Visual tracking autonomous driving

1. Program Functionality

> After starting the program, the servo gimbal will default to pointing downward, and the default tracking color is red. Press the `r/R` key to enter color selection mode. Within the line area on the screen, use the mouse to select a color. Releasing the mouse button automatically loads the processed image. Press the `Spacebar` to enter line-following mode. If the robot encounters an obstacle during operation, it will stop and a buzzer will sound. Press the `q/Q` key to exit the program. The remote controller's `R2` key has the [Pause/Start] function for this gameplay.

2. Program Code Reference Path

```
~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/yahboomcar_astra/follow_line.py
~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/yahboomcar_astra/common/follow_common.py
```

- follow_line.py

  Mainly performs trajectory recognition and motion control, calculating the car's steering speed based on the deviation from the recognized color trajectory.

- follow_common.py

  Contains core visual processing algorithms:

  - color_follow class: Implements color recognition and trajectory extraction
  - simplePID class: Implements the PID control algorithm
  - Image processing tools (ManyImgs function): Multi-image mosaic display

3. Program Startup

3.1. Startup Commands

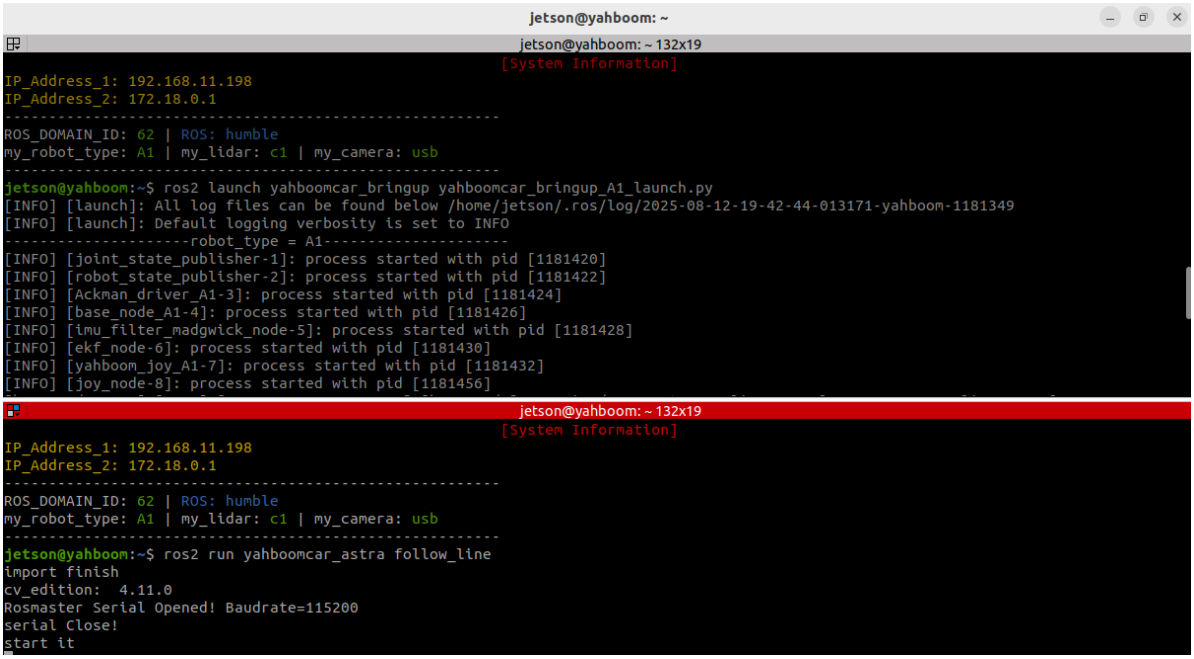
For Raspberry Pi and Jetson-Nano boards, you need to enter the Docker container first. For RDKX5 and Orin controllers, this is not necessary.

