Color Follow

1. Program Functionality

> After starting the program, the default tracking color is red. Press the `r/R` key to enter color selection mode. Select a color with the mouse, and the servo gimbal will lock onto that color. Press the `Spacebar` to enter follow mode. The gimbal will always keep the tracked object in the center of the frame, and the car will always maintain a preset distance from the tracked object. Press the `q/Q` key to exit the program. The remote controller's `R2` key has the [Pause/Start] function for this gameplay.
>
> 📝 The following object example uses a 3x3cm building block. If you change the following object, you will need to adjust the corresponding parameters to achieve your desired tracking effect!

2. Program Code Reference Path

```
~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/yahboomcar_astra/colorFollow.py
~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/yahboomcar_astra/common/astra_common.py
```

- colorFollow.py

  This mainly performs image processing. Based on the center coordinates of the tracked color object, it calculates the required servo rotation angle and the required angular velocity of the car. Based on the radius of the tracked color object, it calculates the required linear velocity of the car. After PID calculation, the data is sent to the car chassis.

  - astra_common.py
    - Color tracking (color_follow class): Identifies and tracks objects of a specific color using the HSV color space
    - Image processing tools: Includes multi-image stitching and display (ManyImgs function)
    - File read/write functions: Used to save/read HSV color ranges

3. Program startup

3.1. Startup command

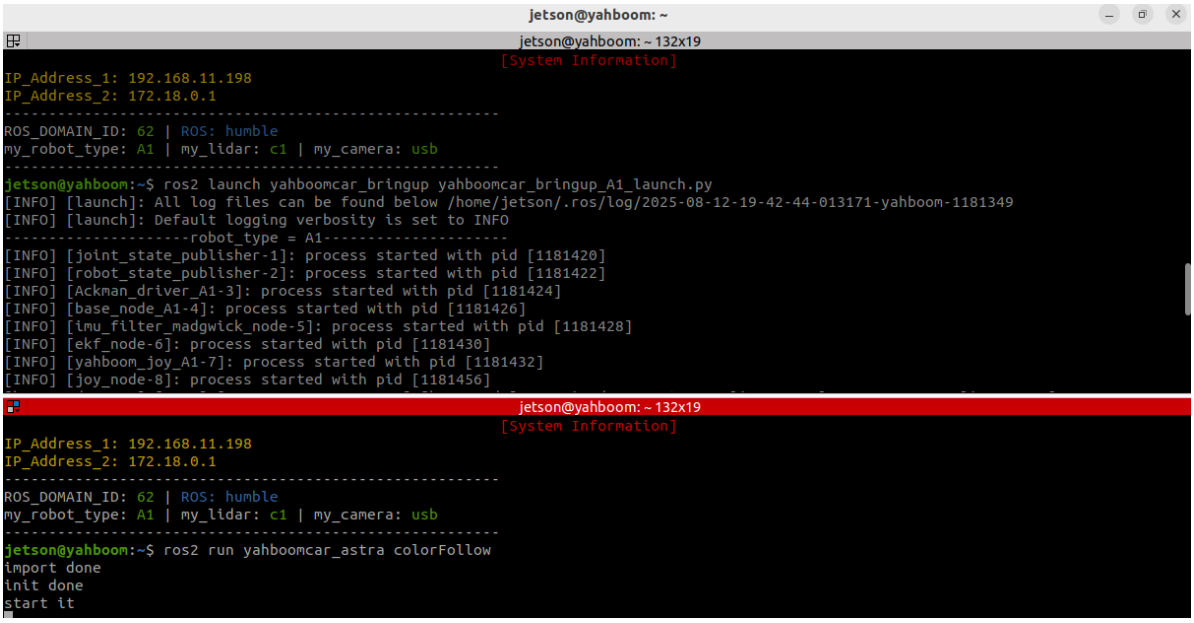
For Raspberry Pi and Jetson-Nano boards, you need to enter the Docker container first. For RDKX5 and Orin controllers, this is not necessary.

Enter the Docker container (see [Docker course] --- [4. Docker Startup Script] for steps).

