Color block shape height abnormality 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 is started, place the color blocks of the same color block in the image. After the color is selected, the height of each color block will be calculated in the image. After pressing the space bar, the robot will lower its claws to grab the color block higher than the set height and place it at the set position. After the placement is completed, the robot returns to the recognized posture.

2. Start and operate

2.1. Start command

Enter the following command in the terminal to start,

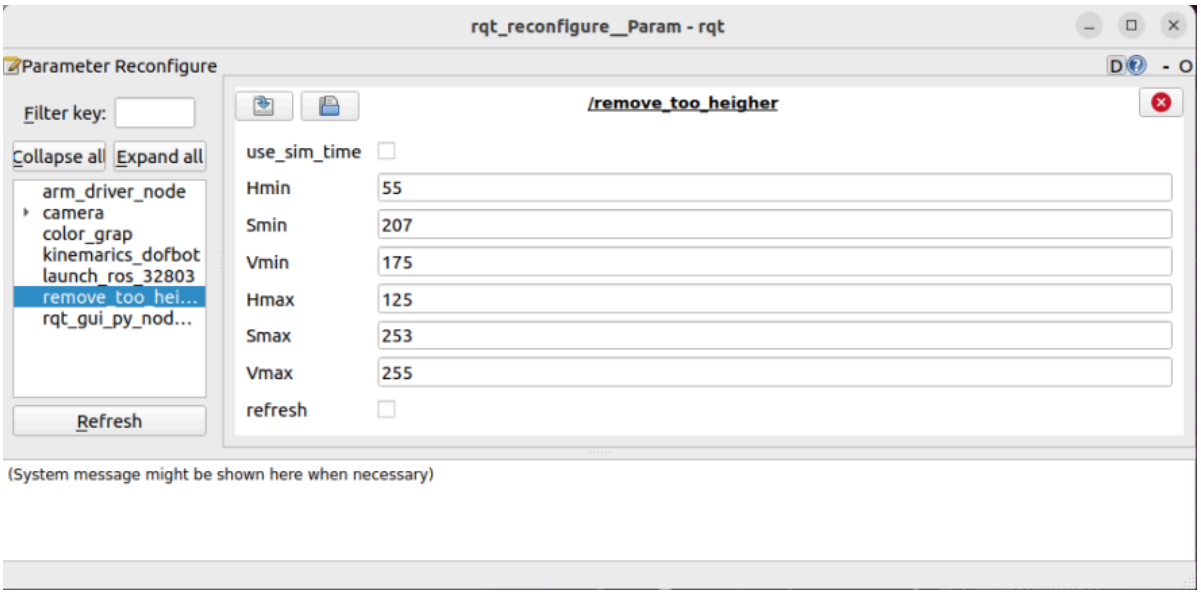
```
#Start the camera:
ros2 launch orbbec_camera dabai_dcw2.launch.py
#Start the underlying control:
ros2 run dofbot_pro_driver arm_driver
#Start the inverse program:
ros2 run dofbot_pro_info kinemarics_dofbot
#Start the robot arm grasping program:
ros2 run dofbot_pro_driver grasp
#Start the color block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

2.2. Operation

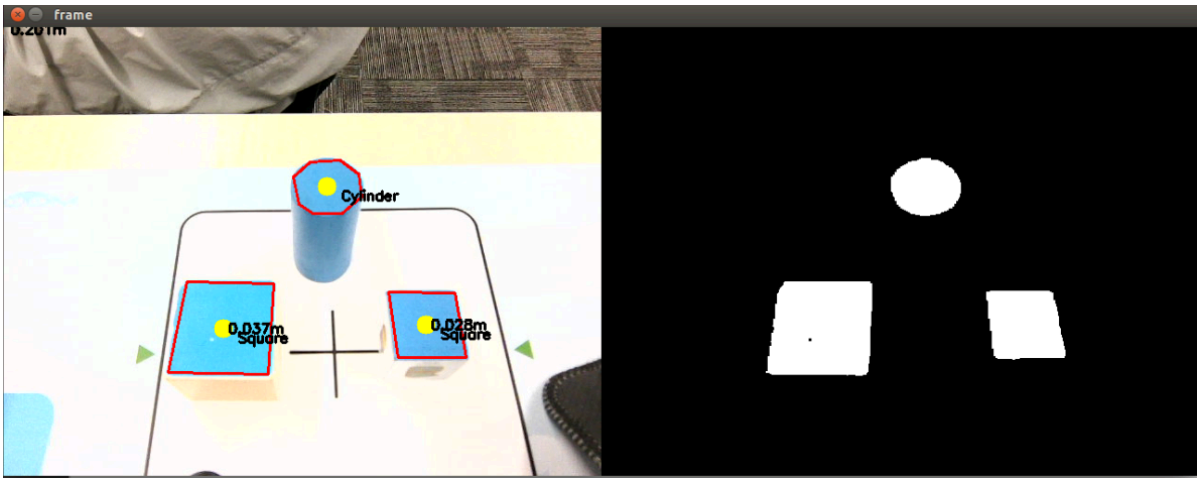
After the program is started, place blocks of the same color in the image, click on one of the blocks with the mouse, and get the HSV value; after releasing the mouse, a binary image will appear on the right. To make all the blocks in the color image appear in the binary image, if all blocks cannot be displayed, you need to use the dynamic parameter regulator. Start the dynamic parameter regulator with the following command,

