Visual tracking

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 want to patrol, and release the mouse to automatically load the processed image; finally, press the space bar to turn on the patrol function. When the car encounters an obstacle during operation, it will stop and the buzzer will sound.

2. Program code reference path

After entering the docker container, the source code of this function is located at,

```
/root/yahboomcar_ws/src/yahboomcar_astra/yahboomcar_astra/
```

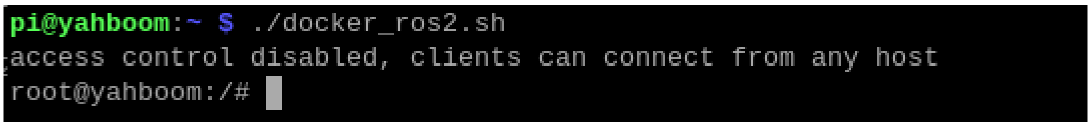
3. Program startup

3.1. Startup command

Open a terminal and enter the following command to enter docker,

```
./docker_ros2.sh
```

The following interface appears, which means that you have successfully entered docker



```
pi@yahboom:~ $ ./docker_ros2.sh
access control disabled, clients can connect from any host
root@yahboom:/#
```

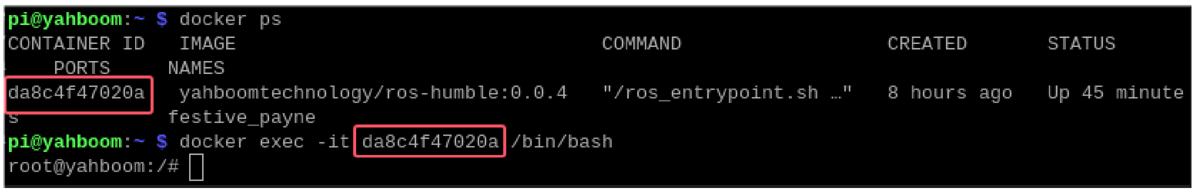
Start chassis

```
ros2 launch yahboomcar_bringup bringup.launch.py
```

Open a new terminal and enter the same docker. Change the following da8c4f47020a to the ID displayed in the actual terminal

```
docker ps
```

```
docker exec -it da8c4f47020a /bin/bash
```