All the following commands must be executed within the same Docker container **(see [Docker course] --- [3. Docker Submission and Multi-Terminal Access] for steps).**

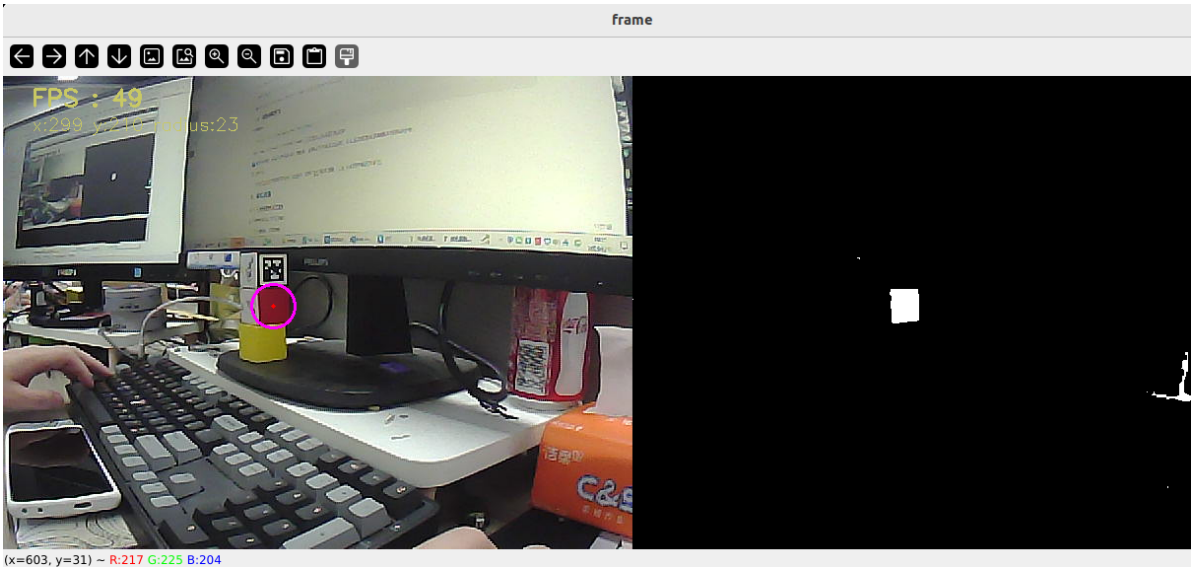
Enter the terminal.

```
# Start the car chassis and joystick nodes
ros2 launch yahboomcar_bringup yahboomcar_bringup_M1_launch.py
# Start the color following program
ros2 run yahboomcar_astra colorFollow
```