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

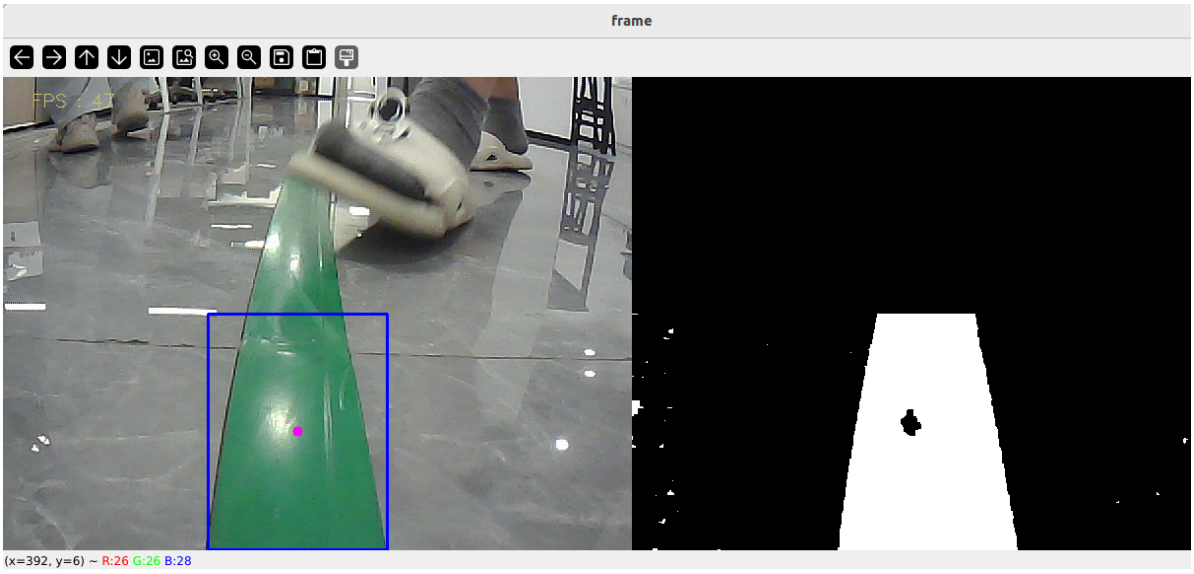
All the following commands must be executed within the same Docker container **(see [Docker course] --- [3. Docker Submission and Multi-Terminal Access] for steps).**

Enter the terminal:

```
# Start the car chassis and handle joy nodes
ros2 launch yahboomcar_bringup yahboomcar_bringup_M1_launch.py
# Start the visual tracking program
ros2 run yahboomcar_astra follow_line
```

