3、color tracking

1、Program Function Description

After the program starts, the default tracking color is red. You can press the r/R button to enter the color selection mode, select a color with the mouse, and the car will lock the color. Press the spacebar to enter the tracking mode. The car will maintain a distance of 1 meter from the tracked object and always ensure that the tracked object remains in the center of the screen. Press the q/Q key to exit the program. After the controller program starts, you can also pause/continue tracking by pressing the R2 key on the controller.

2、Program code reference path

```
~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/yahboomcar_astra/colorHSV
.py
~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/yahboomcar_astra/colorTra
cker.py
```

- colorHSV.py

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

- colorTracker.py

  Mainly based on the center coordinates and depth information of the tracked object, calculate the speed and publish the speed data to the car chassis

3、Program startup

3.1、start command

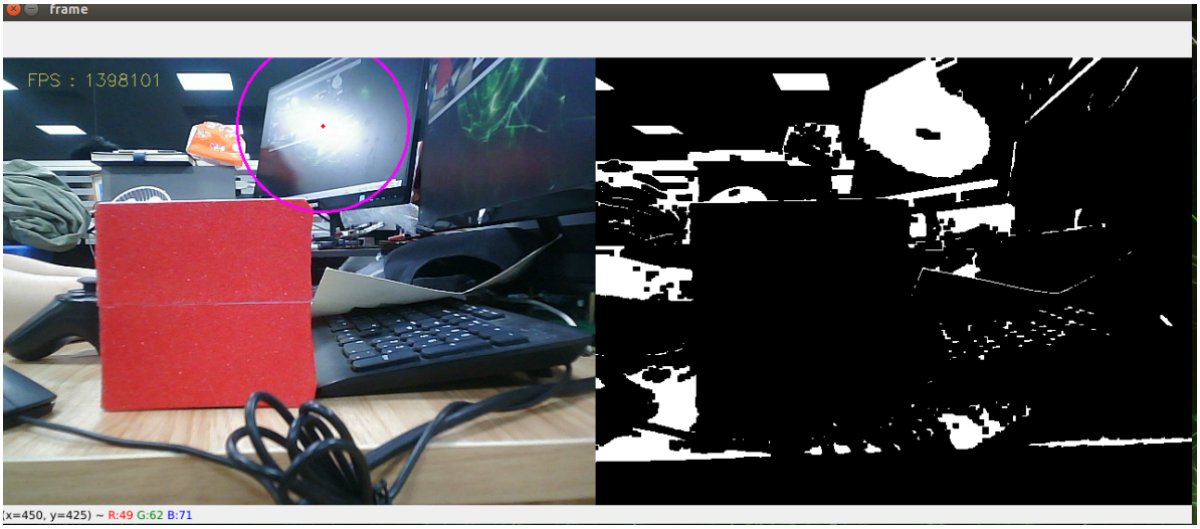
For the Raspberry Pi PI5 main control, you need to enter the docker container first, but the Orin main board does not need to enter.

