Block different height sorting

Before starting this function, you need to close the process of the big program and APP. Enter the following program in the terminal to close the process of the big program and APP.

```
sh ~/app_Arm/kill_YahboomArm.sh
sh ~/app_Arm/stop_app.sh
```

If you need to start the big program and APP again later, start the terminal.

```
sudo systemctl start yahboom_arm.service
sudo systemctl start yahboom_app.service
```

1. Function description

After the program is started, place color blocks of the same color, and the robot arm will perform color recognition according to the set HSV value. After pressing the space bar, the robot arm will lower its claws to grab the color blocks that exceed the height threshold; after placement, it returns to the recognized posture.

2. Start and operate

2.1. Start command

Enter the following command in the terminal to start,

```
#Start the depth camera
roslaunch orbbec_camera dabai_dcw2.launch
#Start the inverse solution program
rosrun dofbot_pro_info kinemarics_dofbot_pro
#Start the underlying driver to control the robotic arm
rosrun dofbot_pro_info arm_driver.py
#Start the color block height anomaly detection program
rosrun dofbot_pro_color AnomalyHeightRemoval.py
#Robot gripping program
rosrun dofbot_pro_info grasp.py
```

2.2. Operation process

After the terminal is started, place color blocks of different colors, the camera captures the image, and press the following buttons to process the image:

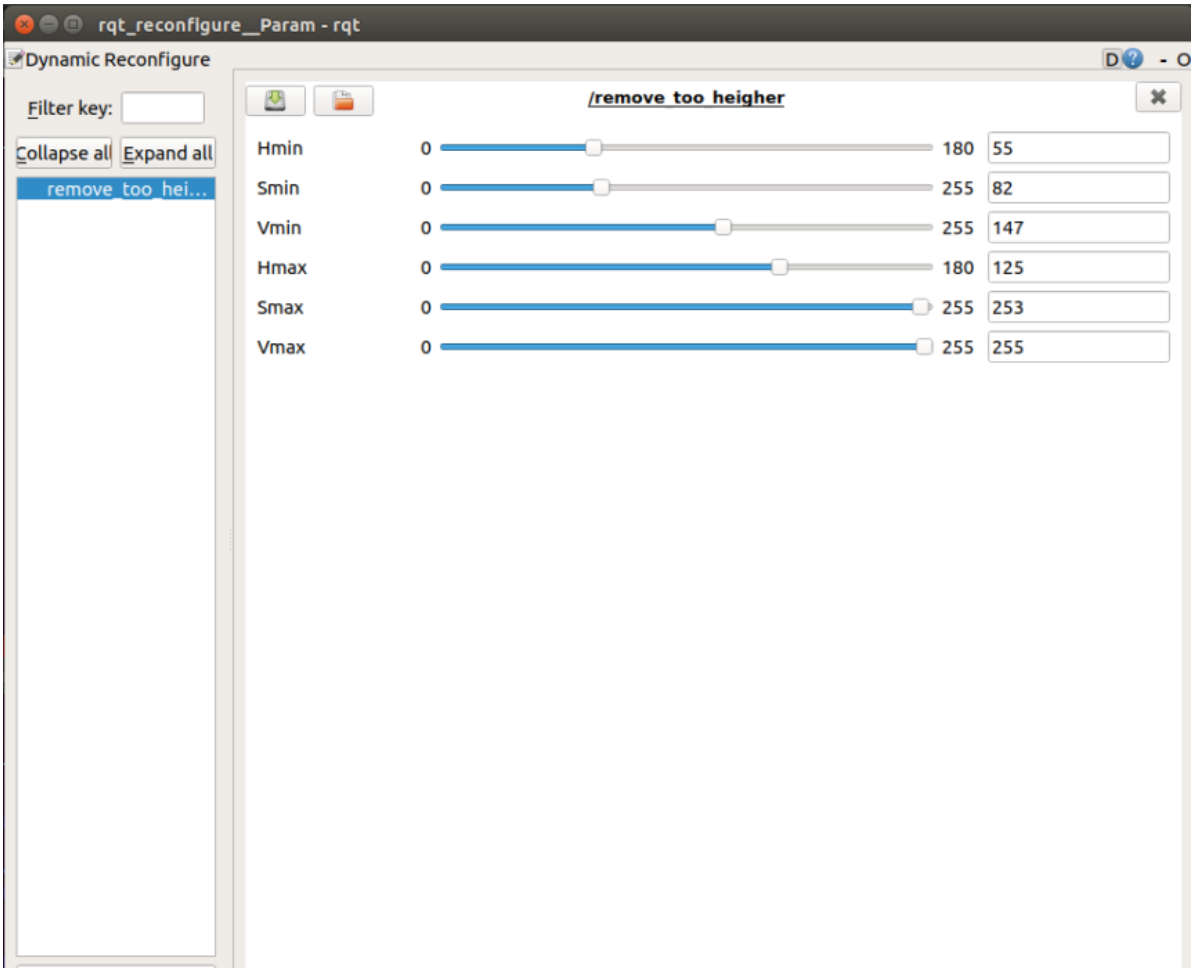
- [i] or [I]: Enter the color recognition mode, directly load the HSV value calibrated by the last program for recognition;
- 【r】or 【R】: reset program parameters, enter color selection mode, select a certain area of the color block with the mouse, obtain the HSV value of this area, release the mouse to enter color recognition mode

