巡线功能使用与讲解

1. 功能简介

巡线功能主要是通过 WHEELTEC 机器人的摄像头硬件，使 ROS 机器人根据特定颜色的轨迹进行运动，添加了通过雷达检测从而避障的节点，巡线运动过程中若遇到障碍时会停止运动，若视野内无选定颜色轨迹时，小车也会停止运动。

2. 使用方法

使用前需要确保上位机连接小车 WiFi，并已经 SSH 远程登录到小车端。

1) 打开终端输入启动巡线功能的 launch 文件指令

```
roslaunch simple_follower line_follower.launch
```

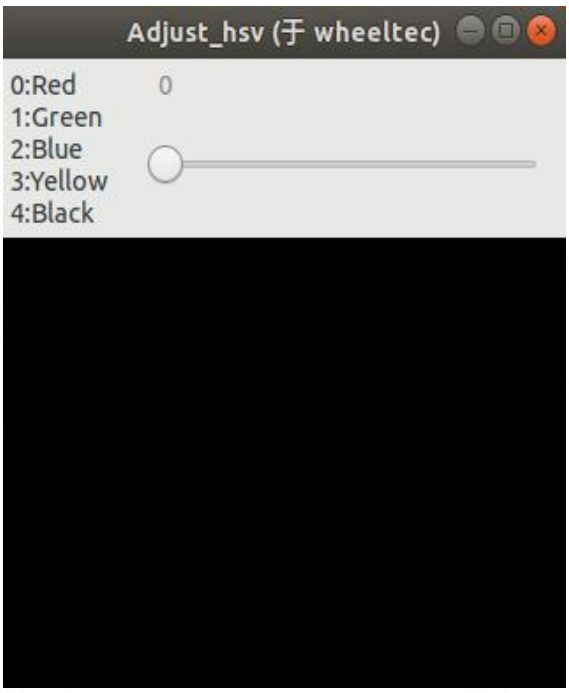
巡线功能节点开启后，终端会出现黄色警告类型的提示，这几个黄色提示为正常现象，也标志着巡线节点和雷达避障节点的开启。当巡线功能启动后，会有此弹窗出现：

```
[ INFO] [1713869653.418409611]: pubscanthread
[ INFO] [1713869653.497345482]: output frame: odom_combined
[ INFO] [1713869653.529909741]: base frame: base_footprint
[ INFO] [1713869653.640009222]: Lidar is N10_P
[ INFO] [1713869653.643771000]: Opening PCAP file
port = /dev/wheeltec_lidar, baud_rate = 460800
open_port /dev/wheeltec_lidar OK !
[ INFO] [1713869653.682231759]: Initialised lslidar without error
[ INFO] [1713869653.766937963]: Initializing Odom sensor
[ INFO] [1713869653.827146648]: Odom sensor activated
[ INFO] [1713869653.850916500]: Kalman filter initialized with odom measurement
[ INFO] [1713869653.867968815]: Initializing Imu sensor
[ INFO] [1713869653.967057685]: Imu sensor activated
[ INFO] [1713869655.251911370]: Device "2bc5/0402@1/6" found.
Warning: USB events thread - failed to set priority. This might cause loss of data...
[ INFO] [1713869655.394171388]: device name: Orbbec Astra S
[WARN] [1713869655.398684]: 1
[WARN] [1713869655.513818]: laser no object found
[ INFO] [1713869656.452601943]: Starting color stream.
[ INFO] [1713869656.947524943]: using default calibration URL
[ INFO] [1713869656.947851776]: camera calibration URL: file:///home/wheeltec/.ros/camera_info
/rgb_Astra_Orbbec.yaml
```

