default HSV threshold file path, this file stores the last saved HSV values
    self.hsv_text =
"/home/jetson/dofbot_pro_ws/src/dofbot_pro_color/dofbot_pro_color/colorHSV.text"

    #Minimum enclosing circle radius of color block obtained after image
processing
    self.circle_r = 0 #Prevent false recognition of other scattered points
    #Flag for publishing AprilTag information, when True publish /PosInfo topic
data
    self.pubPos_flag = False
```

Main image processing function Color_Detect

```
def Color_Detect(self,color_frame,depth_frame):
    #rgb_image
    #Receive color image topic message, convert message data to image data
    rgb_image = self.rgb_bridge.imgmsg_to_cv2(color_frame,'bgr8')
    result_image = np.copy(rgb_image)
    #depth_image
    #Receive depth image topic message, convert message data to 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 parameter to process, and also pass
keyboard event action
    result_frame, binary = self.process(result_image,action)
    #Determine if self.cx, self.cy are not 0, indicating color block is processed
and when enclosing circle radius is greater than 30, indicating a color block
matching HSV threshold is detected
    if self.cx!=0 and self.cy!=0 and self.circle_r>30:
        if self.cx<=640 or self.cy <=480:
            center_x, center_y = self.cx,self.cy
            self.x = int(center_x)
            self.y = int(center_y)
            #Create message to calculate depth information and assign values to
data inside
            pos = AprilTagInfo()
            pos.x = center_x
            pos.y = center_y
            pos.z = depth_image_info[self.y,self.x]/1000
            #Determine if self.pubPos_flag value is True, True means message can
be published
            if self.pubPos_flag == True:
                self.pub_ColorInfo.publish(pos)
                self.pubPos_flag = False
    #Determine if binary image exists, if exists display color and binary images,
otherwise only display 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 = []
    #Determine key press event, when spacebar is pressed, change information
publishing flag status, self.pubPos_flag True means information topic can be
published
    if action == 32: self.pubPos_flag = True
    #Determine key press event, when i or I is pressed, change state, switch to
recognition mode
    elif action == ord('i') or action == ord('I'): self.Track_state = "identify"
    #Determine key press event, when r or R is pressed, reset all parameters,
enter color selection mode
    elif action == ord('r') or action == ord('R'): self.Reset()
    #Determine state value, if it's init, it means initial state value, at this
time mouse can be used to select region
    if self.Track_state == 'init':
        cv.namedWindow(self.windows_name, cv.WINDOW_AUTOSIZE)
        #Select a region's color within the specified window
        cv.setMouseCallback(self.windows_name, self.onMouse, 0)
        #Determine color selection flag, true means 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)
```

```
            #Determine if selected region exists
            if self.Roi_init[0] != self.Roi_init[2] and self.Roi_init[1] !=
self.Roi_init[3]:
                #Call Roi_hsv function in created color detection object
self.color, returns processed color image and HSV values
                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'
    #Determine state value, if it's "identify", it means color recognition can be
performed
    elif self.Track_state == "identify":
        #Determine if HSV threshold file exists, if exists read values inside
and assign to hsv_range
        if os.path.exists(self.hsv_text): self.hsv_range =
read_HSV(self.hsv_text)
        #If not exists, change state to init for color selection
        else: self.Track_state = 'init'
    if self.Track_state != 'init':
        #Determine length of self.hsv_range value, that is determine if this
value exists, when length is not 0, enter color detection function
        if len(self.hsv_range) != 0:
            #Call object_follow function in created color detection object
self.color, pass color image and self.hsv_range (hsv threshold), returns
processed color image, binary image and information storing graphics matching hsv
threshold, including center point coordinates and its minimum enclosing circle
radius
            rgb_img, binary, self.circle,_= self.color.object_follow(rgb_img,
self.hsv_range)
            #Assign returned values to self.cx and self.cy storing center
values, minimum enclosing circle radius assigned to self.circle_r
            self.cx = self.circle[0]
            self.cy = self.circle[1]
            self.circle_r = self.circle[2]
            #Determine dynamic parameter update flag, True means hsv_text file
can be updated and values on 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

You can refer to section 4.2 [grasp.py] in tutorial [12. 3D Sorting and Grasping Course\1. AprilTag ID sorting].