Volume measurement

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, the program will identify the shape of the selected color block, obtain the relevant data of the color block, and measure the volume of the color block.

2. Start and operate

2.1. Start command

Open four terminals and enter the following command to start,

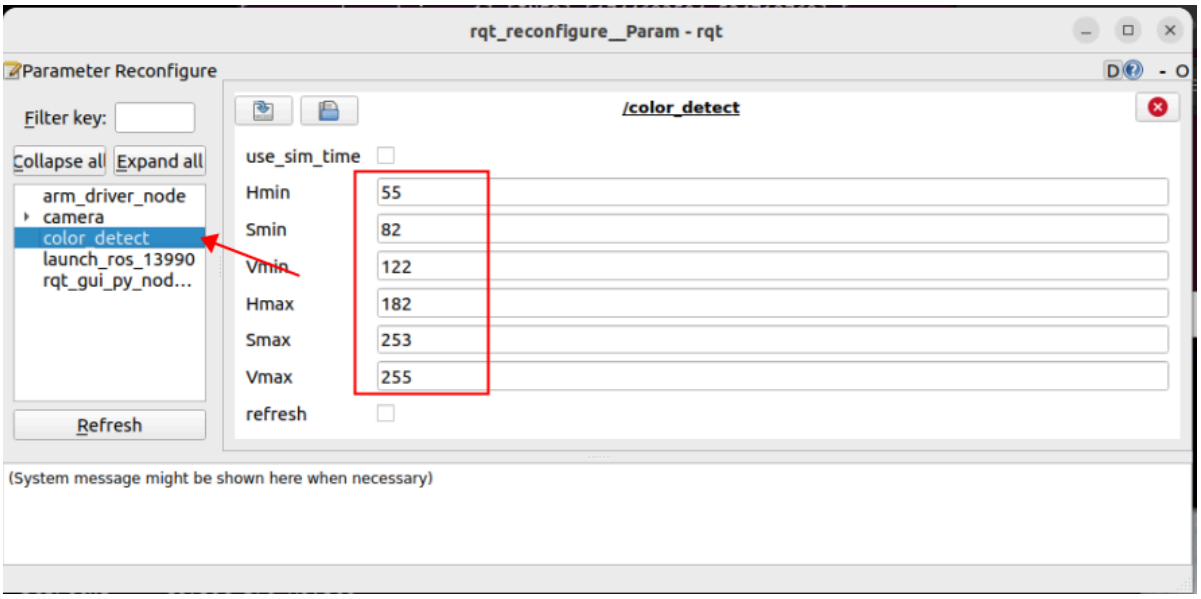
```
#Start the camera
ros2 launch orbbec_camera dabai_dcw2.launch.py
#Start the underlying control robot
ros2 run dofbot_pro_driver arm_driver
#Start the inverse solution program
ros2 run dofbot_pro_info kinemarics_dofbot
#Start the volume test program
ros2 run dofbot_pro_driver calculate_volume
```