- Spacebar: start to grab highly abnormal color blocks

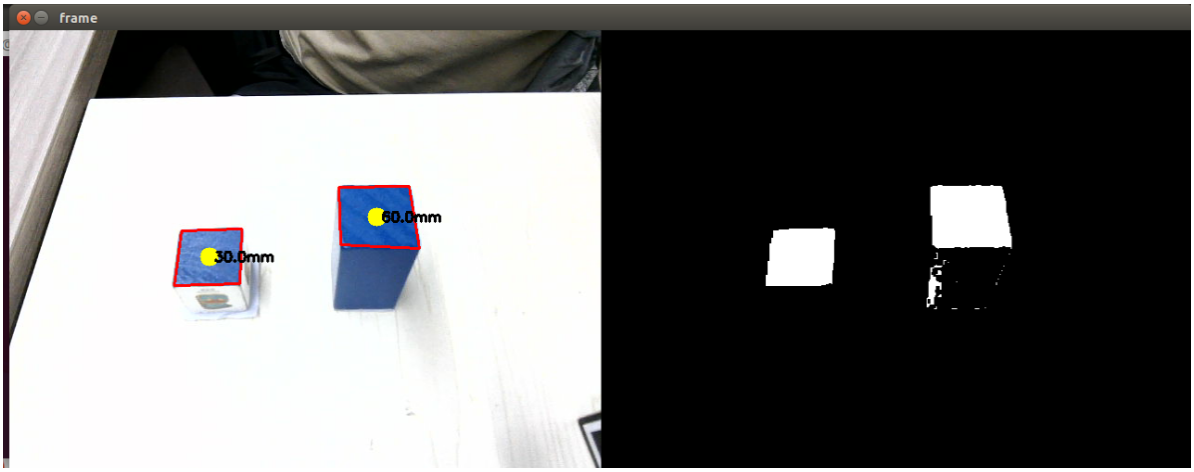
After entering color recognition mode, if the current HSV value still cannot filter out other colors, you can use the dynamic parameter adjuster to fine-tune HSV. Enter the following command in the terminal to start the dynamic parameter adjuster.

```
rosrun rqt_reconfigure rqt_reconfigure
```