According to the actual vehicle model, input in the terminal,

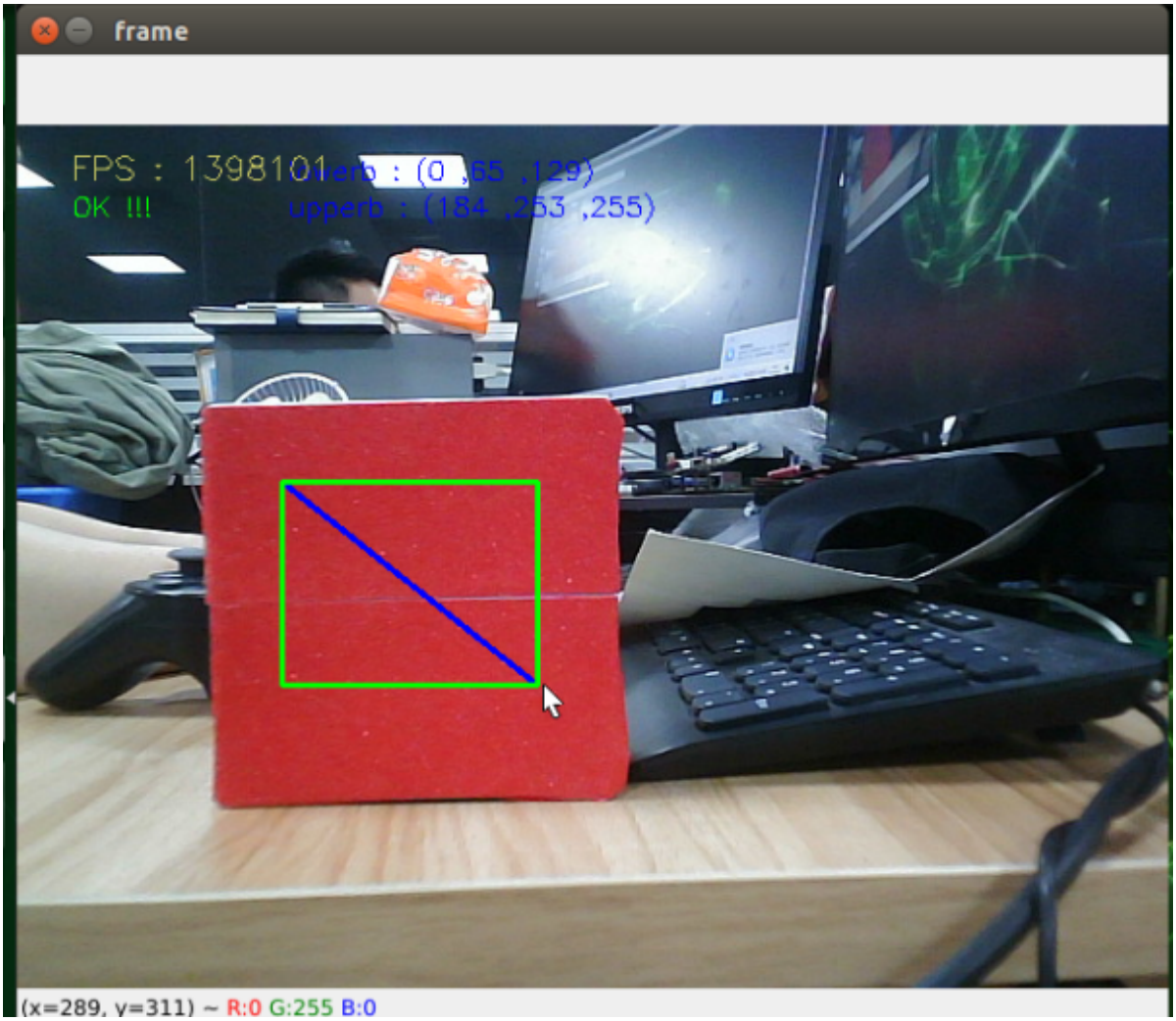
```
#The orin main control starts the depth camera first, but the pi5 main control
does not need to start
ros2 launch astra_camera astro_pro_plus.launch.xml
```

```
#Start the trolley chassis
ros2 run yahboomcar_bringup Mcnamu_driver_X3
#Launch color tracking program
ros2 run yahboomcar_astra colorHSV
ros2 run yahboomcar_astra colorTracker
#Launch depth camera The pi5 master needs to be started, and the orin master does
not need to be started.
ros2 launch astra_camera astra.launch.xml
#Start handle node
ros2 run yahboomcar_ctrl yahboom_joy_X3
ros2 run joy joy_node
```

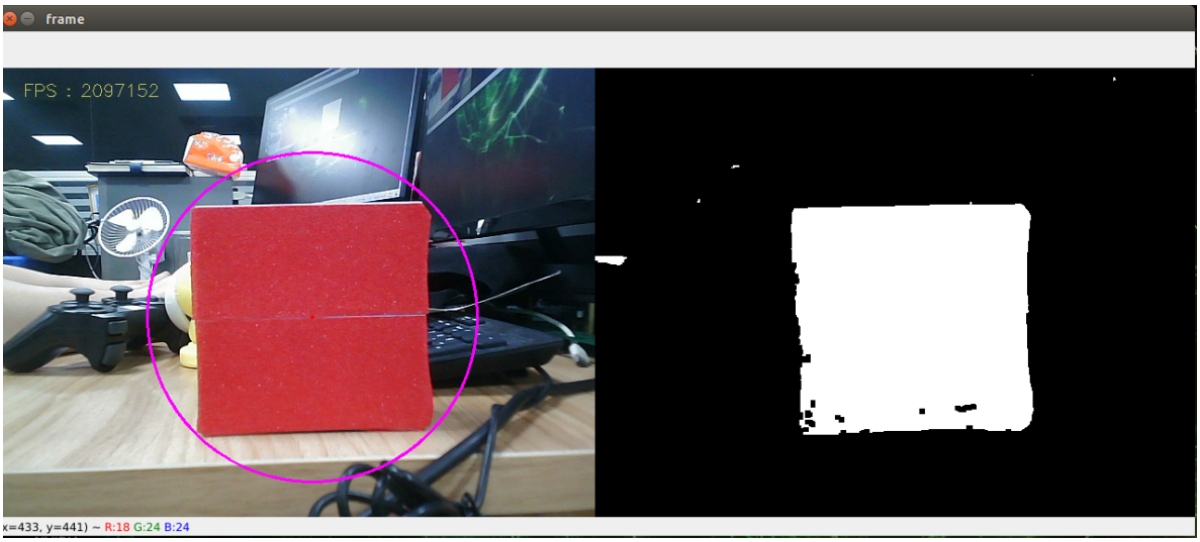
Taking tracking red as an example, after the program starts, the following screen will appear:



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



After selecting, the effect is shown in the following figure,

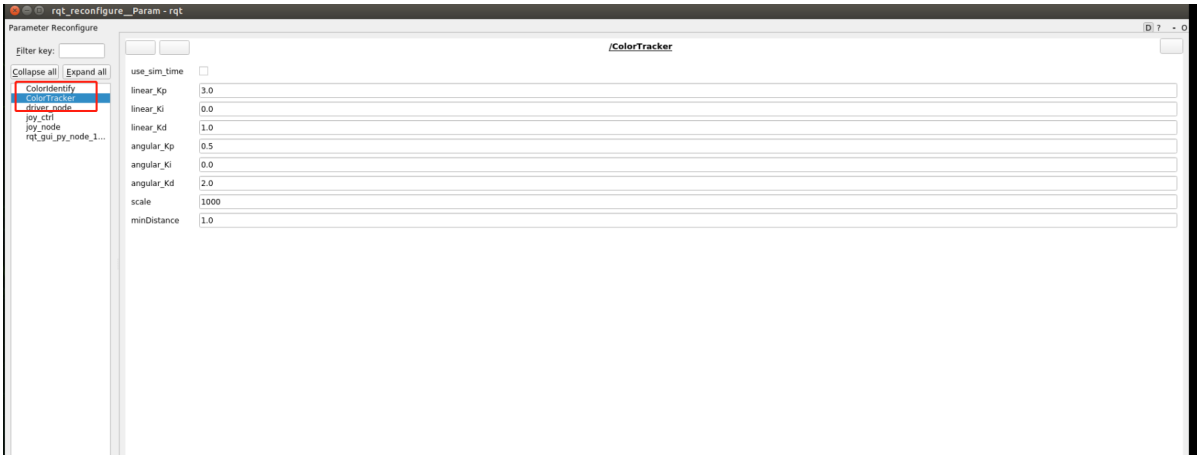


Then, press the Space bar to enter the tracking mode, slowly move the object, and the car will follow, and maintain a distance of 1 meter.

3.2、Dynamic parameter adjustment

Docker terminal input,

```
ros2 run rqt_reconfigure rqt_reconfigure
```



After modifying the parameters, click on the blank space in the GUI to write the parameter values. As can be seen from the above figure,

- ColorTracker is mainly responsible for calculating speed and adjusting speed and distance related parameters.

4、Core code

4.1、colorHSV.py

This program mainly has the following functions:

- Open the camera to obtain images;
- Obtain keyboard and mouse events for switching modes and color selection;
- Process images and publish the center coordinates of tracked objects and publish them.

Some core codes are as follows，

```
```

```
#Create a publisher and publish the center coordinates of tracked objects
self.pub_position = self.create_publisher(Position,"/Current_point", 10)
#Obtain keyboard and mouse events to obtain 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, and store the xy value in
self.circle
rgb_img, binary, self.circle = self.color.object_follow(rgb_img, self.hsv_range)
#Publish message for central 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 mainly has the following functions: receive【/current_ Point】 and depth image topic data, calculate speed size, and then publish speed data.

Some of the codes are as follows,

```
#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, obtaining self. Center_ X value and distance_
value
def positionCallback(self, msg):
def depth_img_Callback(self, msg):
#Self. Center_ X value and distance_ Values 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
```