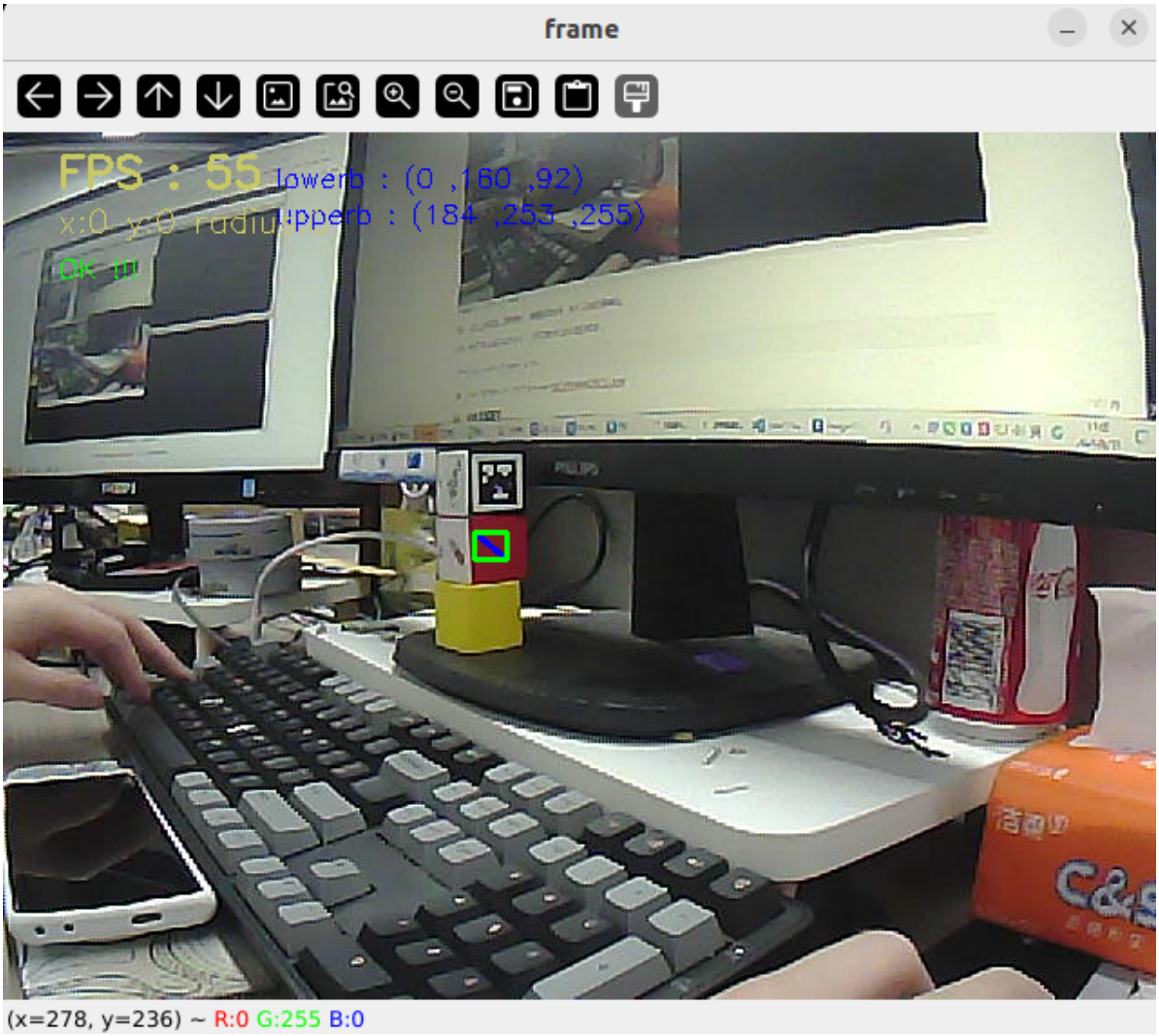


Using red tracking as an example, the following screen will appear after the program is launched.

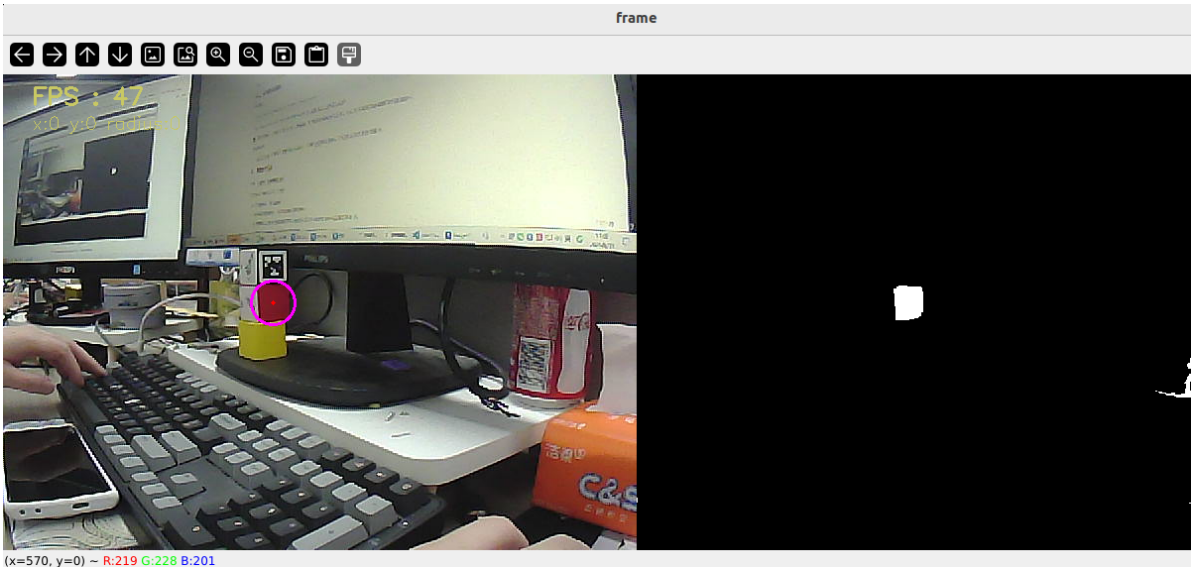
(The HSV values in the parameter file `~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/yahboomcar_astra/colorHSV.text` are loaded by default.)