You can modify the HSV value through the slider

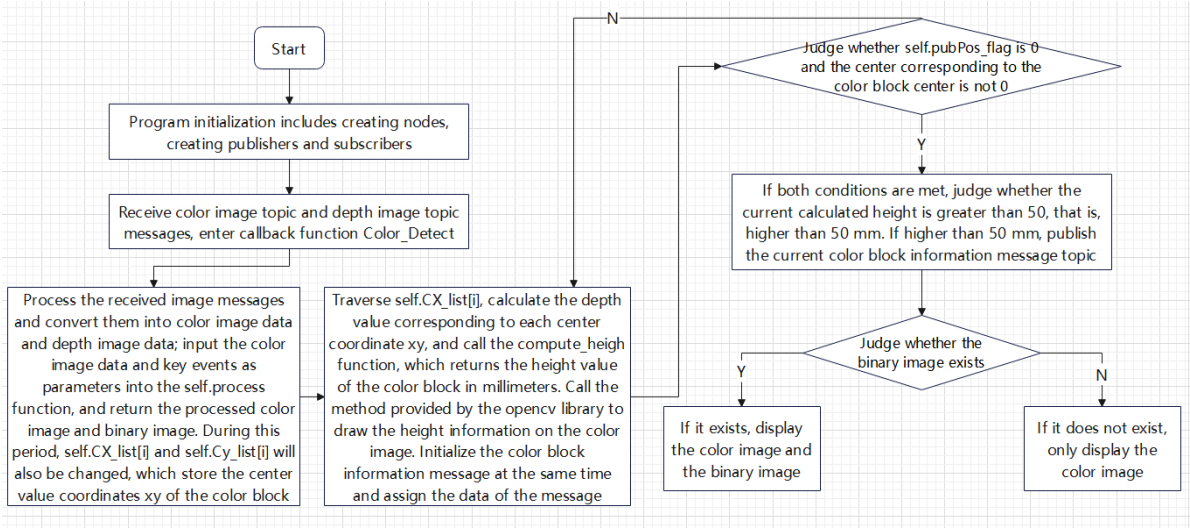


As shown in the figure below, when the image on the left (binarization) only shows the only recognized color, click the color image box and press the spacebar. The robotic arm will lower its claws to grab highly abnormal color blocks.



3. Program flow chart

AnomalyHeightRemoval.py



4. Core code analysis

4.1. AnomalyHeightRemoval.py

Code path:

`/home/jetson/dofbot_pro_ws/src/dofbot_pro_color/scripts/AnomalyHeightRemoval.py`

Import necessary libraries,

```
import rospy
import numpy as np
from sensor_msgs.msg import Image
import message_filters
from std_msgs.msg import Float32,Bool
import os
from cv_bridge import CvBridge
import cv2 as cv
encoding = ['16UC1', '32FC1']
import time
#Import custom image processing library
from height_measurement import *
#Import dynamic parameter service library
from dynamic_reconfigure.server import Server
from dynamic_reconfigure.client import Client
import rospkg
#Import color HSV configuration file
from dofbot_pro_color.cfg import ColorHSVConfig
import math
from dofbot_pro_info.msg import *
#Import transforms3d library for processing transformations in three-dimensional space, performing conversions between quaternions, rotation matrices and Euler angles, supporting three-dimensional geometric operations and coordinate conversions
import transforms3d as tfs
```

```
#Import transformations to process and calculate transformations in three-
dimensional space, including conversions between quaternions and Euler angles
import tf.transformations as tf
```

