Line patrol autopilot

Note: The virtual machine, ROS-wifi image transmission module and ESP32 communication board ROS_DOMAIN_ID need to be consistent, and both must be set to 20. You can view [ESP32 communication board parameter configuration] to set the ESP32 communication board ROS_DOMAIN_ID, and view the tutorial [Connecting MicroROS Agent] to determine whether the ID is consistent.

Before running the experiment, please make sure that the microros balance car and ROS-wifi image transmission module have correctly enabled the agent on the virtual machine (linux with humbleROS2 system)

1. Program function description

After the program is started, adjust the pitch angle of the camera, bend the camera down so that the camera can see the line, then click the image window and press the r key to enter the color selection mode; then in the area of the line in the picture, frame the color of the line you need to patrol, and the processed image will be automatically loaded after releasing the mouse; finally, press the space bar to enable the line patrol function. During the operation of the car, if it encounters an obstacle, it will stop and the buzzer will sound.

2. Program code reference path

The source code of this function is located at,

```
/home/yahboom/yahboomcar_ws/src/yahboom_esp32ai_car/yahboom_esp32ai_car/follow_li
ne.py
```

3. Program startup

3.1. Start command

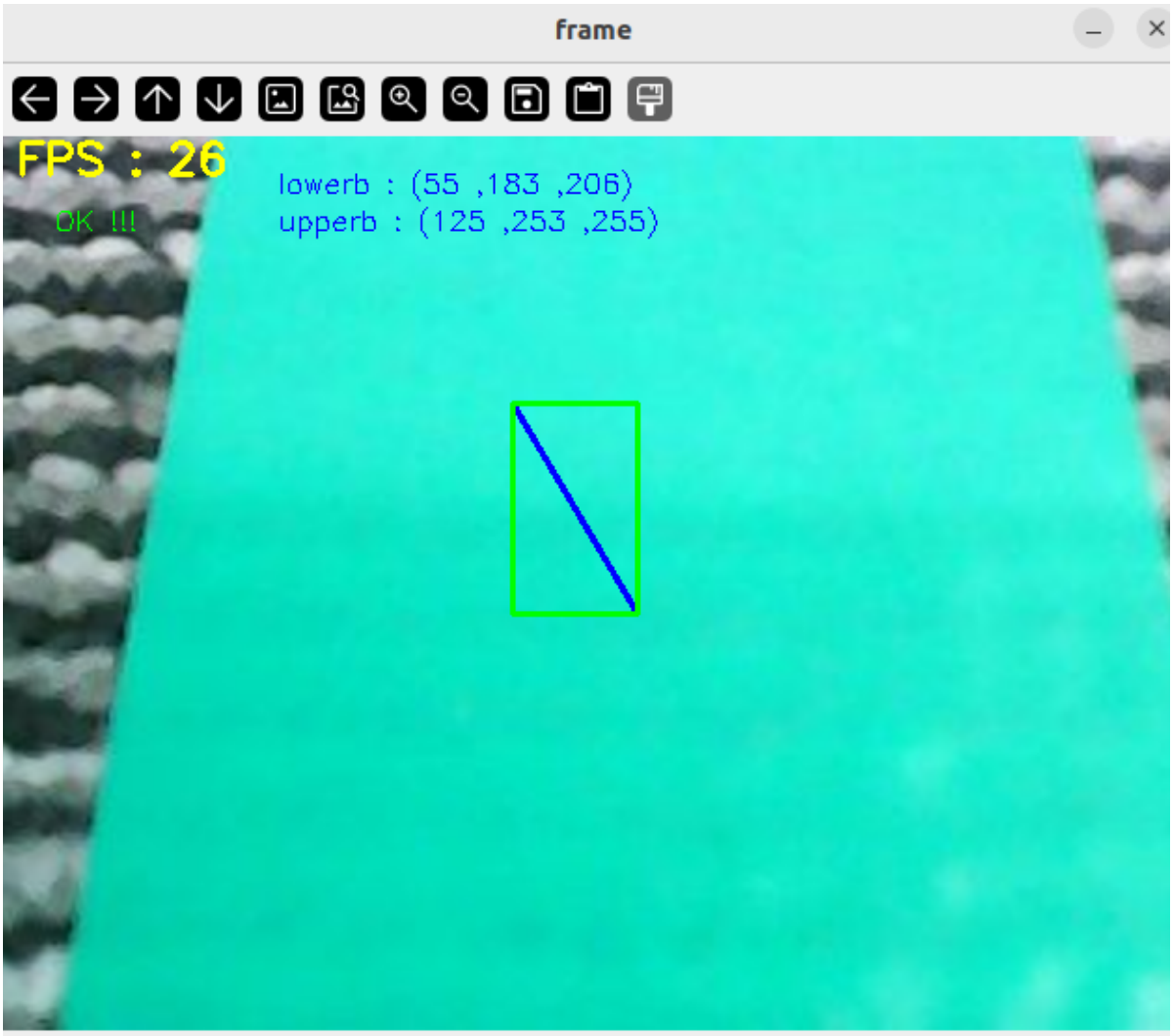
Input in the terminal,

```
#Start chassis driver
ros2 launch yahboomcar_bringup yahboomcar_bringup_launch.py
```