终端提示信息

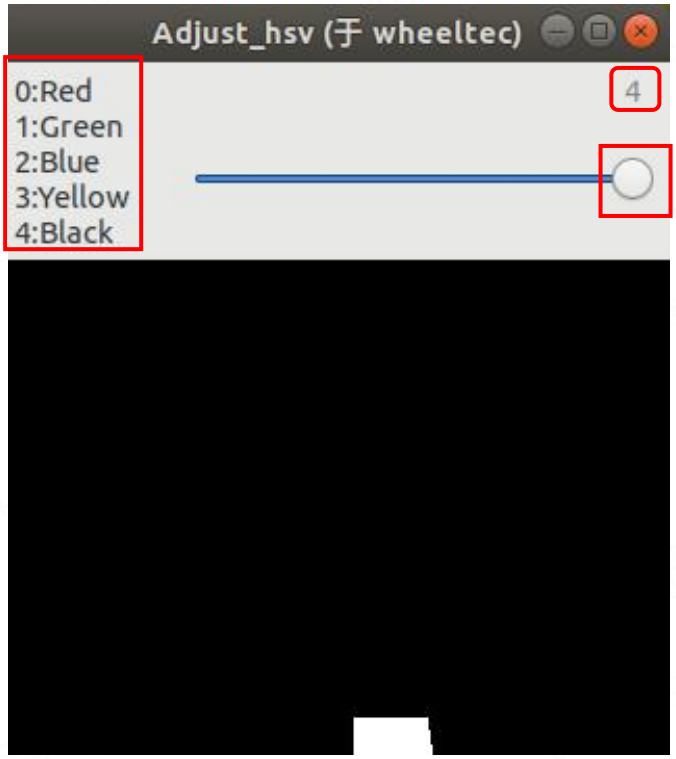
2) 通过弹窗选择跟随的轨迹线的颜色

当巡线功能启动后，会有此弹窗出现：



弹窗

此弹窗的作用是选择小车要跟随运动的轨迹线的颜色，这里我们以黑色为例，将滑块通过鼠标点击（或是鼠标按住按钮进行拖动）的方式放置在“4”处，根据弹窗提示“4”对应颜色为黑色。