Then press the `r/R` key on the keyboard to enter color selection mode. Use the mouse to select an area (this area can only have one color).



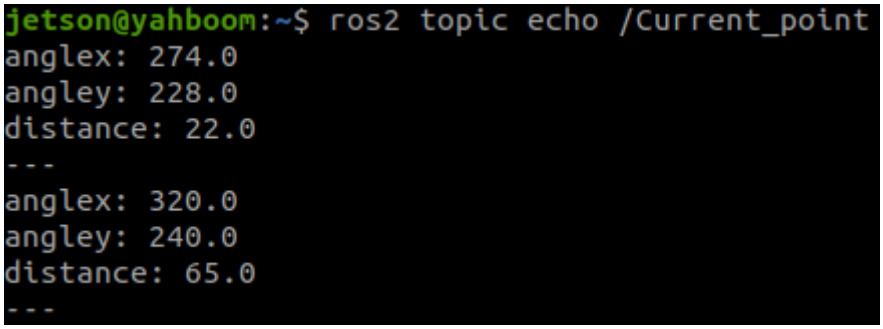
After making this selection, the effect will look like the image below.



Then, press the spacebar to enter follow mode. Slowly move the object, and the servo gimbal will follow.

Alternatively, we can enter the following command to print information about the target center coordinates and radius:

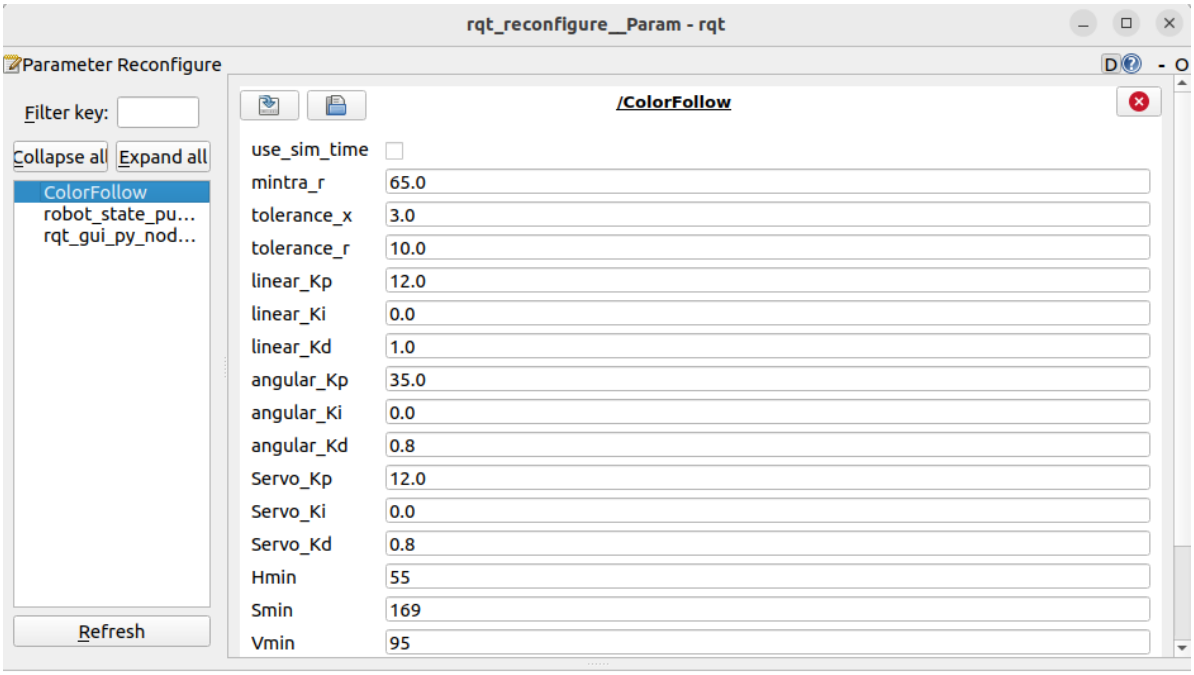
```
ros2 topic echo /Current_point
```



3.2 Dynamic Parameter Adjustment

Enter in the terminal:

```
ros2 run rqt_reconfigure rqt_reconfigure
```



☑ After modifying the parameters, click a blank area in the GUI to enter the parameter value. Note that this will only take effect for the current boot. To permanently take effect, you need to modify the parameters in the source code.