Initialize program parameters, create publishers and subscribers,

```
def __init__(self):
    nodeName = 'remove_too_heigher'
    rospy.init_node(nodeName)
    #The robot arm recognizes the posture of the machine code
    self.init_joints = [90.0, 120, 0.0, 0.0, 90, 90]
    #Create a publisher that publishes color block information
    self.pub_ColorInfo = rospy.Publisher("PosInfo", AprilTagInfo, queue_size=1)
    #Create a publisher that publishes the target angle of the robotic arm
    self.pubPoint = rospy.Publisher("TargetAngle", ArmJoint, queue_size=1)
    #Create a subscriber to subscribe to gesture recognition results
    self.grasp_status_sub = rospy.Subscriber('grasp_done', Bool,
self.GraspStatusCallback, queue_size=1)
    #Create two subscribers to subscribe to the color image topic and the depth
image topic
    self.depth_image_sub =
message_filters.Subscriber('/camera/depth/image_raw',Image)
    self.rgb_image_sub =
message_filters.Subscriber('/camera/color/image_raw',Image)
    #Synchronize the time of color and depth image subscription messages
    self.TimeSynchronizer =
message_filters.ApproximateTimeSynchronizer([self.rgb_image_sub,self.depth_image
_sub],10,0.5)
    #The callback function TagDetect that handles the synchronization message is
connected to the subscribed message so that it can be automatically called when a
new message is received
    self.TimeSynchronizer.registerCallback(self.Color_Detect)
    #Save the xy coordinates of the center of the color block
    self.y = 0 #320
    self.x = 0 #240
    #Create a bridge for converting color and depth image topic message data to
image data
    self.rgb_bridge = CvBridge()
    self.depth_bridge = CvBridge()
    #Store the x and y values ••of the center value of the identified color
block
    self.CX_list = []
    self.CY_list = []
    #Initialize region coordinates
    self.Roi_init = ()
    #Initialize HSV values
    self.hsv_range = ()
    #Initialize the information of the recognized 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'
    #Initialize state value
    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 xy coordinates of mouse selection
    self.Mouse_XY = (0, 0)
    #Path to the default HSV threshold file, which stores the last saved HSV
value
    self.hsv_text = rospkg.RosPack().get_path("dofbot_pro_color") +
"/scripts/colorHSV.text"
    #Load the color HSV configuration file and configure the dynamic parameter
regulator
    Server(ColorHSVConfig, self.dynamic_reconfigure_callback)
    self.dyn_client = Client(nodeName, timeout=60)
    exit_code = os.system('rosservice call /camera/set_color_exposure  50')
    #The center xy coordinates of the currently identified color block and the
radius of the minimum circumscribed circle of the color block
    self.cx = 0
    self.cy = 0
    self.circle_r = 0
    #The flag for publishing color block information, True means that the color
block information can be published
    self.pubPos_flag = False
    #Initialize the height value of the color block
    self.heigh = 0.0
    #The current position and posture of the end of the robot arm
    self.CurEndPos = [-0.006,0.116261662208,0.0911289015753,-1.04719,-0.0,0.0]
    # Camera built-in parameters
    self.camera_info_K = [477.57421875, 0.0, 319.3820495605469, 0.0,
477.55718994140625, 238.64108276367188, 0.0, 0.0, 1.0]
    #The rotation transformation matrix of the end of the robotic arm and the
camera describes the relative position and posture between the two
    self.EndToCamMat =
np.array([[1.00000000e+00,0.00000000e+00,0.00000000e+00,0.00000000e+00],
                                  [0.00000000e+00,7.96326711e-04,9.99999683e-
01,-9.90000000e-02],
                                  [0.00000000e+00,-9.99999683e-01,7.96326711e-
04,4.90000000e-02],

[0.00000000e+00,0.00000000e+00,0.00000000e+00,1.00000000e+00]])
```

Mainly look at the image processing function Color_Detect,

```
def Color_Detect(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 the
keyboard event action
    result_frame, binary = self.process(result_image,action)
    print("self.CX_list: ",self.CX_list)
    print("self.CY_list: ",self.CY_list)
    # Traverse self.CX_list
    for i in range(len(self.CX_list)):
        cx = self.CX_list[i]
        cy = self.CY_list[i]
       #Calculate the depth value corresponding to the center coordinate
       dist = depth_image_info[cy,cx]/1000
       #Put the obtained color block center value coordinates and the depth
information corresponding to the center point into the compute_heigh function,
calculate the color block height value and perform a method on it, and change the
unit from meter to millimeter
       heigh = round(self.compute_heigh(cx,cy,dist),2) *1000
       heigh_msg = str(heigh) + 'mm'
       #Call opencv's putText function to draw the height information on the
color image
       cv.putText(result_frame, heigh_msg, (cx+5, cy+5),
cv.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2)
       #Create a color block information message and assign a value
       pos = AprilTagInfo()
       pos.x = cx
       pos.y = cy
       pos.z = dist
       #Judge whether self.pubPos_flag is 0 and the center corresponding to the
center of the color block is not 0
        if self.pubPos_flag == True and pos.z!=0:
            if i==len(self.CX_list) :
                self.pubPos_flag = False
            #If the calculated value is greater than 50, that is, the height is
higher than 5 cm, then publish the message of the topic of color block
information
            if heigh>50:
                self.pub_ColorInfo.publish(pos)
                self.pubPos_flag = False
    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 so, 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:
            rgb_img, binary, self.CX_list,self.CY_list =
self.color.ShapeRecognition(rgb_img, self.hsv_range)
            #The flag for determining dynamic parameter updates. 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].