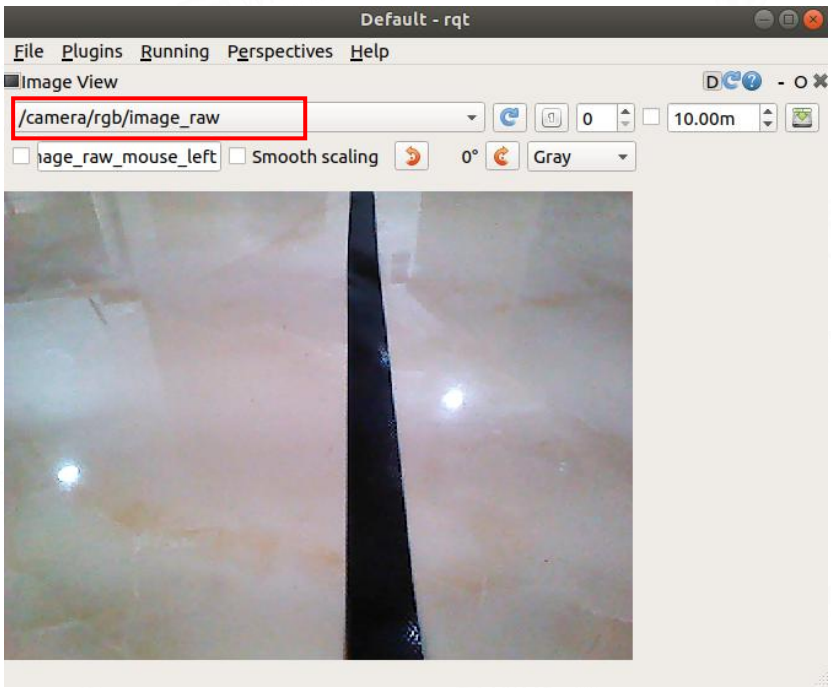
选择颜色

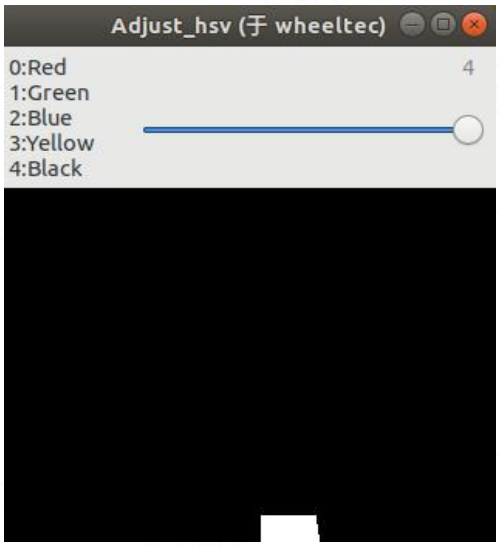
3) 通此时可以在新的终端通过运行 rqt 工具来查看摄像头所看到的画面

```
passoni@passoni:~$ rqt_image_view
```

rqt_image_view

使用 rqt_image_view 查看时，我们可以在左上角选择对应的 RGB 话题，此时可以看到小车面前是有一条黑色的线，而弹窗中白色部分的出现也意味着摄像头信息处理后已经提取到了黑色信息，小车将会沿着黑线进行运动。白色部分只有一小段是因为在我们的源码中仅截取了视野范围内最靠近小车的一小段进行处理。