2.2. Operation

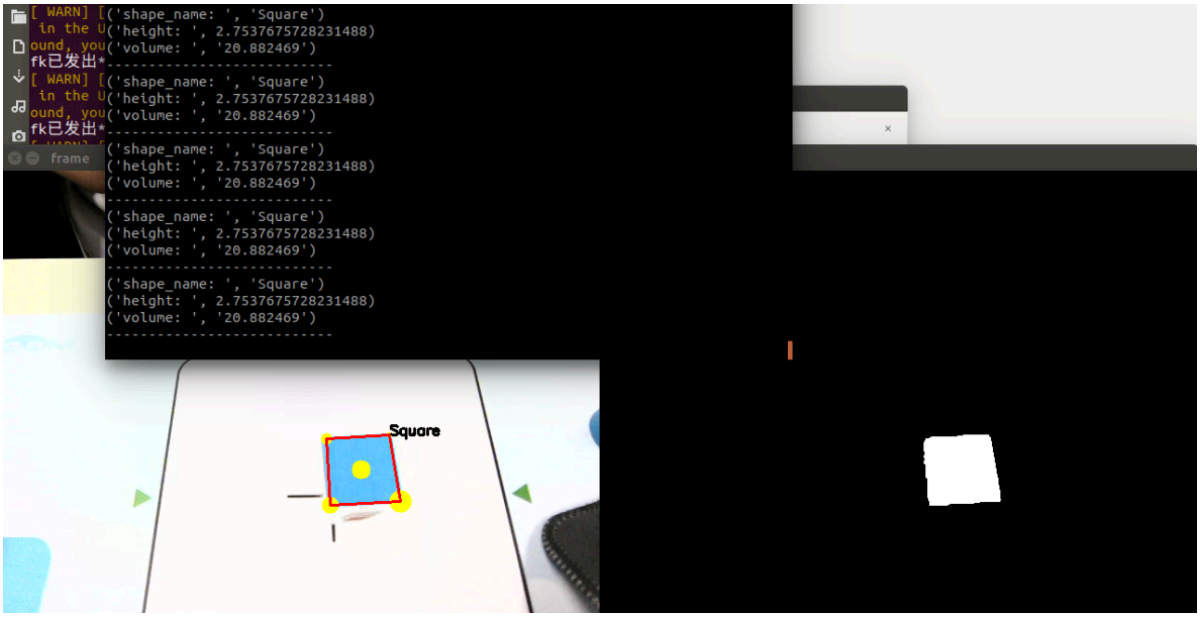
After the program starts, put the color block into the image. Use the mouse to select an area in the color block and get the HSV value. If the HSV value of the selected color cannot correctly identify the shape of the color block, you may need to fine-tune the HSV value. Enter the following command in the terminal to start the dynamic parameter regulator.

```
ros2 run rqt_reconfigure rqt_reconfigure
```

You can modify the HSV value through the slider.

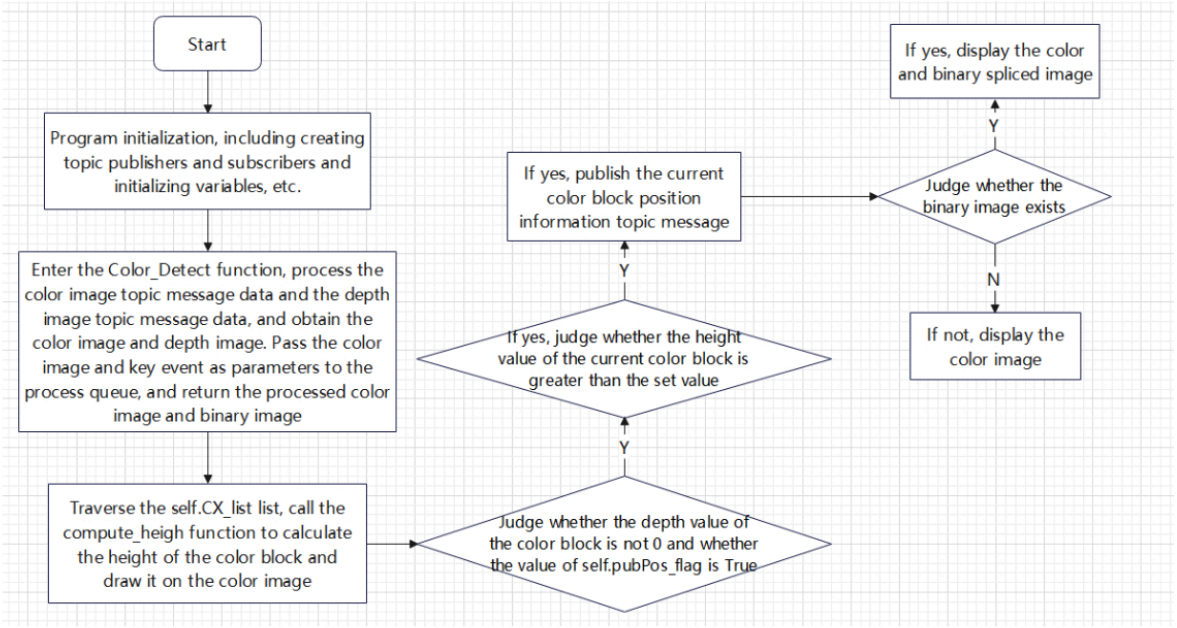


The program filters out other colors through the HSV value and only retains colors that meet the HSV value range. The program will identify the shape of the color block, and calculate the data of the color block based on the center point of the color block, and calculate the volume of the color block based on the identified color block shape.