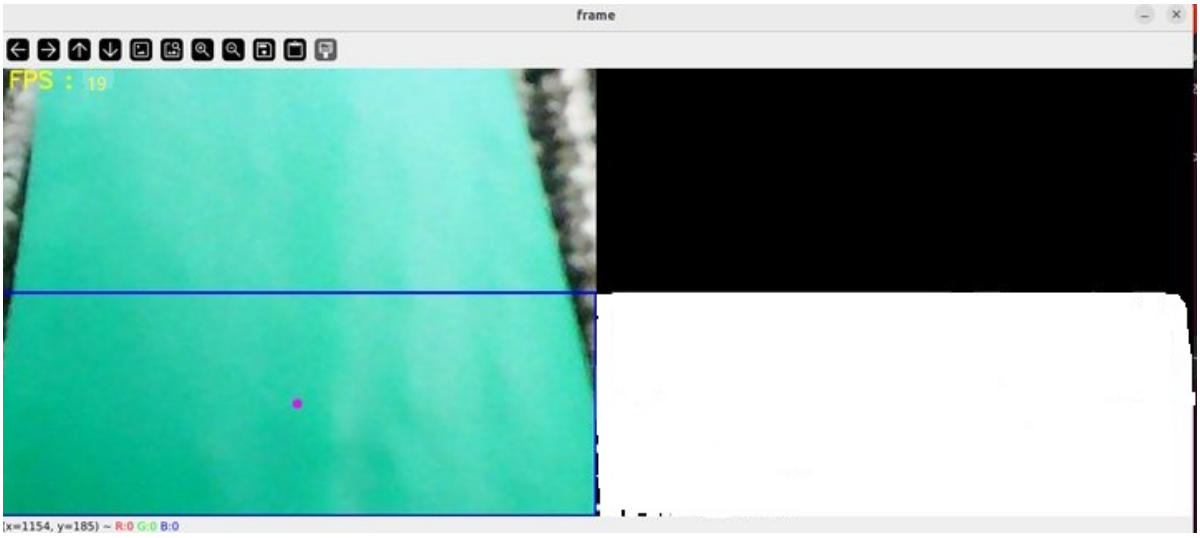
```
#Start the automatic line patrol program
ros2 run yahboom_esp32ai_car follow_line
```

If the camera image is inverted, you need to see **3. Camera image correction (select)** document to correct it yourself, and this experiment will not be described.