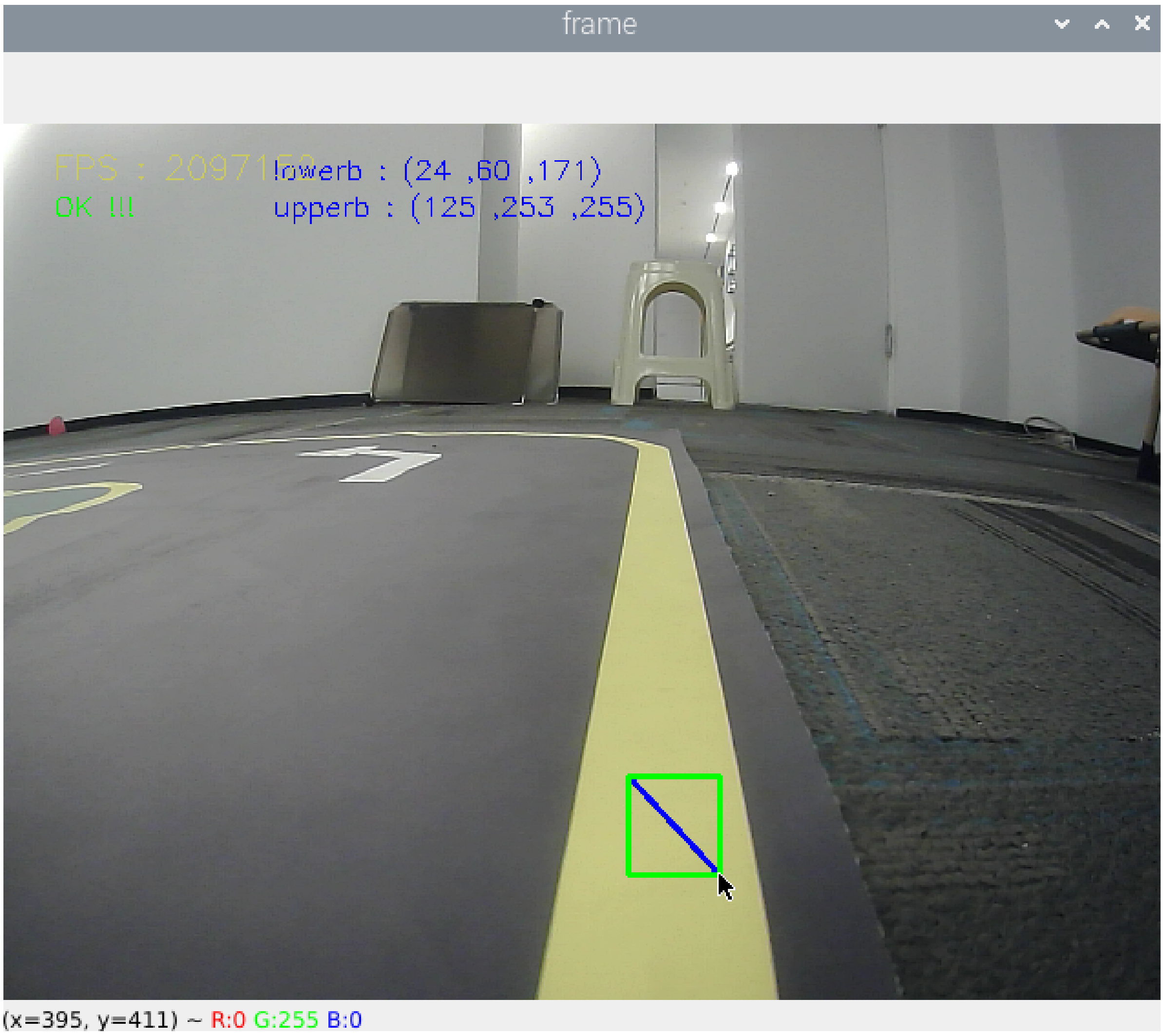


```
pi@yahboom:~ $ docker ps
CONTAINER ID   IMAGE                                 COMMAND                   CREATED         STATUS
    PORTS     NAMES
da8c4f47020a   yahboomtechnology/ros-humble:0.0.4   "/ros_entrypoint.sh …"   8 hours ago     Up 45 minute
s             festive_payne
pi@yahboom:~ $ docker exec -it da8c4f47020a /bin/bash
root@yahboom:/#
```

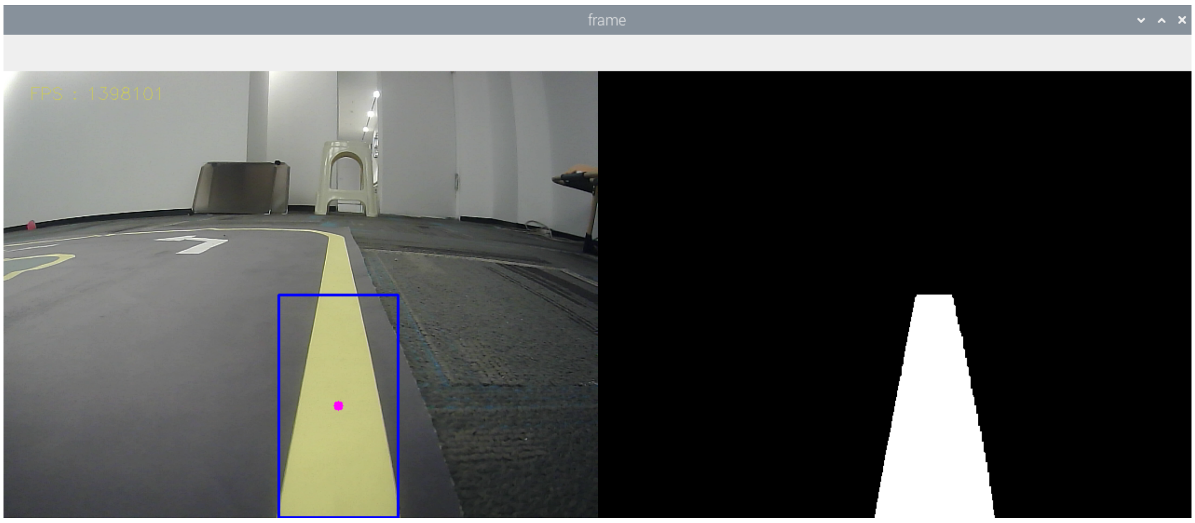
After entering the docker container, enter the terminal,

```
ros2 run yahboomcar_astra  follow_line
```

