Color Block Volume Measurement

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, the program will recognize the shape of the selected color block, obtain relevant data of the color block, and calculate the volume of the color block.

2. Startup and Operation

2.1. Startup Commands

Open four terminals separately and enter the following commands to start:

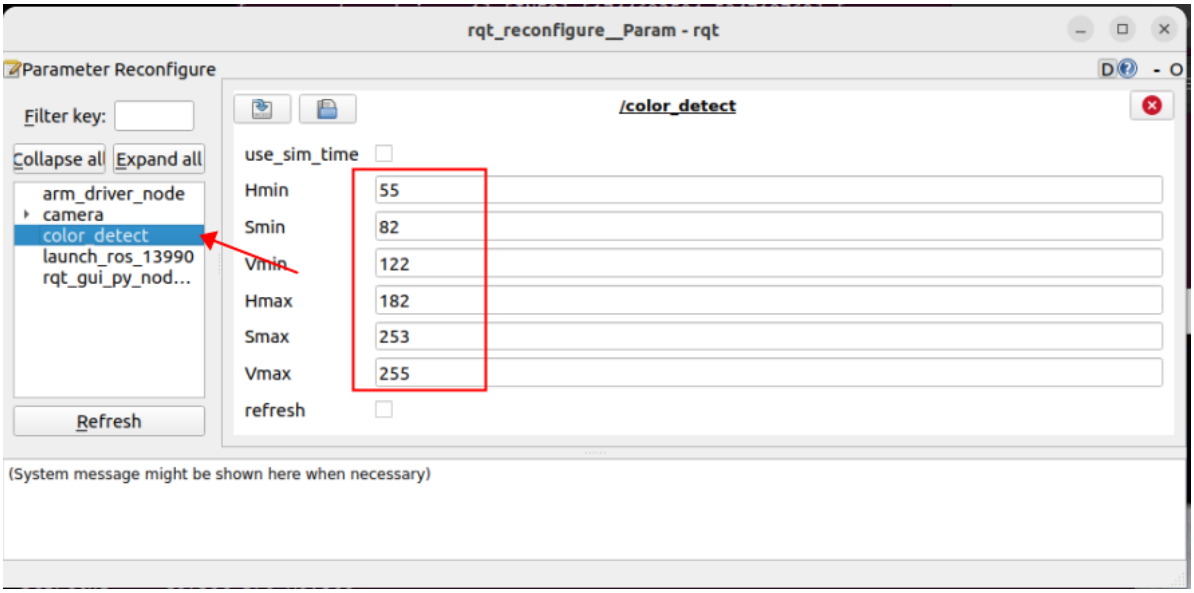
```
#Start camera
ros2 launch orbbec_camera dabai_dcw2.launch.py
#Start underlying control of robotic arm
ros2 run dofbot_pro_driver arm_driver
#Start inverse kinematics program
ros2 run dofbot_pro_info kinemarics_dofbot
#Start volume measurement program
ros2 run dofbot_pro_driver calculate_volume
```