Take the green line patrol as an example,



After pressing the r key, select the blue line area as shown above, release the mouse after selection,

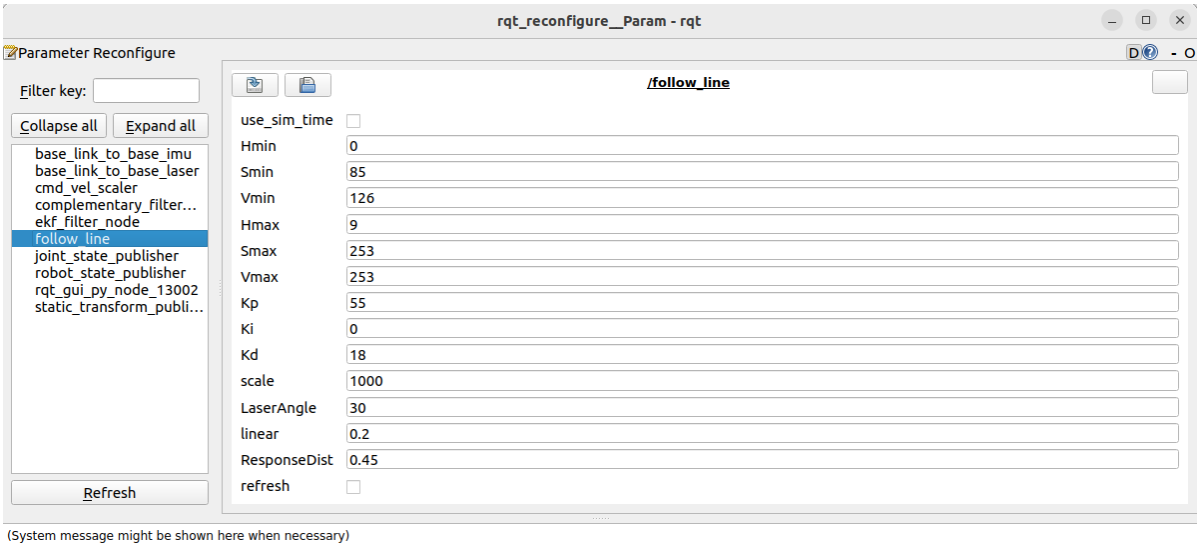


As shown in the above figure, the processed image is displayed on the right, and it will display the green line part. **Because the camera has only 200W pixels, put the line in the middle of the image as much as possible**, and then press the space bar to start calculating the speed, and the car will patrol the line automatically.

3.2, Dynamic parameter adjustment

You can adjust the relevant parameters through the dynamic parameterizer, or directly modify the source code, and then recompile and update. Terminal input,

```
ros2 run rqt_reconfigure rqt_reconfigure
```



The adjustable parameters are,

Parameter	Description
Kp	PID P value
Ki	PID I value
Kd	PID D value
scale	PID adjustment scale factor
LaserAngle	Radar detection angle
linear	Linear speed
ResponseDist	Obstacle avoidance detection distance
refresh	Refresh parameter button

4. Core code

Let's first sort out the implementation principle of the patrol line, by

- Calculate the offset between the center coordinates of the patrol line and the center of the image,
- Calculate the angular velocity value according to the coordinate offset,
- Release the speed to drive the car.

Calculate the center coordinates,

```
#Calculate hsv value
rgb_img, self.hsv_range = self.color.Roi_hsv(rgb_img, self.Roi_init)
#Calculate self.circle, calculate the X coordinate and radius value. If the
radius value is 0, it means that no line is detected, and the parking information
is released.
rgb_img, binary, self.circle = self.color.line_follow(rgb_img, self.hsv_range)
```

Calculate the value of the angular velocity,

```
#320 is the value of the X coordinate of the center point. By the deviation
between the X value of the obtained image and 320, we can calculate "how far am I
from the center now", and then calculate the value of the angular velocity
[z_Pid, _] = self.PID_controller.update([(point_x - 320)*1.0/16, 0])
```