# Block Shape Abnormal Height Sorting

Before starting this function, you need to close the main program and APP processes. If you need to restart the main program and APP later, start them from the terminal,

```
bash ~/dofbot_pro/APP_DOFBOT_PRO/start_app.sh
```

## 1. Function Description

After the program starts, place blocks of the same color but different shapes in the image. After color selection is completed, the program will calculate the height of each block in the image. Press the space key, and the robotic arm will grasp blocks higher than the set height and place them at the designated position. After placement is complete, the robotic arm returns to the recognition posture.

## 2. Startup and Operation

### 2.1. Startup Commands

Enter the following commands in the terminal to start,

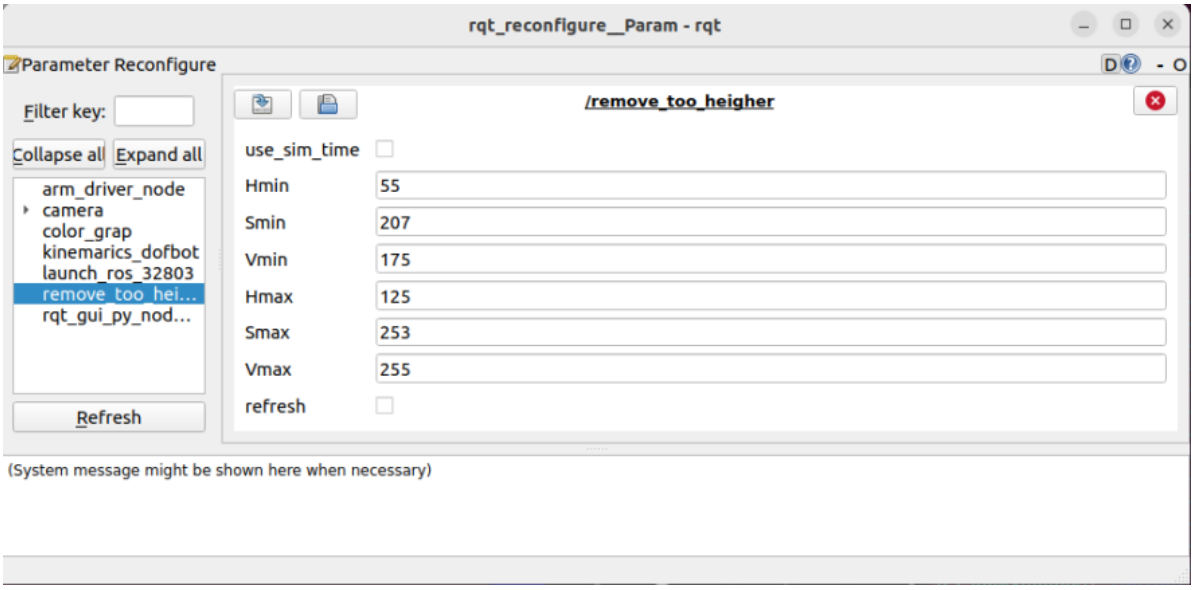
```
#Start camera:
ros2 launch orbbec_camera dabai_dcw2.launch.py
#Start underlying control:
ros2 run dofbot_pro_driver arm_driver
#Start inverse kinematics program:
ros2 run dofbot_pro_info kinemarics_dofbot
#Start robotic arm grasping program:
ros2 run dofbot_pro_driver grasp
#Start color block shape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

### 2.2. Operation

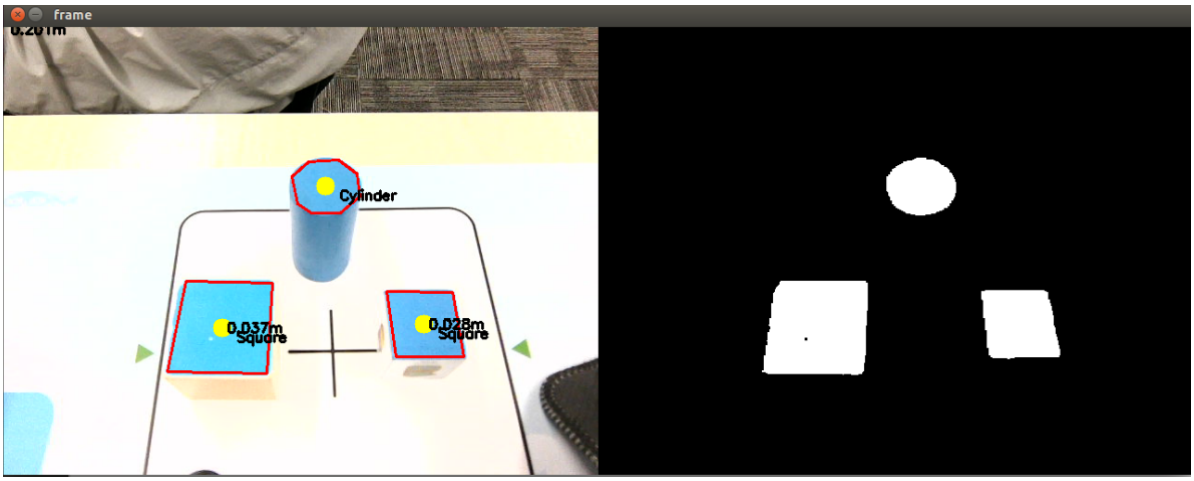
After the program starts, place color blocks of the same color in the image. Use the mouse to click and select an area of one of the color blocks to obtain HSV values. After releasing the mouse, a binary image will appear on the right side. Ensure that all color blocks in the color image are displayed in the binary image. If not all color blocks can be displayed, you need to use the dynamic parameter tuner. Start the dynamic parameter tuner with the following command,

