Color tracking

Note: The ROS_DOMAIN_ID of the Raspberry Pi and the microROS control board need to be consistent. You can check [MicroROS Control Board Parameter Configuration] to set the microROS control board ROS_DOMAIN_ID. Check the tutorial [Connect MicroROS Agent] to determine whether the IDs are consistent.

1、Program function description

The MicroROS robot's monocular color tracking is capable of identifying multiple colors at any time, and can independently store the currently identified colors and control the car's pan/tilt to follow, detect, and identify the colors.

The color tracking of the MicroROS robot can also realize the function of real-time control of HSV. By adjusting the high and low thresholds of HSV, the interfering colors are filtered out, so that the squares can be ideally identified in complex environments. If the color selection effect is not ideal, If so, you need to move the car to different environments and calibrate it at this time, so that it can recognize the colors we need in complex environments.

2、Program code reference path

```
/root/yahboomcar_ws/src/yahboomcar_astra/yahboomcar_astra/colorHSV.py
/root/yahboomcar_ws/src/yahboomcar_astra/yahboomcar_astra/colorTracker.py
```

- colorHSV.py

  It mainly completes image processing and calculates the center coordinates of the tracked object.

- colorTracker.py

  Mainly based on the center coordinates and depth information of the tracked object, the steering gear motion angle data is calculated and sent to the chassis.

