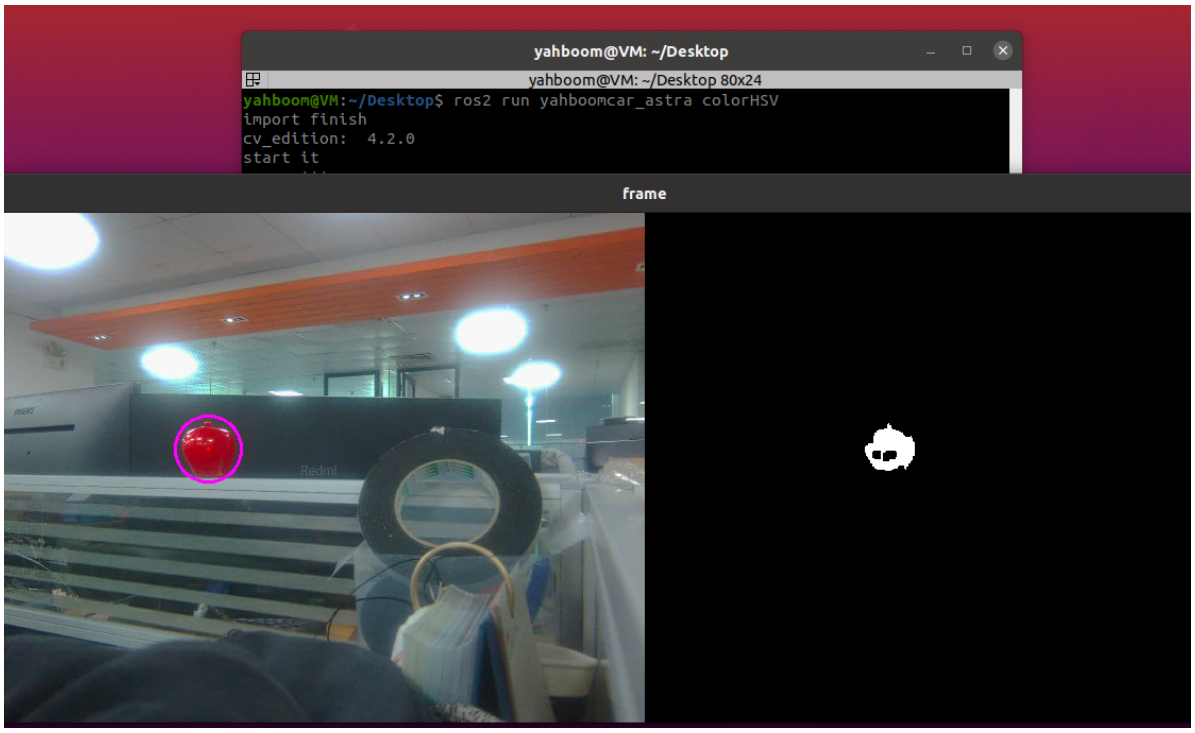
3. Color tracking

3.1. Program startup

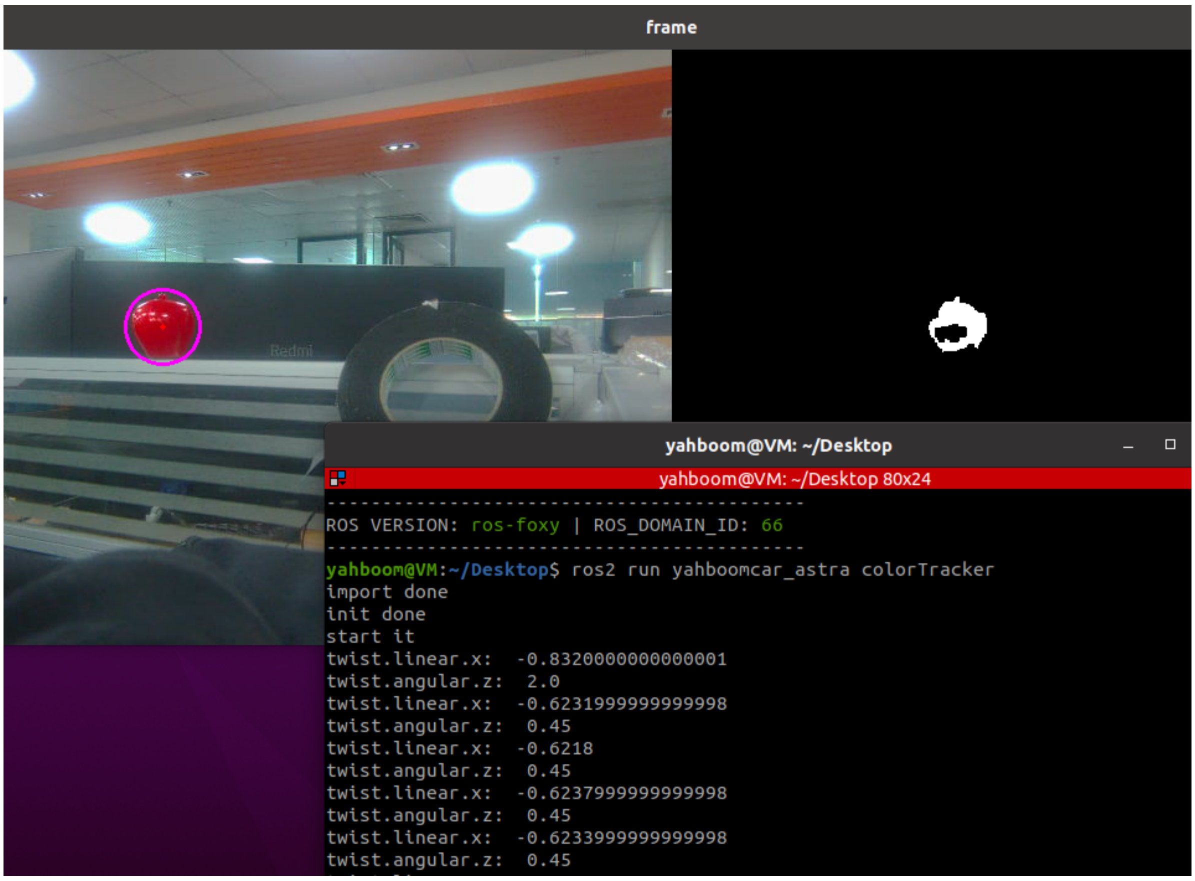
Terminal input,

```
ros2 launch orbbec_camera gemini2.launch.py
ros2 run yahboomcar_astra colorHSV
ros2 run yahboomcar_astra colorTracker
```



```
yahboom@VM:~/Desktop$ ros2 run yahboomcar_astra colorHSV
import finish
cv_edition:  4.2.0
start it
```

After successful startup, the above screen will be displayed. The program will load the HSV value at the beginning and then display the processed image. Press the [r] key to reselect the color, and use the mouse to frame the color that needs to be tracked. The selected area can only have one color. After selecting the color to be