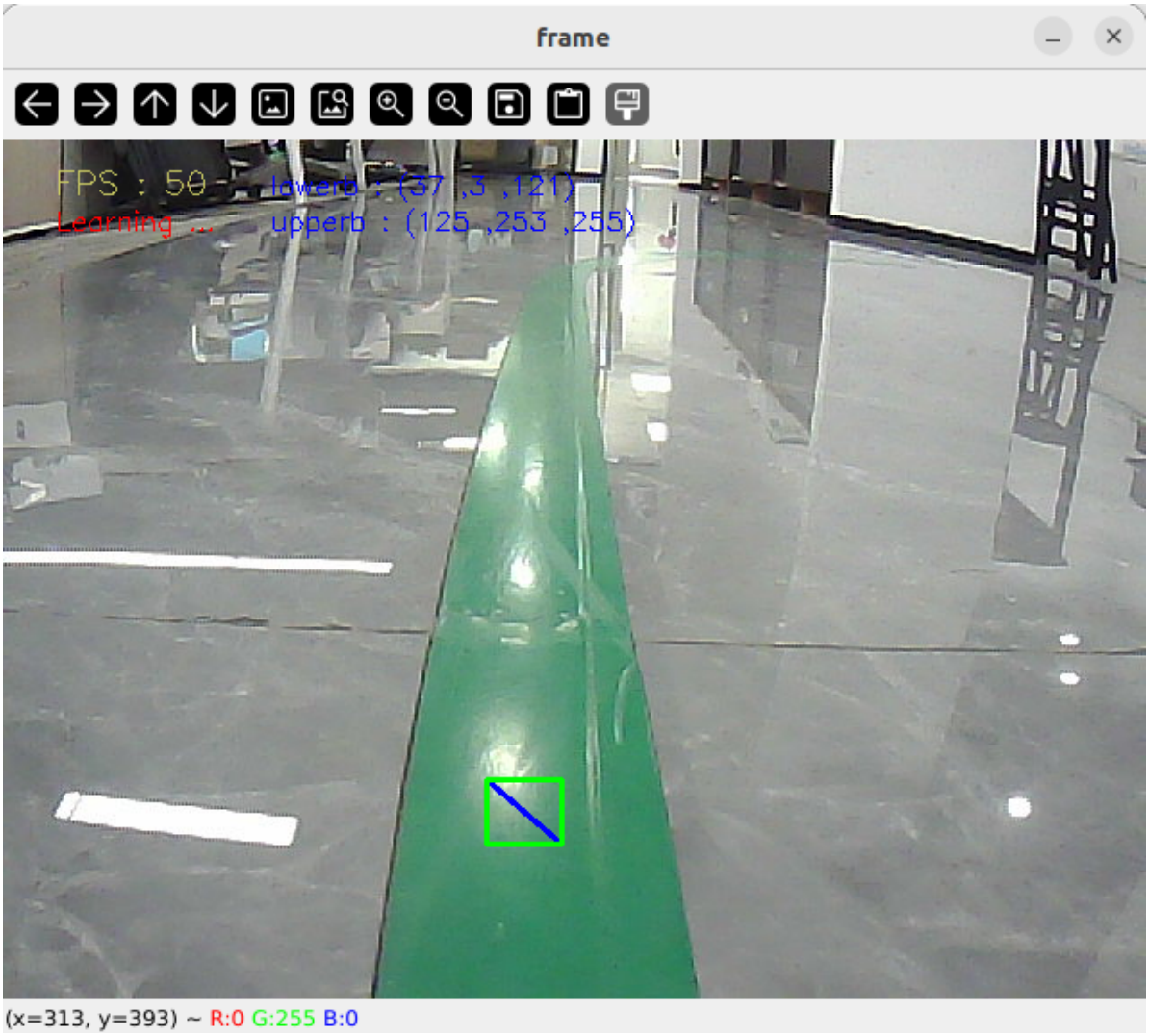


After running the program, the following screen will appear. Taking the Green Line patrol as an example,