As shown in the figure above,

- colorFollow is primarily responsible for robot motion. PID-related parameters can be adjusted to optimize robot motion.

🛠 Parameter Analysis:

[mintra_r]: Critical tracking radius. When the detected object radius is less than this value, the robot advances; when it is greater than this value, the robot retreats.

[tolerance_x] and [tolerance_r]: X-axis position tolerance. When the deviation between the target center point's x-coordinate and the screen center (`320`) is less than this value, the robot is considered aligned and no further angle adjustment is performed. Radius tolerance: When the deviation between the detected target radius and the critical tracking radius (`mintra_r`) is less than this value, the robot is considered to be within the appropriate distance and no further distance adjustment is performed.

[linear_Kp], [linear_Ki], [linear_Kd]: PID control of linear velocity during robot following.

[angular_Kp], [angular_Ki], [angular_Kd]: PID control of angular velocity during robot following.

[Servo_Kp], [Servo_Ki], [Servo_Kd]: PID control of the servo gimbal speed during the following process.

[Hmin], [Smin], [Vmin]: Indicates the lowest HSV value.

[Hmax], [Smax], [Vmax]: Indicates the maximum HSV value.

4. Core Code

4.1. colorFollow.py

This program has the following main functions:

- Opens the camera and acquires images;
- Receives keyboard and mouse events for mode switching and color selection;
- Processes images and publishes the center coordinates and radius of the tracked object;
- Calculates the robot's movement speed and servo angle, and issues control commands.

The following is some of the core code:

```
# Create a publisher to publish the center coordinates and radius of the tracked
object
self.pub_position = self.create_publisher(Position, "/Current_point", 10)
# Define the servo data publisher
self.pub_Servo = self.create_publisher(ServoControl, 'Servo', 10)
# Define the vehicle velocity publisher
self.pub_cmdVel = self.create_publisher(Twist, '/cmd_vel', 10)
# Define the controller node data subscriber
self.sub_JoyState = self.create_subscription(Bool,"/JoyState",
self.JoyStateCallback, 1)
...
# Get keyboard and mouse events and get the hsv value;
if action == 32:
    self.Track_state = 'tracking'
    print("Color 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()
...
# Calculate the center coordinates and radius. self.circle stores the x, y, and
radius values.
rgb_img, binary, self.circle = self.color.object_follow(rgb_img, self.hsv_range)
...
# Publish the center coordinate message and control the car's movement.
threading.Thread(target=self.execute, args=(self.circle[0], self.circle[1],
self.circle[2])).start()
...
# Calculate the car's speed and servo angle using the PID algorithm based on the
x, y, and radius values.
def execute(self, x, y, z=None):
    position = Position()
    position.anglex = x * 1.0
    position.angley = y * 1.0
    position.distance = z * 1.0
    self.pub_position.publish(position)
    ...
    # Calculate PID control quantity
    [linear_Pid, Servox_Pid, Servoy_Pid] = self.PID_controller.update([z -
self.mintra_r, x - 320, y - 240])
    angular_Pid = self.angular_PID_controller.update([self.PWMServo_X -
self.init_servos1])[0]
    ...
    # Limit the linear velocity and angular velocity PID output range
    linear_Pid = max(-0.8, min(linear_Pid, 0.8))
```

```
    angular_Pid = max(-3.0, min(angular_Pid, 3.0))
    # Limit the servo PID polarity and maximum value
    Servox_Pid = Servox_Pid * (abs(Servox_Pid) <=self.Servo_Kp/2.4)
    Servoy_Pid = Servoy_Pid * (abs(Servoy_Pid) <=self.Servo_Kp/2.4)
    ...
    # Set the car speed and servo angle
    self.twist.linear.x = -linear_Pid
    self.twist.angular.z = -angular_Pid if self.img_flip else angular_Pid
    self.PWMServo_X += Servox_Pid if not self.img_flip else -Servox_Pid
    self.PWMServo_Y += Servoy_Pid if not self.img_flip else -Servoy_Pid
    ...
    # Issue control instructions
    self.pub_Servo.publish(self.servo_angle)
    self.pub_cmdVel.publish(self.twist)
```