tracked and the program has finished processing the image, press the space bar to start tracking. The terminal that starts colorTracker will display,

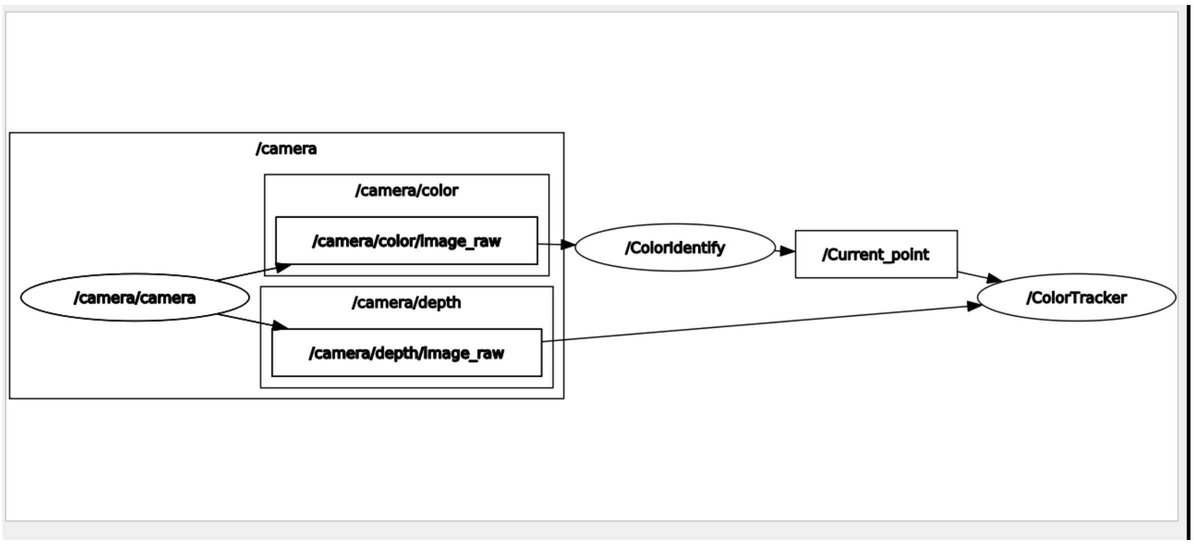


```
-------------------------------------------
ROS VERSION: ros-foxy | ROS_DOMAIN_ID: 66
-------------------------------------------
yahboom@VM:~/Desktop$ ros2 run yahboomcar_astra colorTracker
import done
init done
start it
twist.linear.x:  -0.8320000000000001
twist.angular.z:  2.0
twist.linear.x:  -0.6231999999999998
twist.angular.z:  0.45
twist.linear.x:  -0.6218
twist.angular.z:  0.45
twist.linear.x:  -0.6237999999999998
twist.angular.z:  0.45
twist.linear.x:  -0.6233999999999998
twist.angular.z:  0.45
```