```
ros2 run rqt_reconfigure rqt_reconfigure
```

Slide the slider to fine-tune the HSV value,

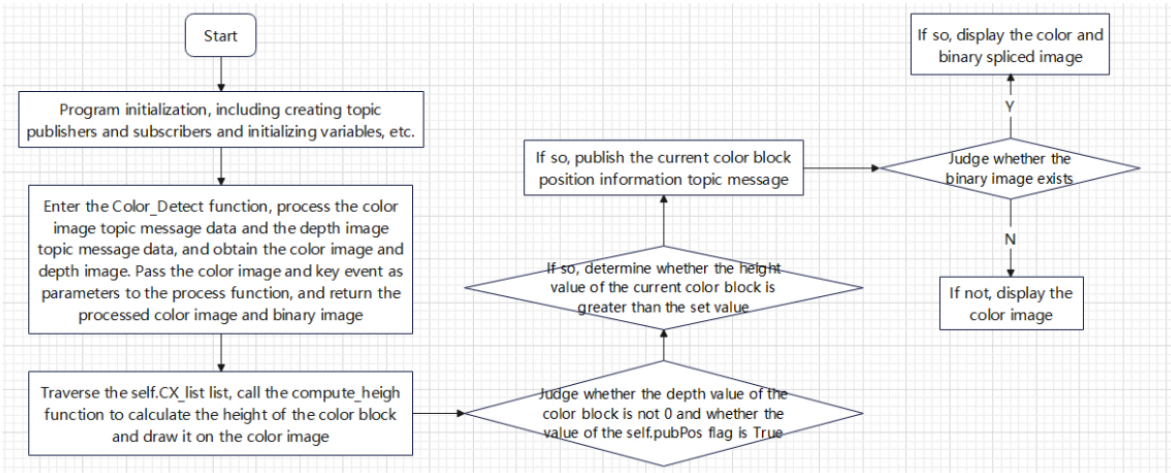


After the HSV value is debugged, click the color image frame and press the space bar. The robot will grab the block with a height higher than the set value and place it at the set position.



3. Program flow chart

shape_height.py



4. Core code analysis

4.1. shape_height.py

Code path:

```
/home/jetson/dofbot_pro_ws/src/dofbot_pro_color/dofbot_pro_color/shape_height.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.height_measurement import *

from dofbot_pro_interface.srv import *
from dofbot_pro_interface.msg import *
```

Program initialization, creating publishers, subscribers, clients, etc.,

```
def __init__(self):
     super().__init__('remove_too_heigher')
     self.declare_param()
     self.target_servox=90
     self.window_name = "depth_image"
     self.target_servoy=45
     self.pr_time = time.time()

     self.init_joints = [90.0, 120.0, 0.0, 0.0, 90.0, 90.0]


     self.pubPoint = self.create_publisher(ArmJoint, "TargetAngle", 1)
     self.grasp_status_sub = self.create_subscription(Bool, 'grasp_done',
self.GraspStatusCallback, 1)
     self.pub_ColorInfo = self.create_publisher(AprilTagInfo, "PosInfo", 1)

     # ROS2 subscribers (message synchronization) 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.Color_Detect)

    self.client = self.create_client(Kinemarics, 'dofbot_kinemarics')
    #Create a bridge for color and depth image topic message data to image data
    self.rgb_bridge = CvBridge()
    self.depth_bridge = CvBridge()
    #Store the list of x and y values ••of the center value of the recognized
color block
    self.CX_list = []
    self.CY_list = []
    #Initialize region coordinates
    self.Roi_init = ()
    #Initialize HSV values
    self.hsv_range = ()
    #Dynamic parameter adjustment flag, if True, dynamic parameter adjustment is
performed
    self.dyn_update = True
    #Mouse selection flag
    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 mouse selected xy coordinates
    self.Mouse_XY = (0, 0)
    #Default HSV threshold file path, 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 flag for publishing color block information. True means that 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
    self.CurEndPos = [0,0,0,0,0,0]
    #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]
    #Rotation transformation matrix between the end of the robot arm and the
camera, describing 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]])
```

```
    #Get the end position of the current posture of the robot
    self.get_current_end_pos()
    print("Target shape: ",self.color.target_shape)
    #Initialize the target color block shape
    self.color.target_shape = "Square" # "Rectangle" ,"Cylinder"
```

Mainly look at the image processing function Color_Detect,

```
def Color_Detect(self,color_frame,depth_frame):
    #rgb_image
    #Receive the 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 the 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 pass the
keyboard event action at the same time
    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 coordinates
        dist = depth_image_info[cy,cx]/1000
        #Pass 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 meter
        heigh = round(self.compute_heigh(cx,cy,dist),3)
        heigh_msg = str(heigh) + 'm'
        #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 it
        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 color block height is higher than 3 cm, publish the color
block topic message data
            if heigh>0.03:
```

```
                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)
```

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].