

# Block Different Shape 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

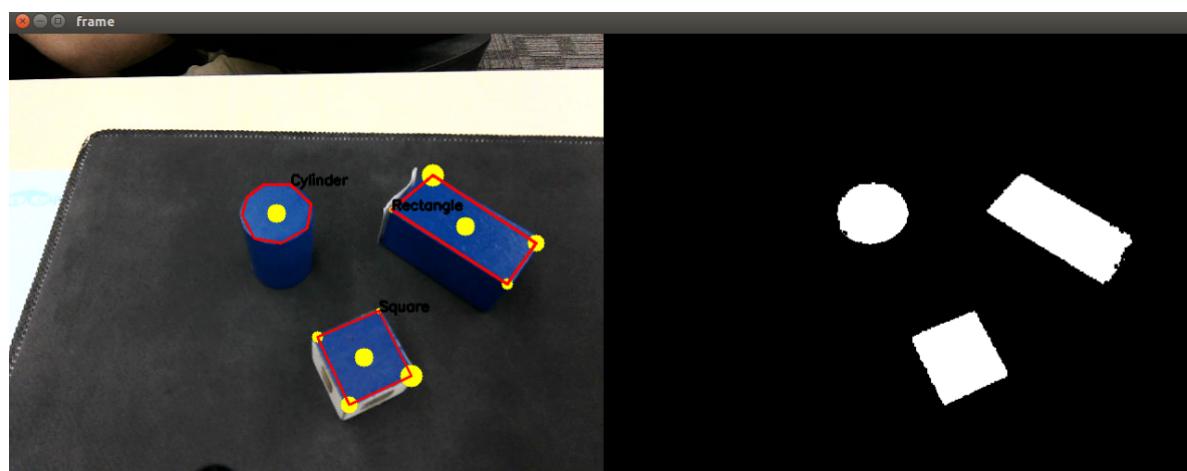
After the program runs, place the same color blocks. After the camera captures the image, perform color recognition based on HSV values and recognize the shapes of the color blocks. Press the spacebar, the robotic arm will grasp the target shape color blocks and place them at the set position; after placement is complete, it returns to the initial color block recognition posture.