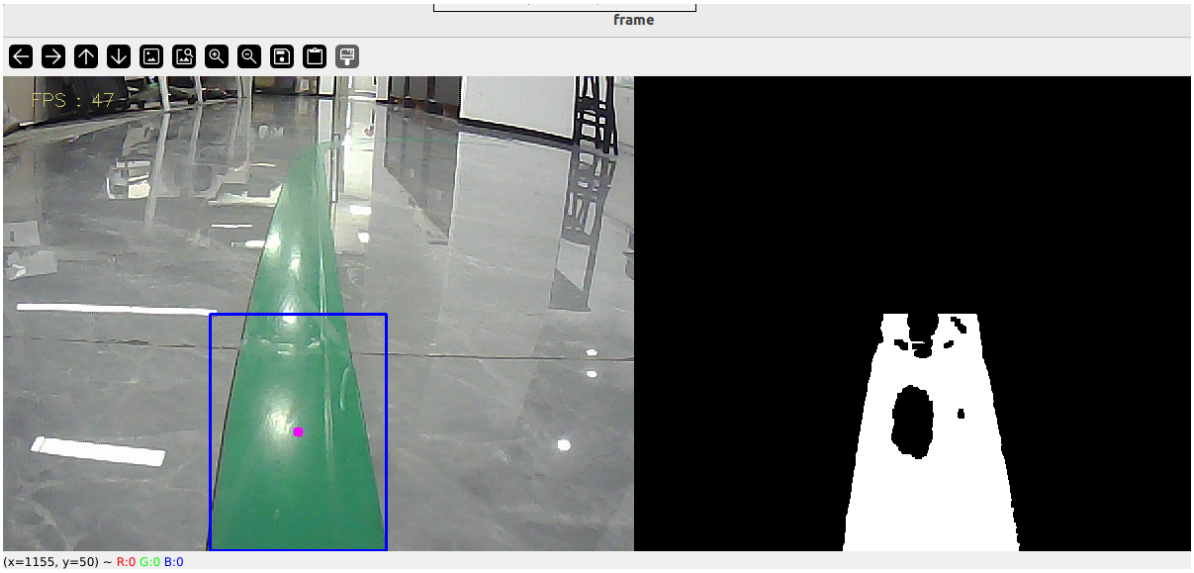


When you start the program, the previous HSV values will be used by default. You will need to select a new color. Press the `r/R` key on your keyboard to enter color selection mode. Use your mouse to select an area (this area can only have one color).

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



After making this selection, the effect will be as shown below.

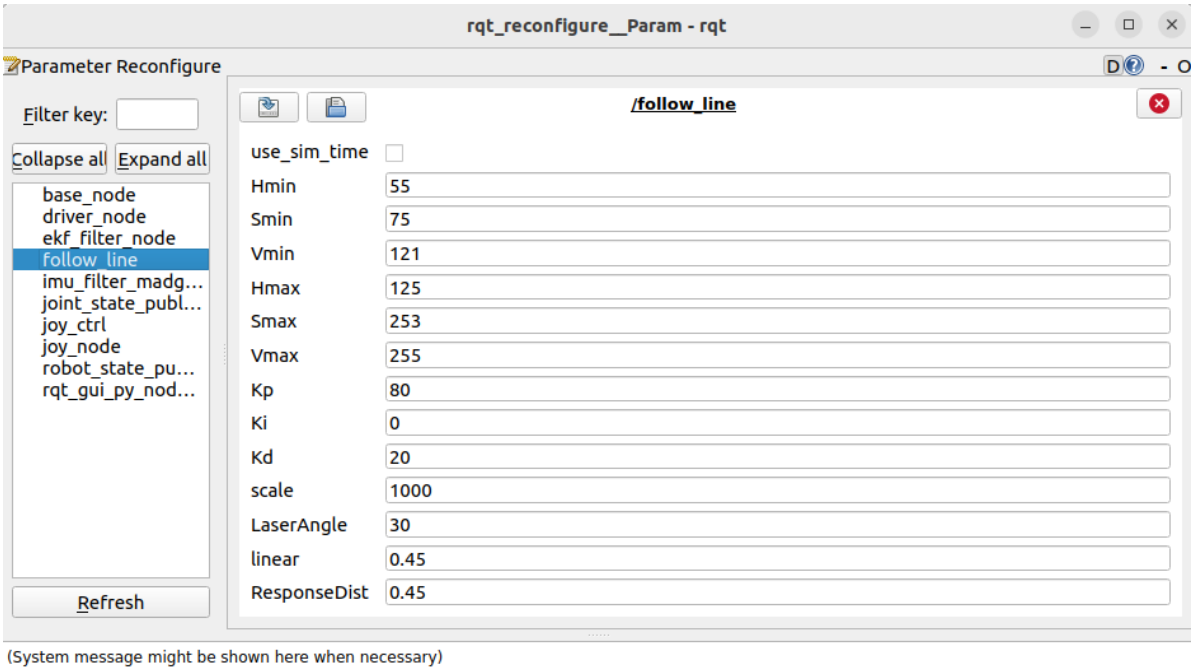


Then, press the spacebar to start calculating the speed, and the car will automatically patrol the line. If it encounters an obstacle during patrol, it will stop and the buzzer will start beeping.