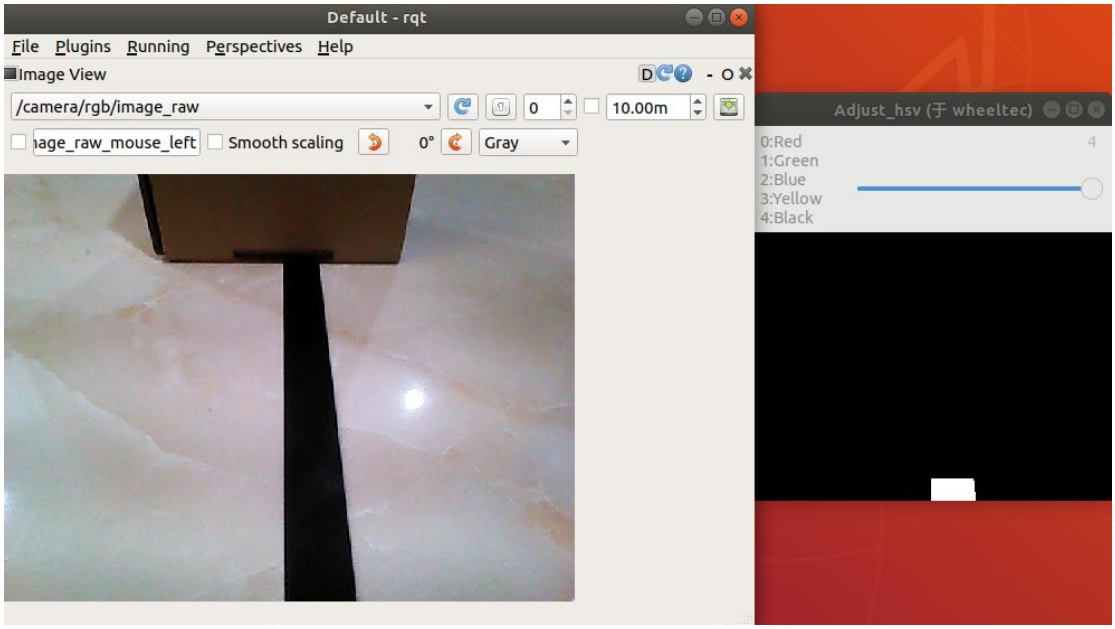




小车摄像头所看到的画面与处理后（弹窗）的画面

4) 检测障碍物

在巡线过程中，如果前方黑线上有障碍物，当雷达检测到障碍物距离较近时，会发布置为 0 的速度从而使小车停止运动。



前方有障碍物的情形

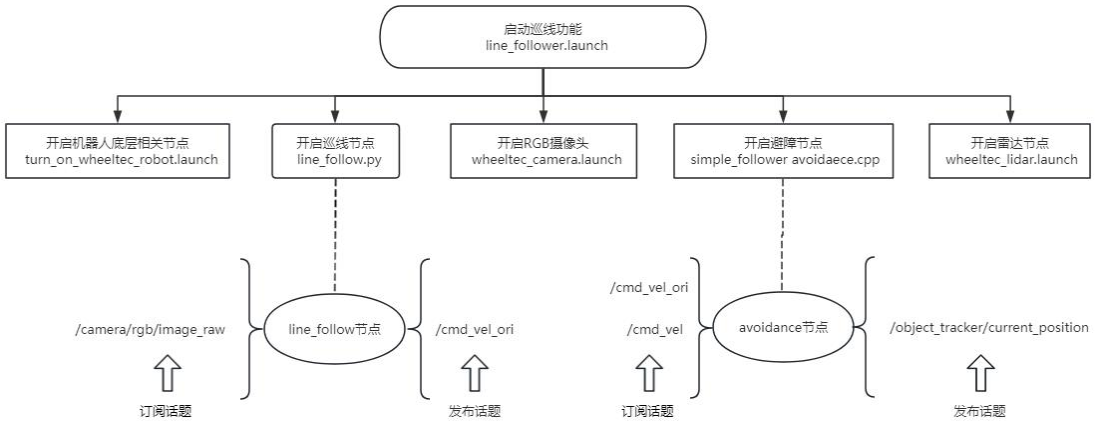
3. 巡线功能程序讲解

巡线功能是通过 line_follower.launch 文件开启的，launch 文件的内容比较简单，主要是运行各个节点，通过各节点共同作用实现功能，下面我们主要讲解一下巡线节点以及避障节点。

```
<launch>
  <!-- 开启 RGB 摄像头  -->
  <include file="$(find turn_on_wheeltec_robot)/launch/wheeltec_camera.launch"
/>
  <!-- 开启巡线节点  -->
  <node name='line_tracker' pkg="simple_follower" type="line_follow.py">
  </node>
  <node pkg="simple_follower" type="avoidance" name="avoidance" />
  <!-- 开启机器人底层相关节点  -->
  <include file="$(find
turn_on_wheeltec_robot)/launch/turn_on_wheeltec_robot.launch" />
  <!-- turn on lidar 开启思岚雷达  -->
  <include file="$(find turn_on_wheeltec_robot)/launch/wheeltec_lidar.launch" />
  <include file='$(find simple_follower)/launch/nodes/laserTracker.launch' />
</launch>
```