```
ros2 run rqt_reconfigure rqt_reconfigure
```

Slide the sliders to fine-tune HSV values,

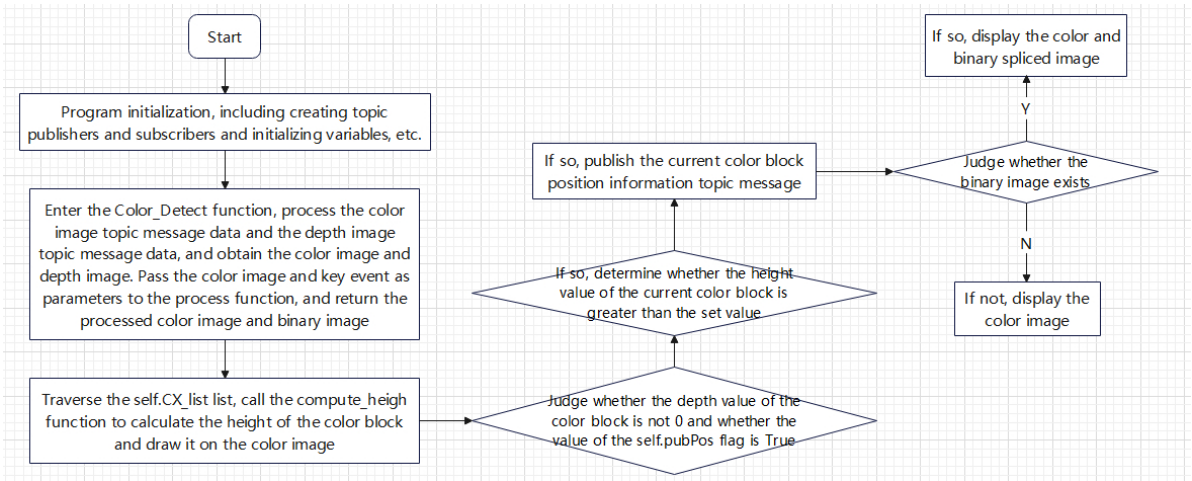



After the HSV values are adjusted, click on the color image frame and press the space key. The robotic arm will grasp blocks higher than the set value and place them at the designated position.




# 3. Program Flowchart

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, and clients,

```
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 bridges for converting color and depth image topic message data to
image data
        self.rgb_bridge = CvBridge()
        self.depth_bridge = CvBridge()
        #List to store x and y values of the center of detected color blocks
        self.CX_list = []
        self.CY_list = []
        #Initialize region coordinates
        self.Roi_init = ()
        #Initialize HSV values
        self.hsv_range = ()
        #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'
        #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 path of HSV threshold file, this file stores the last saved HSV
values
        self.hsv_text = rospkg.RosPack().get_path("dofbot_pro_color") +
"/scripts/colorHSV.text"
        #Load color HSV configuration file, configure dynamic parameter tuner
        Server(ColorHSVConfig, self.dynamic_reconfigure_callback)
        self.dyn_client = Client(nodeName, timeout=60)
        exit_code = os.system('rosservice call /camera/set_color_exposure  50')
        #Flag for publishing color block information, True means color block
information can be published
        self.pubPos_flag = False
        #Initialize height value of color block
        self.heigh = 0.0
        #Current position and pose of robotic arm end effector
        self.CurEndPos = [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 effector 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 current position and pose of robotic arm end effector
```

```
    self.get_current_end_pos()
    print("Target shape: ",self.color.target_shape)
    #Initialize target color block shape
    self.color.target_shape = "Square" # "Rectangle" ,"Cylinder"
```

Let's mainly look at the image processing function Color_Detect,

```
def Color_Detect(self,color_frame,depth_frame):
    #rgb_image
    #Receive color image topic message and 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 and 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 a parameter to process, along with 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 depth value corresponding to center coordinates
        dist = depth_image_info[cy,cx]/1000
        #Pass the obtained color block center coordinates and corresponding
depth information to compute_heigh function, calculate and get color block height
value and convert unit from meters to meters
        heigh = round(self.compute_heigh(cx,cy,dist),3)
        heigh_msg = str(heigh) + 'm'
        #Call opencv's putText function to draw 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 color block information message and assign values
        pos = AprilTagInfo()
        pos.x = cx
        pos.y = cy
        pos.z = dist
        #Check if self.pubPos_flag is 0 and the center corresponding to color
block center 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 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

For reference, see section 4.2 [grasp.py] in the tutorial [12. 3D Sorting and Grasping Course\1. AprilTag ID sorting].