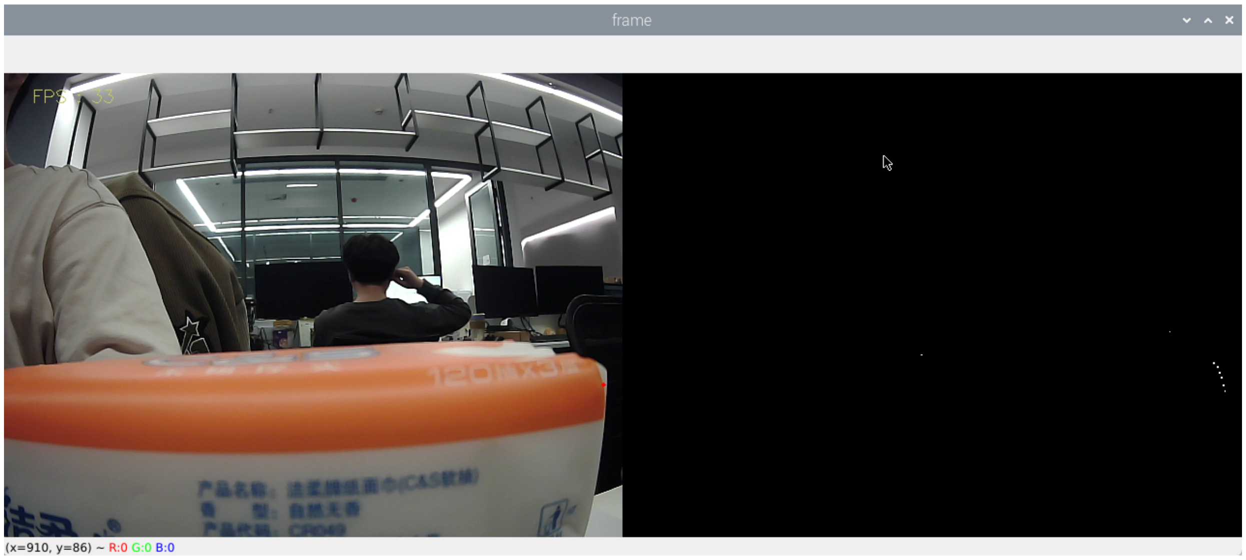
3、Program start

3.1、Start command

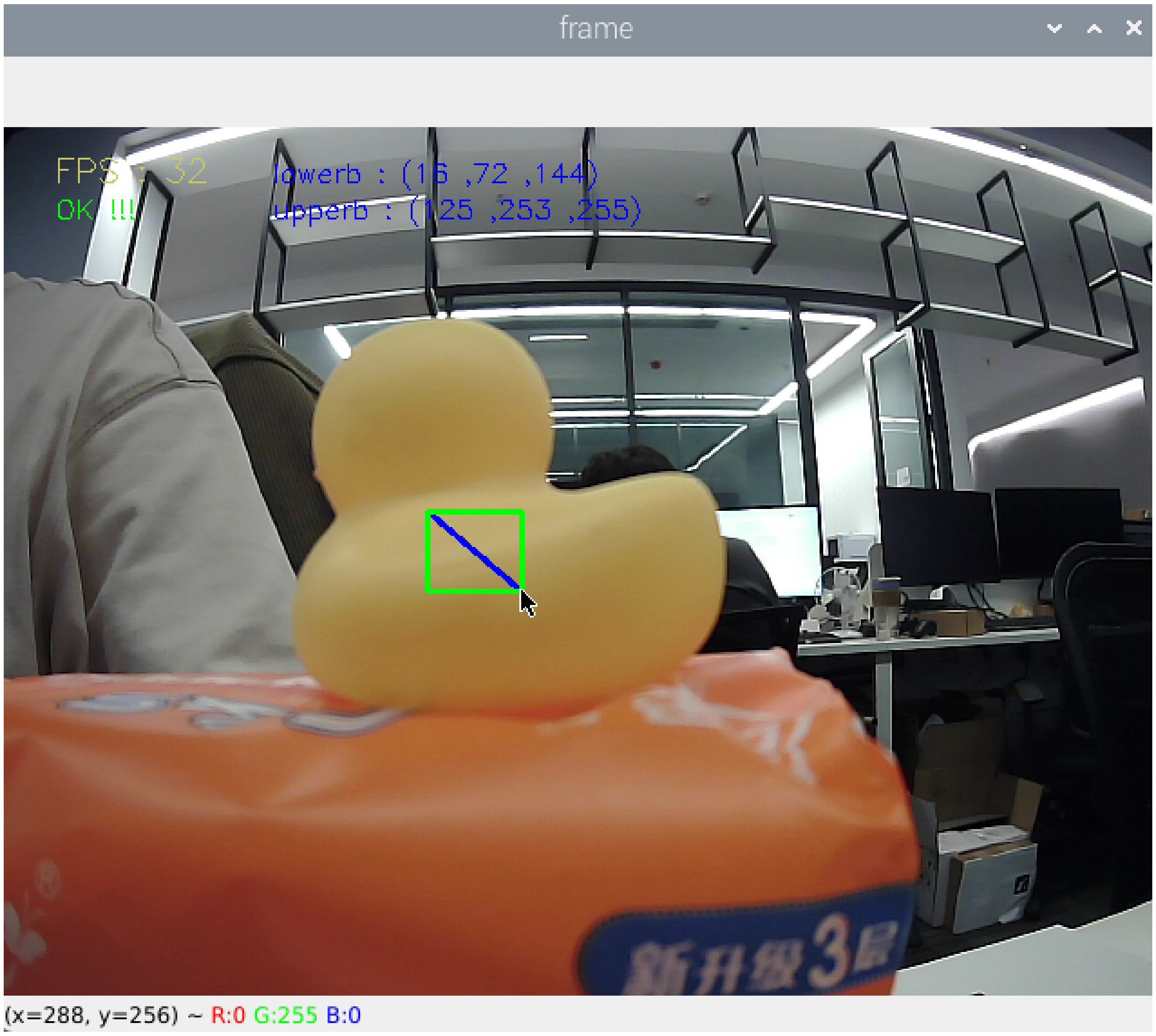
After entering the docker container, enter in the terminal,

```
#Start color tracking program
ros2 run yahboomcar_astra colorHSV
ros2 run yahboomcar_astra colorTracker
```

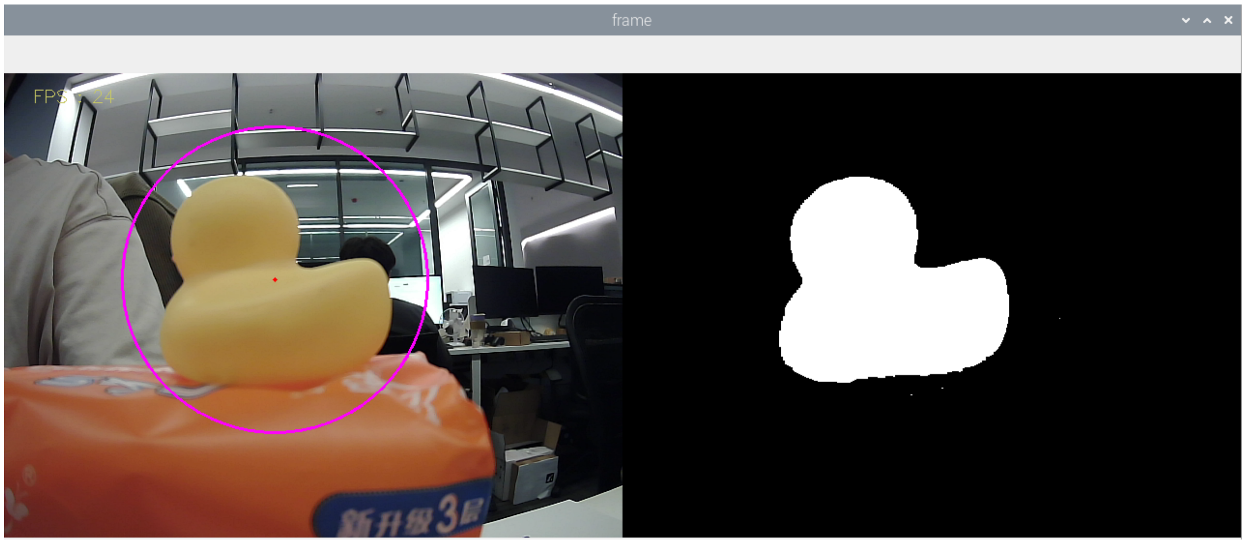
Taking tracking red as an example, after the program is started, the following screen will appear.



