3. Color tracking

1. Program function description

After the program starts, the default tracking color is red, you can press r/R to enter the color selection mode, select a color by mouse, the cart will lock this color, press space bar to enter the tracking mode. The cart will keep a distance of 1 meter from the object being tracked, and always make sure the object being tracked stays in the center of the screen. Press q/Q to exit the program. After the joystick program is started, you can also pause/resume tracking by using the R2 key on the joystick.

2. Program code reference path

```
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/yahboomcar_astra/col
orHSV.py
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/yahboomcar_astra/col
orTracker.py
```

- colorHSV.py

  The main thing is to complete the image processing and calculate the center coordinates of the object being tracked.

- colorTracker.py

  The main purpose is to calculate the speed according to the center coordinate and depth information of the tracked object, and release the speed data to the chassis.

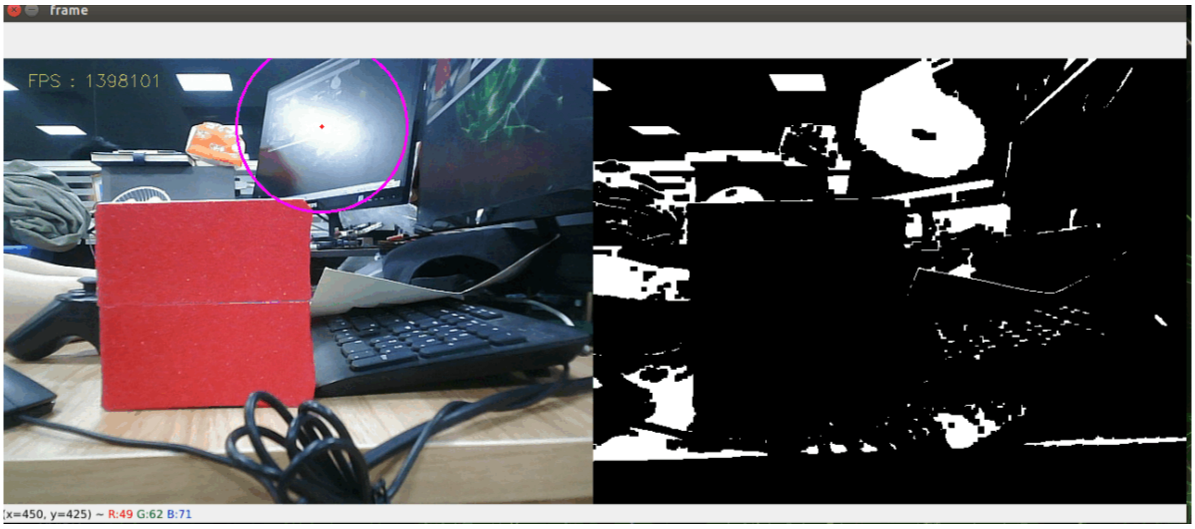
3. Program Startup

3.1. Startup Commands

After entering the docker container, according to the actual model, terminal input, the