2.2. Operation

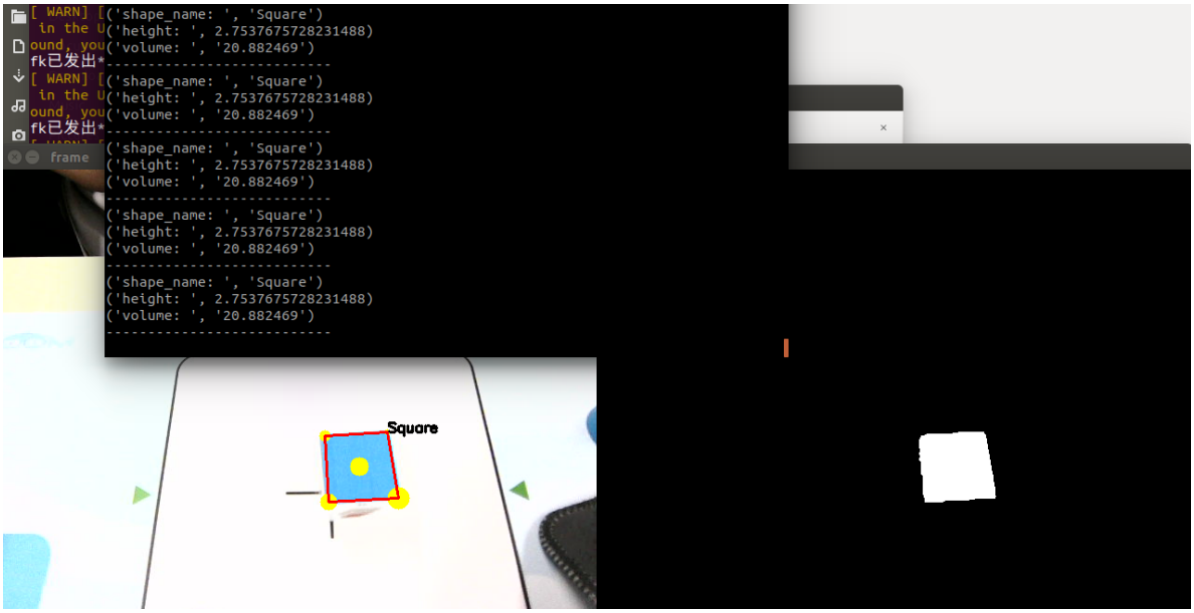
After the program starts, place the color block into the image. Use the mouse to select an area within the color block to obtain HSV values. If the obtained HSV values cannot correctly recognize the color block shape, you may need to fine-tune the HSV values. Enter the following command in the terminal to start the dynamic parameter tuner:

```
ros2 run rqt_reconfigure rqt_reconfigure
```

You can modify the HSV values through sliders:

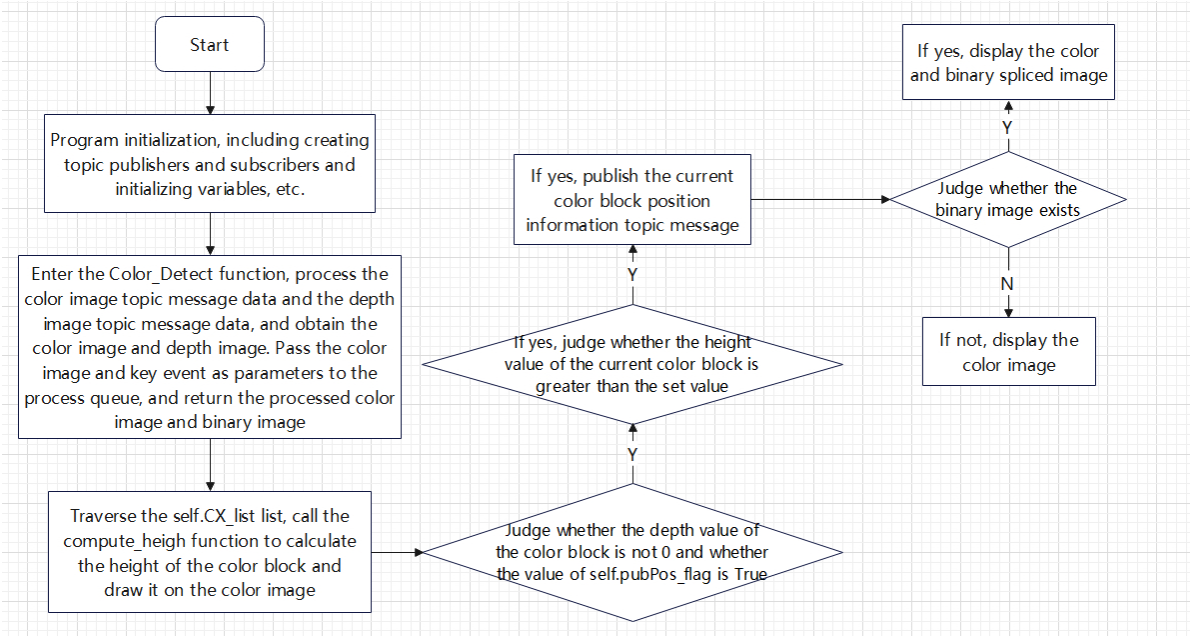


The program filters out other colors through HSV values, only retaining colors within the HSV value range. The program will recognize the shape of the color block, calculate the block's data based on its center point, and calculate the volume of the color block based on the recognized shape.



