

# 无人驾驶课程

## 目录

1.无人驾驶之车道保持 .....	4
1.1 准备工作 .....	4
1.2 实验原理 .....	5
1.3 实现流程 .....	6
1.4 调节摄像头位置并设置颜色阈值 .....	7
1.5 启动车道保持 .....	11
1.6 实现效果 .....	12
1.7 程序分析 .....	12
1.7.1 基础配置 .....	13
1.7.2 巡线初始化 .....	14
1.7.3 得到颜色最大面积 .....	14
1.7.4 得到转弯的 Y 参数 .....	15
1.7.5 得到巡线的 X, Y 参数 .....	15
1.7.6 图像二值化 .....	15
1.7.7 巡线回调函数 .....	17
1.7.8 巡线判定 .....	18
2.无人驾驶之路标检测 .....	19
2.1 准备工作 .....	19
2.2 实验原理 .....	20
2.3 实现流程 .....	20

2.4 实验步骤 .....	21
2.5 功能实现 .....	22
2.6 程序简要分析 .....	23
2.6.1 基础通信部分 .....	23
2.6.2 停车 .....	24
2.6.3 右转 .....	24
3.无人驾驶之红绿灯识别 .....	25
3.1 准备工作 .....	26
3.2 实现原理 .....	26
3.3 实现流程 .....	27
3.4 实验步骤 .....	27
3.5 功能实现 .....	29
3.6 参数调节说明 .....	29
4.无人驾驶之转向决策 .....	31
4.1 准备工作 .....	31
4.2 实验原理 .....	31
4.3 实现流程 .....	32
4.4 实验步骤 .....	32
4.5 功能实现 .....	34
4.6 参数调节说明 .....	34
5.无人驾驶之自主泊车 .....	36
5.1 准备工作 .....	36

5.2 实现原理 .....	36
5.3 实现流程 .....	37
5.4 实验步骤 .....	38
5.5 功能实现 .....	39
5.6 参数调节说明 .....	40
6.无人驾驶之综合应用 .....	41
6.1 准备工作 .....	41
6.2 调节摄像头位置并设置颜色阈值和置信度 .....	42
6.3 实现流程 .....	47
6.4 实验步骤 .....	48
6.5 功能实现 .....	49
6.6 Launch 文件说明 .....	49
6.6.1 启动设备 .....	50
6.1.2 启动 Yolov5 节点 .....	50
6.1.3 启动无人驾驶 .....	51
无人驾驶玩法常见问题 .....	51