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



After selecting, the effect will be as shown below:

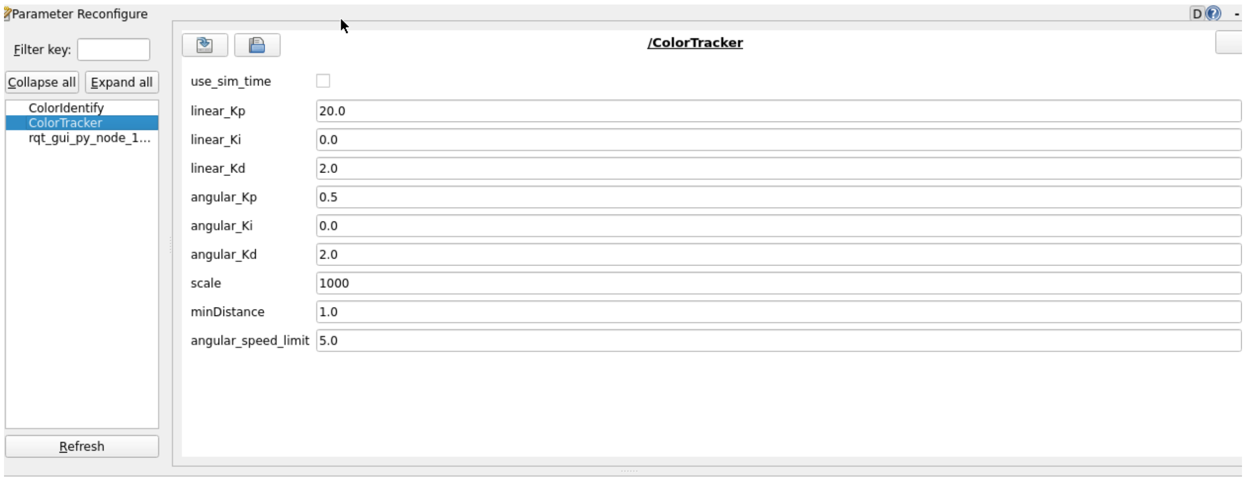


Then, press the space bar to enter tracking mode, move the object slowly, and you can see that the robot gimbal will track the movement of the color block.

3.2、Dynamic parameter adjustment

Docker terminal input,

```
ros2 run rqt_reconfigure rqt_reconfigure
```



use_sim_time	☐
linear_Kp	20.0
linear_Ki	0.0
linear_Kd	2.0
angular_Kp	0.5
angular_Ki	0.0
angular_Kd	2.0
scale	1000
minDistance	1.0
angular_speed_limit	5.0

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

- colorTracker mainly adjusts the three parameters of PID to make the gimbal more sensitive.

4、core code

4.1、colorHSV.py

This program mainly has the following functions:

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

Part of the core code is as follows:

```
#Create a publisher to publish the center coordinates of the tracking object
self.pub_position = self.create_publisher(Position,"/Current_point", 10)
#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)
```

4.2、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 data.

Part of the code is as follows,

```
#Define the topic data that subscribers need to receive
self.sub_depth = self.create_subscription(Image,"/image_raw",
self.depth_img_Callback, 1)
self.sub_JoyState
=self.create_subscription(Bool,'/JoyState',self.JoyStateCallback,1)
self.sub_position
=self.create_subscription(Position,"/Current_point",self.positionCallback,1)
#Define velocity publisher
self.pub_cmdVel = self.create_publisher(Twist,'/cmd_vel',10)
self.pub_Servo1 = self.create_publisher(Int32,"servo_s1" , 10)
self.pub_Servo2 = self.create_publisher(Int32,"servo_s2" , 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, point_y):
        [x_Pid, y_Pid] = self.linear_pid .update([point_x - 320, point_y - 240])
        if self.img_flip == True:
            self.PWMServo_X += x_Pid
            self.PWMServo_Y += y_Pid
        else:
            self.PWMServo_X  -= x_Pid
            self.PWMServo_Y  += y_Pid

        if self.PWMServo_X  >= 90:
            self.PWMServo_X  = 90
        elif self.PWMServo_X  <= -90:
            self.PWMServo_X  = -90
        if self.PWMServo_Y >= 20:
            self.PWMServo_Y = 20
        elif self.PWMServo_Y <= -90:
            self.PWMServo_Y = -90

        # rospy.loginfo("target_servox: {}, target_servoy:
{}".format(self.target_servox, self.target_servoy))
        print("servo1",self.PWMServo_X)
        servo1_angle = Int32()
        servo1_angle.data = int(self.PWMServo_X)
        servo2_angle = Int32()
        servo2_angle.data = int(self.PWMServo_Y)
        self.pub_Servo1.publish(servo1_angle)
        self.pub_Servo2.publish(servo2_angle)
```