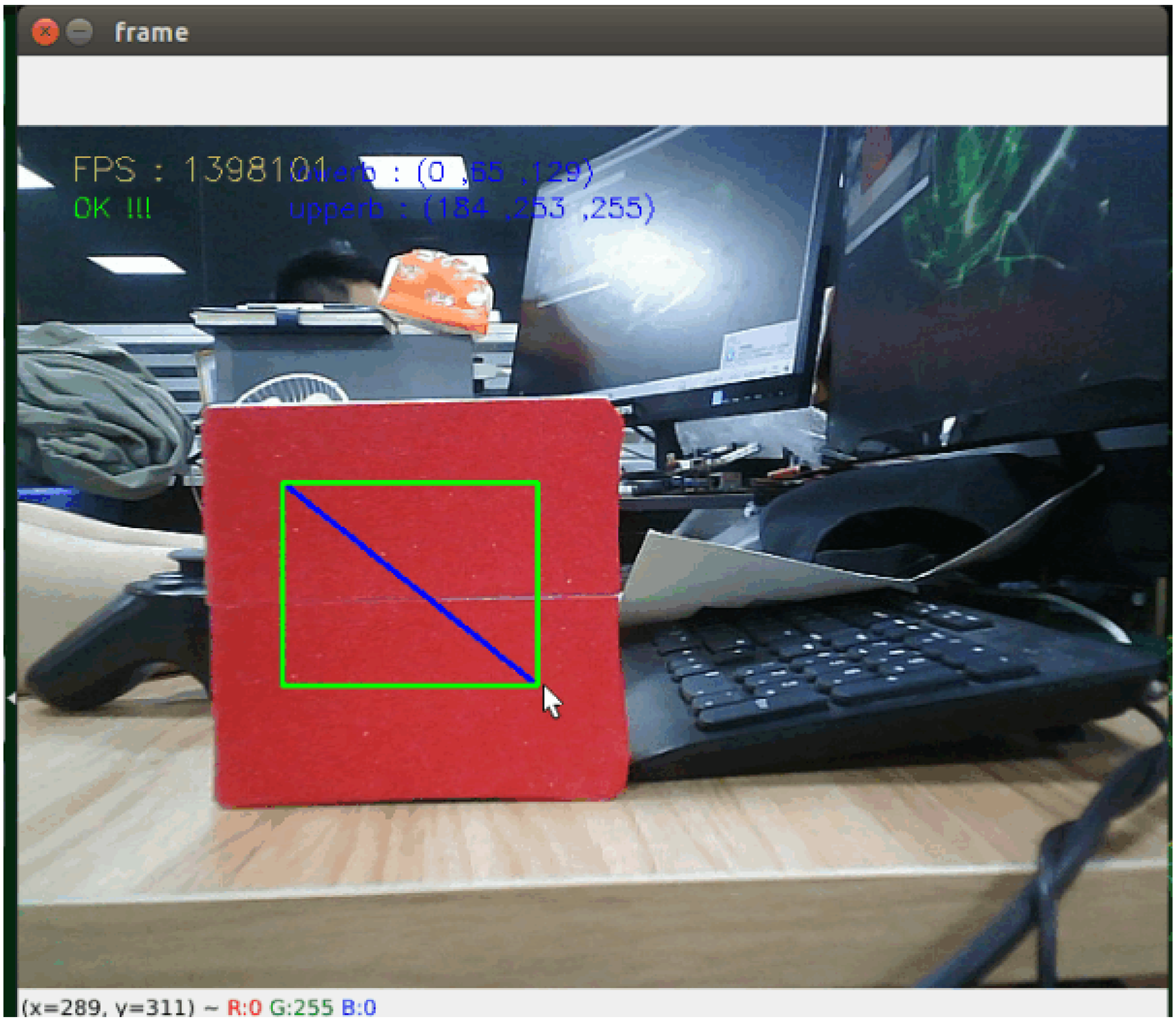
```
#启动小车底盘 #Start the trolley chassis
ros2 run yahboomcar_bringup Ackman_driver_R2
#启动颜色追踪程序 #Start the color tracking program
ros2 run yahboomcar_astra colorHSV
ros2 run yahboomcar_astra colorTracker
#启动深度相机 #Start the depth camera
ros2 launch astra_camera astra.launch.xml
#启动手柄节点 #Start the depth camera
ros2 run yahboomcar_ctrl yahboom_joy_R2
ros2 run joy joy_node
```

In the case of Tracking Red, for example, when the program starts, the following screen will appear.



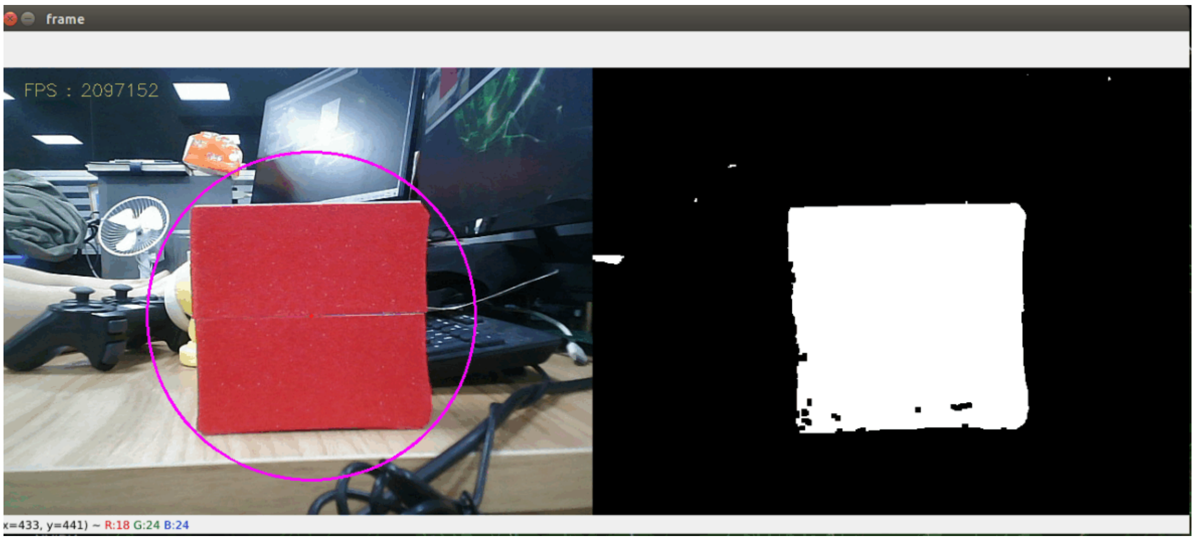
```
FPS : 1398101
(x=450, y=425) ~ R:49 G:62 B:71
```

Then press the r/R key on the keyboard to enter the color selection mode, and use the mouse to frame an area (the area can only have one of the colors), the



```
FPS : 1398101
OK !!!
lowerb : (0 ,65 ,129)
upperb : (184 ,253 ,255)
(x=289, y=311) ~ R:0 G:255 B:0
```

After the selection, the effect is as follows.