line_follower.launch 文件内容

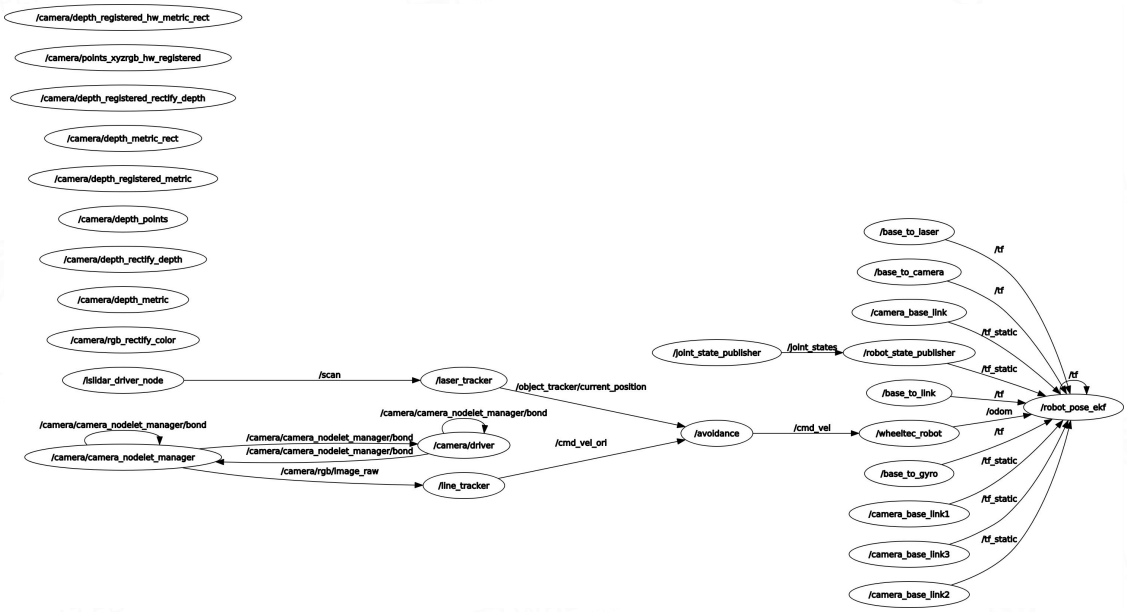
① 巡线功能启动框架：



巡线功能启动框架图

② 查看巡线功能节点：

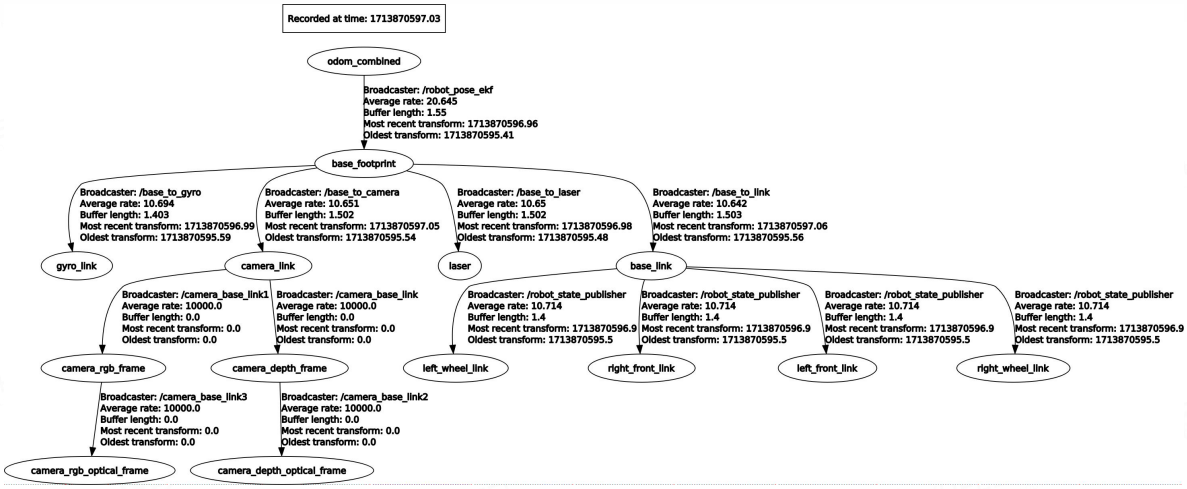
```
rqt_graph
```



巡线功能节点关系图

③ 查看巡线功能 TF 树：

```
rosrun rqt_tf_tree rqt_tf_tree
```



巡线功能 TF 树

④ **巡线节点解析：**

与巡线功能实现相关的文件主要是 simple_follower 功能包中的 line_follwer.py 文件，

1.首先设定了各种可选颜色的 HSV 阈值，不同相机对色彩的敏感程度不同，因此若要适配其他相机需要自行修改阈值，

```
def nothing(s):
    pass
col_black = (0,0,0,180,255,46)# black
col_red = (0,100,80,10,255,255)# red
col_blue = (90,90,90,110,255,255)# blue
col_green= (65,70,70,85,255,255)# green
col_yellow = (26,43,46,34,255,255)# yellow
```