## 2. Startup and Operation

### (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 kinematics_dofbot  
#Start robotic arm grasping program:  
ros2 run dofbot_pro_driver grasp  
#Start color block shape recognition program:  
ros2 run dofbot_pro_color shape_recognize
```

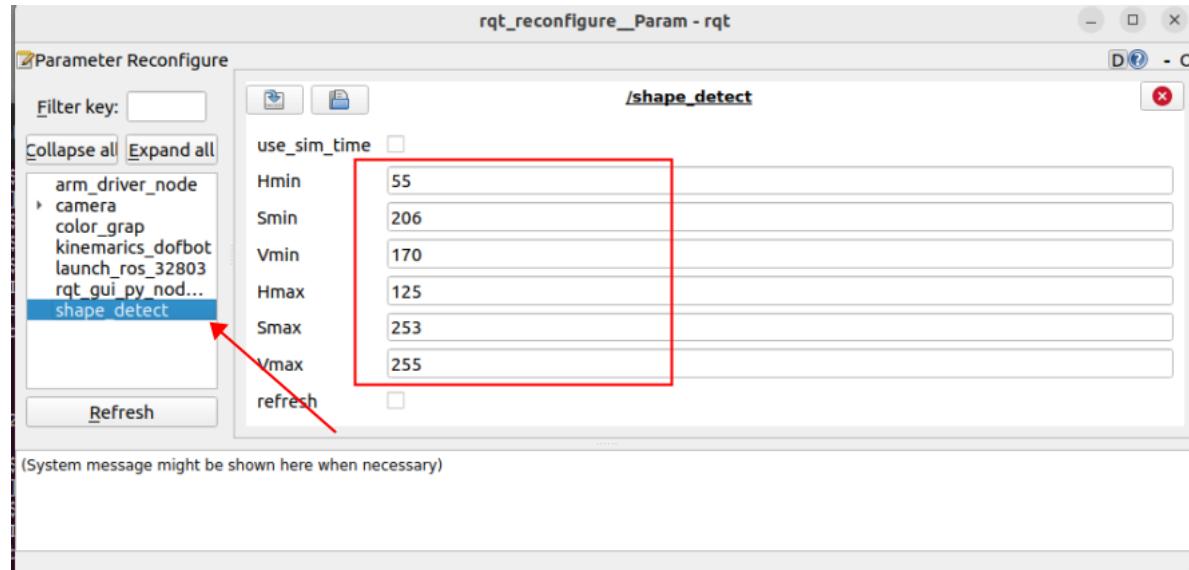


### (2). Operation Process

After starting in the terminal, place color blocks of the same color. The camera captures the image. Press [r] or [R] to 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. 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:

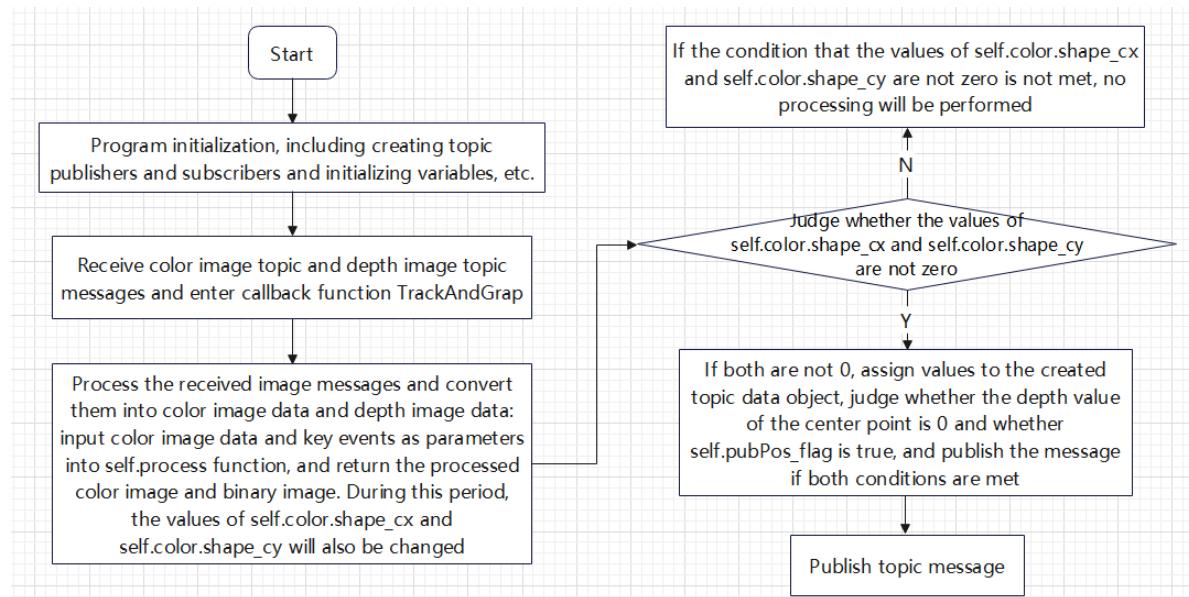
```
ros2 run rqt_reconfigure rqt_reconfigure
```

You can modify the HSV values through sliders. 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 color blocks of the set target shape. The default grasping target shape is square.



### 3. Program Flowchart

`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 *
```

Program parameter initialization, 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 publisher for color block information
    self.pub_ColorInfo = self.create_publisher(AprilTagInfo, "PosInfo", 1)
    #Create publisher for robotic arm target angle
    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)
    self.TimeSynchronizer = ApproximateTimeSynchronizer([self.depth_image_sub,
self.rgb_image_sub],queue_size=10,slop=0.5)
    self.TimeSynchronizer.registerCallback(self.TrackAndGrap)
    #Create bridges for converting color and depth image topic message data to
image data
    self.rgb_bridge = CvBridge()
    self.depth_bridge = CvBridge()

#color
#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)
#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"
if os.path.exists(self.hsv_text): self.roi_hsv_range =
read_HSV(self.hsv_text)
#Flag for publishing AprilTag information, when True publish /PosInfo topic
data
self.pubPos_flag = False
#Set target shape, default is cube, can choose rectangular prism and
cylinder
self.color.target_shape = "Square" # "Rectangle" , "Cylinder"
print("Target shape: ", self.color.target_shape)

```

Main image processing function TrackAndGrap

```

def TrackAndGrap(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.color.shape_cx, self.color.shape_cy are not 0, indicating
    there are color blocks meeting 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)

#Determine if self.pubPos_flag value is True and the color block's depth
value is not 0, meeting both conditions means message can be published
if self.pubPos_flag == True and pos.z!=0:
    self.pubPos_flag = False
    self.pub_colorInfo.publish(pos)

#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_image)

```

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 ShapeRecognition 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.shapeRecognition(rgb_img,
            self.hsv_range)
            #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].