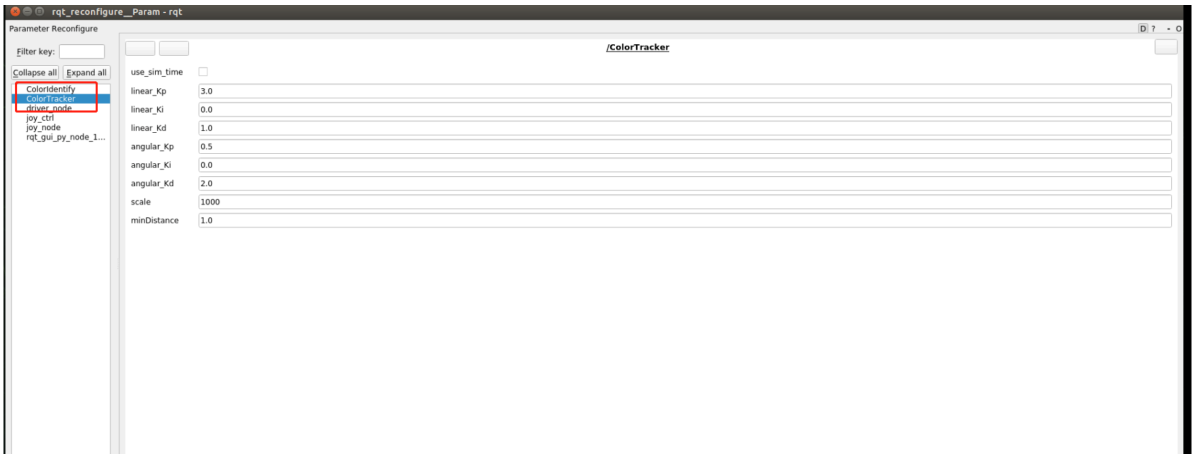


Then, press the spacebar to enter the tracking mode, move the object slowly, the cart will follow and keep a distance of 1 meter.

3.2. Dynamic Parameter Adjustment

Docker terminal input.

```
ros2 run rqt_reconfigure rqt_reconfigure
```



rqt_reconfigure__Param - rqt

Parameter Reconfigure

Filter key:

Collapse all | Expand all

ColorIdentify
ColorTracker
driver_node
joy_ctrl
joy_node
rqt_gui_py_node_1...

/ColorTracker

use_sim_time	☐
linear_Kp	3.0
linear_Ki	0.0
linear_Kd	1.0
angular_Kp	0.5
angular_Ki	0.0
angular_Kd	2.0
scale	1000
minDistance	1.0

After modifying the parameters, click on the GUI blank to write the parameter values. From the above figure, we can see that

- colorTracker is mainly responsible for calculating the speed, adjustable speed and distance related parameters.

4. Core Code

4.1, colorHSV.py

This program has the following main functions:

- Turn on the camera and get the image;
- Getting keyboard and mouse events for switching modes and taking colors;
- Process the image and publish the center coordinates of the tracked object and publish it.

Some of the core code is as follows.

```
#创建发布者，发布追踪物体的中心坐标
# Create a publisher to publish the center coordinates of the tracked object
```

```
self.pub_position = self.create_publisher(Position,"/Current_point", 10)
#获取键盘鼠标事件，得到hsv的值；
# Get keyboard and mouse events to 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'
#计算中心坐标的值，self.circle存放xy值
# Calculate the value of the center coordinate, self.circle holds the xy value.
rgb_img, binary, self.circle = self.color.object_follow(rgb_img, self.hsv_range)
#发布中心坐标的消息 #Publish a message with center coordinates
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)
```

4.2, colorTracker.py

This program has the following main functions: receive /Current_point and depth image topic data, calculate the speed magnitude, and then publish the speed data.

Part of the code is as follows.

```
#定义订阅者接收需要的话题数据
#Define subscribers to receive needed topic data
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)
#定义速度发布者 #Define the speed publisher
self.pub_cmdVel = self.create_publisher(Twist,'/cmd_vel',10)
#两个重要的回调函数，获取到self.Center_x值和distance_值
# two important callback functions to get the self.Center_x value and the
distance_ value
def positionCallback(self, msg):
def depth_img_Callback(self, msg):
#self.Center_x值和distance_值根据计算线速度，角速度
```

```
#self.Center_x value and distance_ value based on calculating linear velocity,
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
```