2.下面定义了选择颜色的弹窗，共 5 种颜色可选，

```
cv2.namedWindow('Adjust_hsv',cv2.WINDOW_NORMAL)
Switch = '0:Red\n1:Green\n2:Blue\n3:Yellow\n4:Black'
cv2.createTrackbar(Switch,'Adjust_hsv',0,4,nothing)
```

3.初始化过程中，定义了一个发布者与订阅者，订阅 RGB 话题，同时发布速度话题，同时定义了 Twist 用于存放要发布的速度，

```
def __init__(self):
        self.bridge = cv_bridge.CvBridge()
                # 订阅 usb 摄像头
        self.image_sub = rospy.Subscriber("/camera/rgb/image_raw", Image,
self.image_callback)

        self.cmd_vel_pub = rospy.Publisher("cmd_vel_ori", Twist, queue_size=1)
        self.twist = Twist()
```

4.image_callback 回调函数是巡线时图像信息处理的关键，

```
image = self.bridge.imgmsg_to_cv2(msg, desired_encoding='bgr8')
image = cv2.resize(image, (320,240), interpolation=cv2.INTER_AREA)#提高帧率
kernel = numpy.ones((5,5),numpy.uint8)
hsv_erode = cv2.erode(hsv,kernel,iterations=1)
hsv_dilate = cv2.dilate(hsv_erode,kernel,iterations=1)
m=cv2.getTrackbarPos(Switch,'Adjust_hsv')
mask=cv2.inRange(hsv_dilate,(lowerbH,lowerbS,lowerbV),(upperbH,upperbS,upperbV)
)

masked = cv2.bitwise_and(image, image, mask=mask)
# 计算mask图像的重心，即几何中心
        M = cv2.moments(mask)
        if M['m00'] > 0:
            cx = int(M['m10']/M['m00'])
            cy = int(M['m01']/M['m00'])
            cv2.circle(image, (cx, cy), 10, (255, 0, 255), -1)
            #cv2.circle(image, (cx-75, cy), 10, (0, 0, 255), -1)
            #cv2.circle(image, (w/2, h), 10, (0, 255, 255), -1)
            if cv2.circle:
            # 计算图像中心线和目标指示线中心的距离
                erro = cx - w/2-15
                d_erro=erro-last_erro
                self.twist.linear.x = 0.18
                if erro!=0:
                    self.twist.angular.z =
-float(erro)*0.005-float(d_erro)*0.000
                else :
                    self.twist.angular.z = 0
                last_erro=erro
        else:
            self.twist.linear.x = 0
            self.twist.angular.z = 0
        self.cmd_vel_pub.publish(self.twist)
#此处为部分代码行节选，非完整内容
```

首先调用了 CvBridge 将图像信息转换为 OpenCV 图像信息的形式，然后根据阈值对图像信息进行二值化处理，提取出目标颜色并保留，对目标颜色区域进行膨胀腐蚀操作后，再通过对目标颜色色块位置进行比较与处理，判断此时小车位置是居中或者偏左偏右，通过位置的偏移量对小车角速度进行 PID 速度调节，使线一直保持在小车的中心位置，并且小车在看到目标颜色的线时就会一直保持前进，当小车视野内没有选定颜色的轨迹时角速度与线速度均被置零，小车停止