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 effect the parameter, modify it in the source code.

🛠 Parameter Analysis:

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

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

[Kp], [Ki], [Kd]: Tracking PID control parameters

[scale]: PID adjustment scale factor

[LaserAngle]: LiDAR detection angle range

[linear]: Vehicle forward linear velocity

[ResponseDist]: Obstacle response distance threshold

4. Core Code

4.1. lineFollow.py

Main Functions:

- Camera image acquisition and processing
- Guideline center position detection
- PID control algorithm implementation
- LiDAR obstacle avoidance function
- Control command issuance

Some core code is as follows:

```
# Define vehicle velocity publisher
self.pub_cmdVel = self.create_publisher(Twist, '/cmd_vel', 10)
# Define the buzzer publisher
self.pub_Buzzer = self.create_publisher(Bool,"/Buzzer",1)
# Define the servo data publisher
self.pub_Servo = self.create_publisher(ServoControl, 'Servo', 10)
# Define the controller node data subscriber
self.sub_JoyState =
self.create_subscription(Bool,"/JoyState",self.JoyStateCallback,1)
# Define the radar data subscriber
self.sub_scan =
self.create_subscription(LaserScan,"/scan",self.registerScan,self.qos_profile)
...
# Get keyboard and mouse events and get the hsv value;
if action == 32: self.Track_state = 'tracking'
elif action == ord('i') or action == 105: self.Track_state = "identify"
elif action == ord('r') or action == 114: self.Reset()
elif action == ord('q') or action == 113: self.cancel()
...
# Calculate the center coordinates and radius. self.circle stores the x, y, and z
values.
rgb_img, binary, self.circle = self.color.line_follow(rgb_img, self.hsv_range)
...
# Calculate the angular velocity of the robot based on the x value using the PID
algorithm.
def execute(self, center_x):
    ...
    # PID control calculation steering
    [z_Pid, _] = self.PID_controller.update([(point_x - 320)*1.0/16, 0])
    ...
    # Setting the speed
    twist.linear.x = self.linear
    ...
    # Obstacle Detection
    if self.warning > 10:
        print("Obstacles ahead !!!")
        self.pub_cmdVel.publish(Twist())
        self.Buzzer_state = True
        b.data = True
        self.pub_Buzzer.publish(b)
    else:
        if self.Buzzer_state == True:
            b.data = False
            for i in range(3): self.pub_Buzzer.publish(b)
            self.Buzzer_state = False
```

4.2、follow_common.py

Trajectory recognition core function:

```
def line_follow(self, rgb_img, hsv_msg):
    # Convert to HSV color space
    hsv_img = cv.cvtColor(rgb_img, cv.COLOR_BGR2HSV)
    # Create a mask based on the HSV range
    lower = np.array(hsv_msg[0], dtype="uint8")
    upper = np.array(hsv_msg[1], dtype="uint8")
    mask = cv.inRange(hsv_img, lower, upper)
```

```
    # Morphological processing
    kernel = cv.getStructuringElement(cv.MORPH_RECT, (5, 5))
    mask = cv.morphologyEx(mask, cv.MORPH_CLOSE, kernel)
    # Extract contours
    contours, _ = cv.findContours(mask, cv.RETR_EXTERNAL,
cv.CHAIN_APPROX_SIMPLE)
    if len(contours) > 0:
        # Find the largest contour
        max_contour = max(contours, key=cv.contourArea)
        # Calculate contour moments
        M = cv.moments(max_contour)
        if M["m00"] != 0:
            center_x = int(M["m10"] / M["m00"])
            center_y = int(M["m01"] / M["m00"])
            # Draw trajectory lines and center points
            cv.drawContours(rgb_img, [max_contour], -1, (0, 255, 0), 2)
            cv.circle(rgb_img, (center_x, center_y), 5, (0, 0, 255), -1)
            return rgb_img, mask, (center_x, center_y)
    return rgb_img, mask, (0, 0)
```