3. Program flow chart

calculate_volume.py



4. Core code analysis

Code path:

```
/home/jetson/dofbot_pro_ws/src/dofbot_pro_driver/dofbot_pro_driver/calculate_volume.py
```

Import necessary libraries,

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import cv2 as cv
import rclpy
from rclpy.node import Node
import numpy as np
from sensor_msgs.msg import Image
from message_filters import ApproximateTimeSynchronizer, Subscriber
from std_msgs.msg import Float32, Bool
import os
from cv_bridge import CvBridge
from rclpy.qos import QoSProfile, QoSReliabilityPolicy, QoSHistoryPolicy
from rcl_interfaces.msg import ParameterDescriptor
#Import custom database
from dofbot_pro_interface.msg import *
from dofbot_pro_interface.srv import *
#Import custom image processing library
from dofbot_pro_driver.astra_common import *
#Import transformations to process and calculate transformations in three-dimensional space, including conversions between quaternions and Euler angles
import tf_transformations as tf
#Import transforms3d library to process transformations in three-dimensional space, perform conversions between quaternions, rotation matrices and Euler angles, and support three-dimensional geometric operations and coordinate conversions
import transforms3d as tfs
encoding = ['16UC1', '32FC1']
```

Program initialization, creation of publishers, subscribers, etc.

```
def __init__(self):
        super().__init__('color_detect')
        #Load color HSV configuration file
        self.declare_param()
        #Robot recognizes color block posture
        self.init_joints = [90.0, 120, 0.0, 0.0, 90, 90]
        #Create a client and call the inverse solution service
        self.client = self.create_client(Kinemarics, "dofbot_kinemarics")
        #Create two subscribers to subscribe to color image topics and depth
image topics
        self.depth_image_sub = Subscriber(self, Image, '/camera/depth/image_raw')
        self.rgb_image_sub = Subscriber(self, Image, '/camera/color/image_raw')
        #Time synchronization of color and depth image subscription messages
        self.ts = ApproximateTimeSynchronizer([self.rgb_image_sub,
self.depth_image_sub],queue_size=10,slop=0.5)
        #Callback function ComputeVolume that handles synchronous messages. The
callback function is connected to the subscribed message so that it can be
automatically called when a new message is received
        self.ts.registerCallback(self.ComputeVolume)
        #Create a bridge 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 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'
        #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)
        #Default path of HSV threshold file, which stores the last saved HSV
value
        self.hsv_text =
"/home/jetson/dofbot_pro_ws/src/dofbot_pro_driver/dofbot_pro_driver/colorHSV.text
"
```

```
        #Center xy coordinates of the currently recognized color block and the
radius of the minimum circumscribed circle of the color block
        self.cx = 0
        self.cy = 0
        self.error = False
        self.circle_r = 0 #Prevent misidentification of other messy points
        #End position posture of the robot arm under the current posture
        self.CurEndPos = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
        #Camera internal 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 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 robot arm at the current posture
        self.get_current_end_pos()
        #Initialize the corner point coordinate list
        self.get_corner = [0,0,0,0,0,0]
```

Image processing function ComputeVolume,

```
def ComputeVolume(self,color_frame,depth_frame):
    #接收到彩色图像话题消息，把消息数据转换成图像数据
    #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)
    #接收到深度图像话题消息，把消息数据转换成图像数据
    #Receive the deep 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))
    #把得到的彩色图像，作为参数传入process中，并且同时传入键盘事件action
    # Pass the obtained color image as a parameter to the process, and also pass
it to the keyboard event action
    result_frame, binary= self.process(result_image,action)
    #判断当前色块的中心值的坐标是否不为0，如果成立，则说明有检测到色块
    #Judge whether the coordinates of the center value of the current color block
are not 0. If so, it means that a color block has been detected
    if self.color.shape_cx!=0 and self.color.shape_cy!=0:
        #判断色块的形状，如果是长方体或者圆柱，则需要计算角点坐标值
        #Judge the shape of the color block. If it is a cuboid or a cylinder, you
need to calculate the coordinates of the corner points
        if self.color.shape_name == "Rectangle" or self.color.shape_name ==
"Cylinder":
            x1 = self.get_corner[0]
            y1 = self.get_corner[1]
            z1 = depth_image_info[y1,x1]/1000
```