运动。

⑤ **避障节点解析：**

避障主要是通过 avoidance 节点来实现的，创建了两个订阅者，分别订阅巡线节点发布的速度以及雷达扫描节点所扫描到的障碍物的位置信息，同时创建了一个发布者，作为最终发布的速度。当检测到障碍物距离较近时，发布置为 0 的速度，如果没有遇到障碍物，则将巡线节点所发布的速度按原样进行发布。

```
/simple_follower/src/avoidance.cpp
int main(int argc, char** argv)
{
    int temp_count = 0;    //计数变量
    string str1 = "遇到障碍物";    //障碍物字符串

    ros::init(argc, argv, "avoidance");    //初始化 ROS 节点

    ros::NodeHandle node;    //创建句柄

    /***创建底盘速度控制话题发布者***/
    ros::Publisher cmd_vel_Pub = node.advertise<geometry_msgs::Twist>("cmd_vel",
1);

    /***创建底盘运动话题订阅者***/
    ros::Subscriber vel_sub = node.subscribe("cmd_vel_ori", 1,
cmd_vel_ori_Callback);

    /***创建障碍物方位话题订阅者***/
    ros::Subscriber current_position_sub =
node.subscribe("/object_tracker/current_position", 1,
current_position_Callback);

    double rate2 = 30;    //频率 30Hz
    ros::Rate loopRate2(rate2);

    while(ros::ok())
    {
        ros::spinOnce();

        if(distance_judgment() && dis_angleX_judgment())    //判断障碍物的距离和
方向
        {
            temp_count++;
            if(temp_count > 5)    //连续计数 5  次停止运动防止碰撞，避免雷达有噪点
```

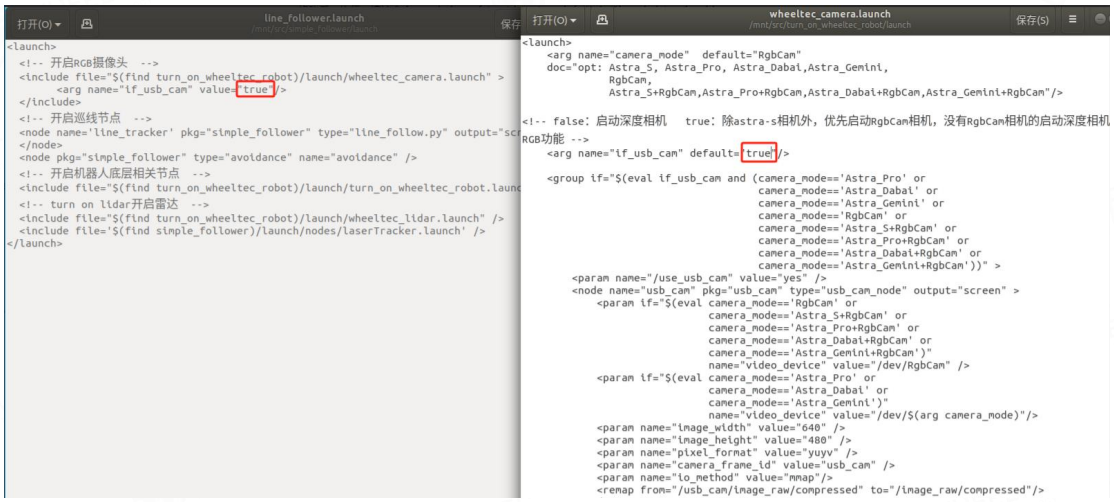
```
            {
                cmd_vel_Pub.publish(geometry_msgs::Twist());
                temp_count = 0;
            }
        }
        else
        {
            temp_count = 0;     //排除雷达噪点
            cmd_vel_Pub.publish(cmd_vel_msg);     //将速度指令发送给机器人
        }

        ros::spinOnce();
        loopRate2.sleep();
    }
    return 0;
}
```

4. 注意事项（运行后无巡线弹窗出现）

1）检查相机类型

如果相机是 RGB 相机（如：C70、C100），需要将下图两处代码修改为 true。其他型号的相机改为 false。路径在文件名下方。



切换相机类型

2）检查相机话题

根据本文档 2.3 小节操作，运行 rqt_image_view 查看相机话题，然后检查文件内的代码，话题是否与 rqt_image_view 查看的一致。如果不一致，则需要修改。如：在 rqt_image_view 查看到的话题名称是/camera/color/image_raw，但代码内

填写的是/camera/rgb/image_raw，则需要将代码改成/camera/color/image_raw

修改相机话题