Take the Yellow Line patrol as an example.



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

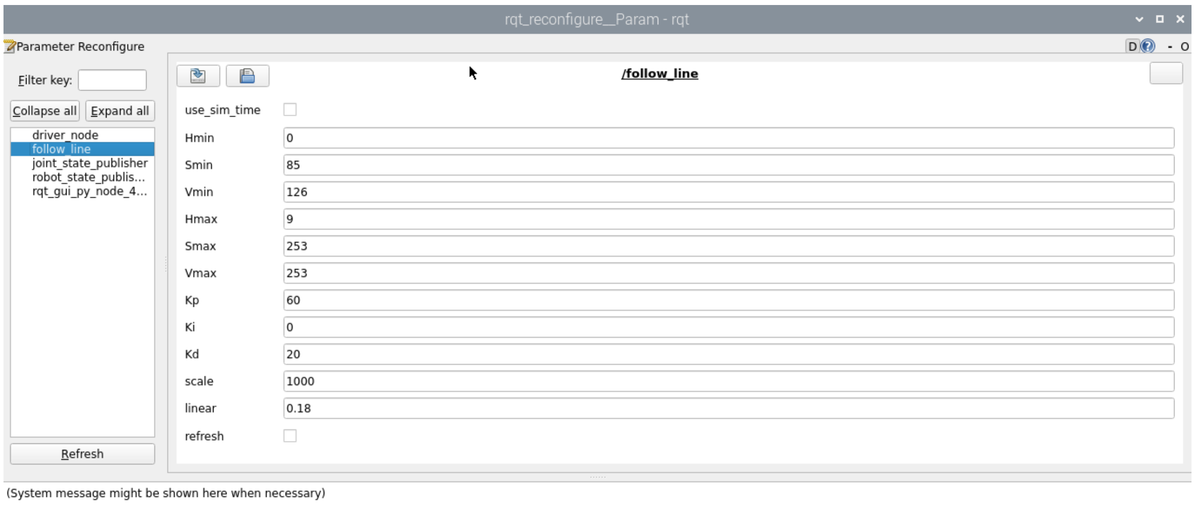


As shown in the above picture, the processed image is shown on the right, and the yellow line part will be displayed. Then press the space bar to start calculating the speed, and the car will patrol the line and drive automatically.

3.2, Dynamic parameter adjustment

The relevant parameters can be adjusted through the dynamic parameterizer, input in the docker terminal,

```
ros2 run rqt_reconfigure rqt_reconfigure
```



```
use_sim_time  [ ]
Hmin          0
Smin          85
Vmin          126
Hmax          9
Smax          253
Vmax          253
Kp            60
Ki            0
Kd            20
scale         1000
linear        0.18
refresh       [ ]
```

(System message might be shown here when necessary)

The adjustable parameters are,

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

4. Core code

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

- Calculate the offset between the center coordinates of the patrol line and the center of the image,
- Calculate the value of the angular velocity based on 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, then the parking
information will be released
rgb_img, binary, self.circle = self.color.line_follow(rgb_img, self.hsv_range)
```

Calculate the value of angular velocity,

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