```
            if z1!=0:
                c1_pose_T = self.get_pos(x1,y1,z1)
            else:
                self.error = True
                print("z1 invalid Distance!")

            x2 = self.get_corner[2]
            y2 = self.get_corner[3]
            z2 = depth_image_info[y2,x2]/1000
            if z1!=0:
                c2_pose_T = self.get_pos(x2,y2,z2)
            else:
                self.error = True
                print("z2 invalid Distance!")


            x3 = self.get_corner[4]
            y3 = self.get_corner[5]
            z3 = depth_image_info[y3,x3]/1000
            if z1!=0:
                c3_pose_T = self.get_pos(x3,y3,z3)
            else:
                self.error = True
                print("z3 invalid Distance!")

            cx = self.color.shape_cx
            cy = self.color.shape_cy
            cz = depth_image_info[cy,cx]/1000
            if cz!=0:
                center_pose_T = self.get_pos(cx,cy,cz)
                #计算色块高度 Calculate the color block height
                height = center_pose_T[2]*100
                print("get height: ",height)

            if self.error!=True:
                # 定义两个点的坐标 Define the coordinates of two points
                point1 = np.array([c1_pose_T[0], c1_pose_T[1], c1_pose_T[2]])
                point2 = np.array([c2_pose_T[0], c2_pose_T[1], c2_pose_T[2]])
                point3 = np.array([c3_pose_T[0], c3_pose_T[1], c3_pose_T[2]])
                c_ponit = np.array([center_pose_T[0], center_pose_T[1],
center_pose_T[2]])
                #计算半径 Calculate Radius
                r = np.linalg.norm(point1 - c_ponit)
                # 计算欧几里得距离（矩形边长） Calculate Euclidean distance (rectangle
side length)
                distance1 = np.linalg.norm(point1 - point3)
                distance2 = np.linalg.norm(point3 - point2)

                #长方体的体积等与长乘宽乘高 The volume of a cuboid is equal to length
times width times height
                if self.color.shape_name == "Rectangle":
                    print("shape_name: ",self.color.shape_name)
                    print("distance1: ",distance1)
                    print("distance2: ",distance2)
                    print("height: ",height)
                    volume = distance1 * distance2 * height
```

```
                    print("volume: ",format(volume, 'f'))
                    print("----------------------------")
                #圆柱体的体积等与π乘半径的平方乘高 The volume of a cylinder is equal
to π times the radius squared times the height
                if self.color.shape_name == "Cylinder":
                    print("r: ",r)
                    print("height: ",height)
                    print("shape_name: ",self.color.shape_name)
                    volume = math.pi*r*r* height
                    print("volume: ",format(volume, 'f'))
                    print("----------------------------")

        if self.color.shape_name == "Square":
            print("shape_name: ",self.color.shape_name)
            cx = self.color.shape_cx
            cy = self.color.shape_cy
            cz = depth_image_info[cy,cx]/1000
            if cz!=0:
                center_pose_T = self.get_pos(cx,cy,cz)
                height = center_pose_T[2]*100
                print("height: ",height)
                #正方体的体积等于任一边长的三次方 The volume of a cube is equal to the
cube of the length of any side
                volume = height * height * height
                print("volume: ",format(volume, 'f'))
                print("----------------------------")

        self.error = False
    if len(binary) != 0: cv.imshow(self.windows_name, ManyImgs(1, ([result_frame,
binary])))
    else:
        cv.imshow(self.windows_name, result_frame)
```

Get the position function get_pos in the world coordinate system,

```
def get_pos(self,x,y,z):
    #进行坐标系的转换，最终得到点在世界坐标下的位置pose_T和位姿pose_R
    #Convert the coordinate system to get the position pose_T and pose pose_R of
the point in the world coordinate system
    camera_location = self.pixel_to_camera_depth((x,y),z)
    PoseEndMat = np.matmul(self.EndToCamMat,
self.xyz_euler_to_mat(camera_location, (0, 0, 0)))
    EndPointMat = self.get_end_point_mat()
    WorldPose = np.matmul(EndPointMat, PoseEndMat)
    pose_T, pose_R = self.mat_to_xyz_euler(WorldPose)
    return pose_T
```