This was originally intended to be run in combination with a robot. Programs that do not drive the chassis can only be verified by printing the speed that needs to be released. When the selected object is moved, the printed linear speed and angular speed will change. The speed topic here is /cmd_vel, If a robot chassis driver subscribes to this node, it can drive the robot.

Check the communication between nodes, terminal input,

```
ros2 run rqt_graph rqt_graph
```



3.2. Core code analysis

Code reference path,

```
~/orbbec_ws/src/yahboomcar_astra/yahboomcar_astra/colorHSV.py
~/orbbec_ws/src/yahboomcar_astra/yahboomcar_astra/colorTracker.py
```

3.2.1, colorHSV.py

This program mainly has the following functions:

- Subscribe to camera image data;
- Obtain keyboard and mouse events for switching modes and picking colors;
- Process images and publish the center coordinates of tracked objects and publish them

Part of the core code is as follows:

```
#Create publishers and subscribers
self.pub_position = self.create_publisher(Position,"/Current_point", 10)
self.sub_img
=self.create_subscription(Image,'/camera/color/image_raw',self.handleTopic,1)
#Subscribe to the image callback function and pass the image to the process
function
frame, binary =self.process(frame, action)
#Get keyboard and mouse events and get the value of hsv;
if action == 32: self.Track_state = 'tracking'
elif action == ord('i') or action == ord('I'): self.Track_state =
"identify"
elif action == ord('r') or action == ord('R'): self.Reset()
elif action == ord('q') or action == ord('Q'): self.cancel()
if self.Track_state == 'init':
cv.namedWindow(self.windows_name, cv.WINDOW_AUTOSIZE)
cv.setMouseCallback(self.windows_name, self.onMouse, 0)
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)
if self.Roi_init[0] != self.Roi_init[2] and self.Roi_init[1] !=
self.Roi_init[3]:
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'
#Calculate the value of the center coordinate, self.circle stores the xy value
rgb_img, binary, self.circle = self.color.object_follow(rgb_img, self.hsv_range)
#Publish center coordinate news
threading.Thread(target=self.execute, args=(self.circle[0], self.circle[1],
self.circle[2])).start()
def execute(self, x, y, z):
position = Position()
position.anglex = x * 1.0
position.angley = y * 1.0
position.distance = z * 1.0
self.pub_position.publish(position)
```

3.2.2, colorTracker.py

This program mainly has the following functions: receiving /Current_point and depth image topic data, calculating the speed, and then publishing the speed

degree data.

```
#Define the topic data that subscribers need to receive
self.sub_depth =
self.create_subscription(Image,"/camera/depth/image_raw",self.depth_img_Callback
, 1)
self.sub_position
=self.create_subscription(Position,"/Current_point",self.positionCallback,1)
#Define speed publisher
self.pub_cmdVel = self.create_publisher(Twist,'/cmd_vel',10)
#Two important callback functions, obtain the self.Center_x value and
distance_value
def positionCallback(self, msg):
def depth_img_Callback(self, msg):
#self.Center_x value and distance_value are calculated based on linear velocity
and angular velocity
self.execute(self.Center_x, distance_)
def execute(self, point_x, dist):
self.get_param()
if abs(self.prev_dist - dist) > 300:
self.prev_dist = dist
return
if abs(self.prev_angular - point_x) > 300:
self.prev_angular = point_x
return
if self.Joy_active == True: return
linear_x = self.linear_pid.compute(dist, self.minDist)
angular_z = self.angular_pid.compute(320, point_x)
if abs(dist - self.minDist) < 30: linear_x = 0
if abs(point_x - 320.0) < 30: angular_z = 0
twist = Twist()
if angular_z>2.0:
angular_z = 2.0
if angular_z<-2.0:
angular_z = -2.0
if linear_x > 1.0:
linear_x = 1.0
if linear_x <-1.0:
linear_x = -1.0
twist.angular.z = angular_z * 1.0
twist.linear.x = linear_x * 1.0
```