3. Program Flowchart

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 for processing and calculating transformations in 3D space, including conversions between quaternions and Euler angles
import tf_transformations as tf
#import transforms3d library for handling transformations in 3D space, performing conversions between quaternions, rotation matrices, and Euler angles, supporting 3D geometric operations and coordinate transformations
import transforms3d as tfs
encoding = ['16UC1', '32FC1']
```

Program initialization, create publishers, subscribers, etc.,

```
def __init__(self):
    super().__init__('color_detect')
    #Load color HSV configuration file
    self.declare_param()
    #Robotic arm recognizes color block pose
    self.init_joints = [90.0, 120, 0.0, 0.0, 90, 90]
    #Create client to call inverse kinematics service
    self.client = self.create_client(Kinemarics, "dofbot_kinemarics")
    #Create two subscribers, subscribe to color image topic and depth image topic
    self.depth_image_sub = Subscriber(self, Image, '/camera/depth/image_raw')
    self.rgb_image_sub = Subscriber(self, Image, '/camera/color/image_raw')
    #Time-synchronize the subscribed messages from color and depth image topics
    self.ts = ApproximateTimeSynchronizer([self.rgb_image_sub,
self.depth_image_sub],queue_size=10,slop=0.5)
    #Callback function ComputeVolume for processing synchronized messages,
connect the callback function with the subscribed messages so that this function
is automatically called when new messages are received
    self.ts.registerCallback(self.ComputeVolume)
    #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 the center
x coordinate, center y coordinate, and minimum enclosing circle radius r of the
color block
    self.circle = (0, 0, 0)
    #Dynamic parameter adjustment flag, True means 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, this file stores the last saved HSV values
    self.hsv_text =
"/home/jetson/dofbot_pro_ws/src/dofbot_pro_driver/dofbot_pro_driver/colorHSV.tex
t"

    #Current recognized color block's center xy coordinates and color block's
minimum circumscribed circle radius
    self.cx = 0
    self.cy = 0
    self.error = False
    self.circle_r = 0 #Prevent misrecognition of other scattered points
    #End position and pose of robotic arm end in current pose
```

```
    self.CurEndPos = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
    #Camera intrinsic 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 robotic arm end and camera,
describing the relative position and pose between them
    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 pose of robotic arm in current pose
    self.get_current_end_pos()
    #Initialize corner coordinate list
    self.get_corner = [0,0,0,0,0,0]
```

Image processing function ComputeVolume,

```
def ComputeVolume(self,color_frame,depth_frame):
    #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)
    #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)
    #Check if the current color block's center coordinates are not 0, if true, it
means a color block is detected
    if self.color.shape_cx!=0 and self.color.shape_cy!=0:
        #Check the shape of color block, if it's rectangular or cylindrical,
need to calculate corner coordinate values
        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)
                #Calculate color block height
                height = center_pose_T[2]*100
                print("get height: ",height)

            if self.error!=True:
                # Define 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]])
                #Calculate radius
                r = np.linalg.norm(point1 - c_ponit)
                # Calculate Euclidean distance (rectangle edge length)
                distance1 = np.linalg.norm(point1 - point3)
                distance2 = np.linalg.norm(point3 - point2)

                #Volume of rectangular prism equals 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("----------------------------")
                 #Volume of cylinder equals π times radius squared times 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)
                #Volume of cube equals the cube of any edge length
                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)
```

Function to get position in world coordinate system get_pos,

```
def get_pos(self,x,y,z):
    #Perform coordinate system conversion, finally get the position pose_T and
pose_R of the point in world coordinates
    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
```