# 1.无人驾驶之车道保持

小车会根据地图边缘的黄线进行巡线。



## 1.1 准备工作

---

**注意：体验此玩法时应保证在光照均匀的室内环境进行，并且避免灯以直射的方式照射地图，防止出现误识别的情况发生。**

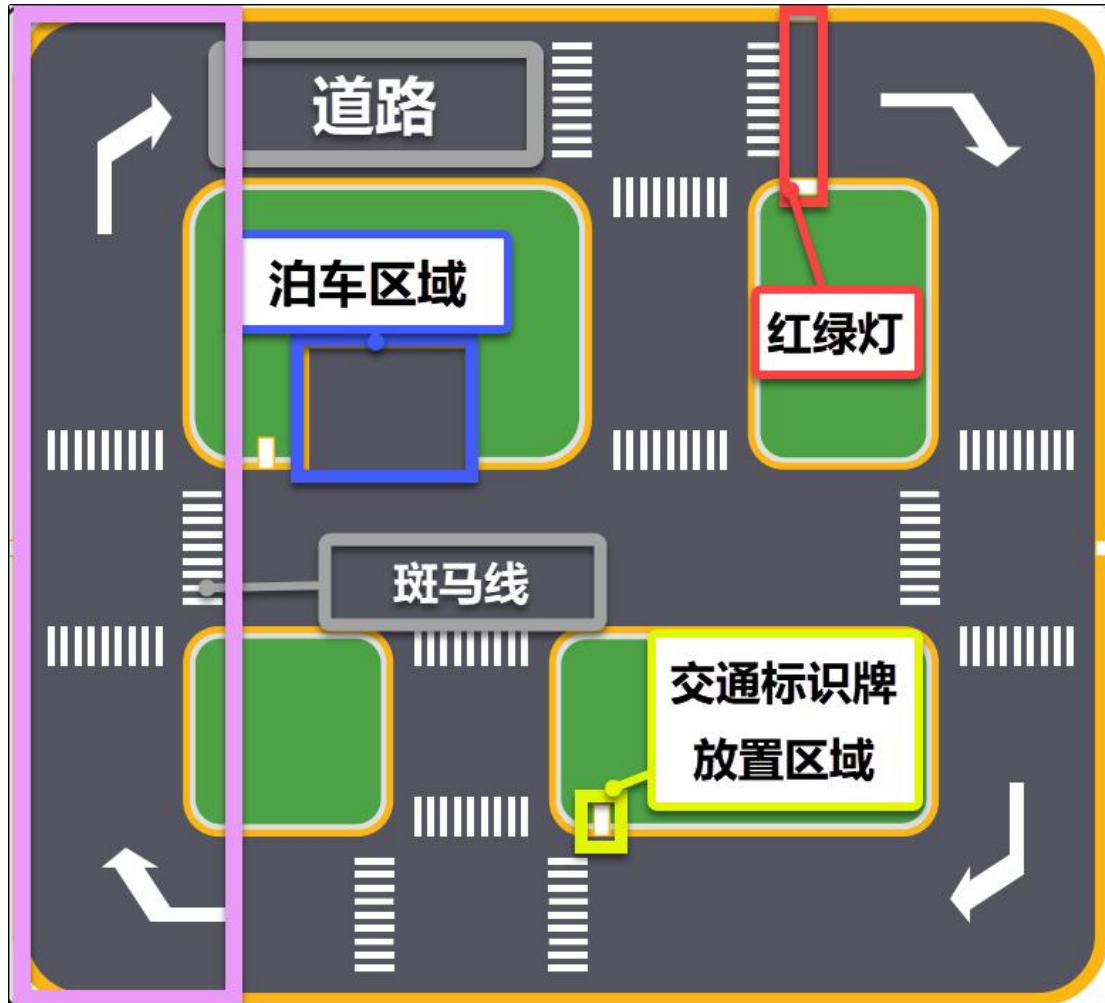
---

1) 开始前需要铺设好地图，确保地图平整无褶皱，道路畅通无障碍物，具体铺设地图可以前往与本节同目录下的“**地图铺设及道具安装**”查看学习。(本节课我们只体验道路行驶-巡线功能，因此不需要放置红绿灯和标识牌等道具)

2) 需要提前对颜色阈值进行调节，设定好黄线的颜色阈值，避免后续在识别时造成误识别，具体的颜色阈值的调节可以参考“**第 10 章 ROS+OpenCV 视觉识别标准课程/第 1 课 颜色阈值的调节**”进行参考学习。

3) 建议摆放在道路的正中间，方便识别！

4) 我们需要只需要使用到部分地图，将小车放置在下图中对应的地图位置：



## 1.2 实验原理

车道保持可以分为获取实时画面、图像处理以及结果比较三部分。

首先是获取实时画面，调用摄像头对画面进行实时拍摄；

接着是图像处理，其中包括颜色识别、对识别到的图像进行颜色空间转换、腐蚀膨胀处理、二值化处理等

结果比较部分是对处理好的图像进行图像 ROI 处理，将处理好的图像轮廓进行框选，再进行对比计算。

最后，根据比较结果，调整前进的方向，让机器人保持在道路中间。

该程序的源代码位于：

`/home/Jetauto_ws/src/Jetauto_example/scripts/self_driving/self_driving.py`

```

4 import cv2
5 import math
6 import yaml
7 import rospy
8 import threading
9 import numpy as np
10 import jetauto_sdk.pid as pid
11 import jetauto_sdk.misc as misc
12 from sensor_msgs.msg import Image
13 import geometry_msgs.msg as geo_msg
14 from jetauto_interfaces.msg import ObjectsInfo
15 from hiwonder_servo_msgs.msg import MultiRawIdPosDur
16 from hiwonder_servo_controllers.bus_servo_control import set_servos
17
18 def get_yaml_data(yaml_file):
19     yaml_file = open(yaml_file, 'r', encoding='utf-8')
20     file_data = yaml_file.read()
21     yaml_file.close()
22
23     data = yaml.load(file_data, Loader=yaml.FullLoader)
24
25     return data
26
27 lab_data = get_yaml_data("/home/jetauto/jetauto_software/lab_tool/lab_config.yaml")
28
29 class LineFollower:
30     def __init__(self, color):
31         self.target_color = color
32         self.rois = ((450, 480, 0, 320, 0.7), (390, 420, 0, 320, 0.3), (330, 360, 0, 320, 0.1))
33         self.weight_sum = 1.0
34
35     @staticmethod
36     def get_area_max_contour(contours, threshold=100):
37         """
38         获取最大面积对应的轮廓
39         :param contours:
40         :param threshold:
41         :return:
42         """
43         contour_area = zip(contours, tuple(map(lambda c: math.fabs(cv2.contourArea(c)), contours)))
44         contour_area = tuple(filter(lambda c_a: c_a[1] > threshold, contour_area))
45         if len(contour_area) > 0:

```

## 1.3 实现流程

1、将回传画面用 OpenCV 进行图像处理(转换颜色空间、腐蚀、碰撞、模糊等)，可以前往“\JetAuto\第 4 章 OpenCV 计算机视觉学习”进行学习。

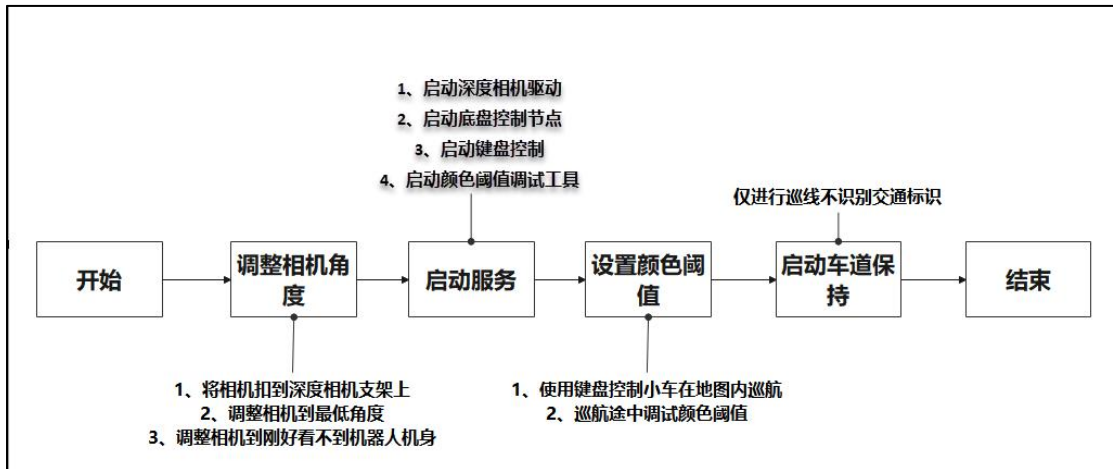
2、使用我们调节好的颜色阈值对回传画面中的黄色进行识别。可以前往“第 10 章 ROS+OpenCV 视觉识别标准课程\第 1 课 颜色阈值的调节”进行学习。

3、在识别的过程中，将画面的左半边截出，机器人识别到左边的颜色，并使用 ROI 将识别到的区域框起来。

ROI 就是在图像中选择一个区域，对这个区域内的图像重点关注，方便对这个区域的内容进一步的处理。这个区域可以是任何形状，使用 ROI 进行巡线能够减少图像处理的时间，提高判断精度。

4、通过识别框的中心点位置，使用 PID 控制机器人靠近识别框的中心。可以前往“第 7 章 JetAuto 运动控制&URDF 模型及仿真课程\1.基础控制课程课程\第 6 课 PID 算法介绍”进行学习。

5、实现流程图如下：



## 1.4 调节摄像头位置并设置颜色阈值

注意：输入指令时需要严格区分大小写和空格，且可使用“Tab”键补齐关键词。

1) 启动 JetAuto，将其连接至远程控制软件 NoMachine。

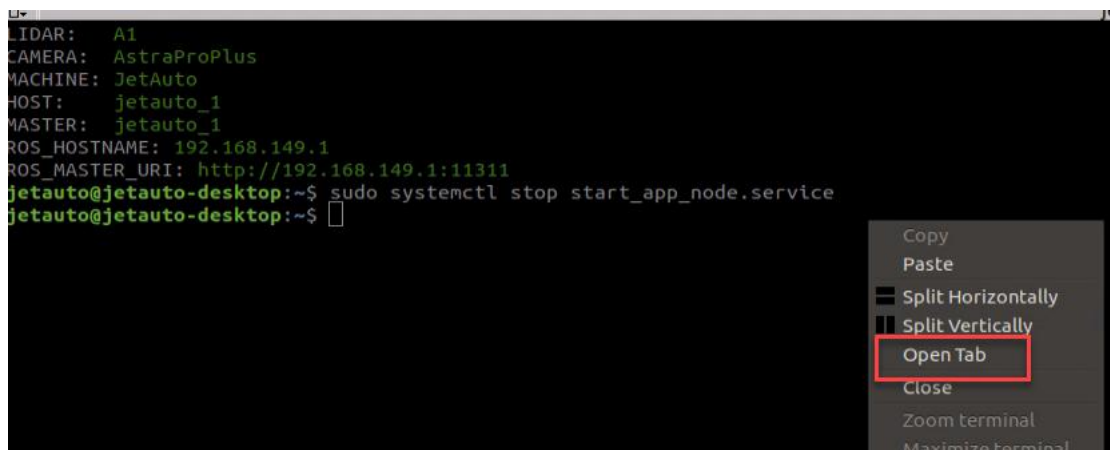


2) 点击系统桌面的左侧图标，打开命令行终端。

3) 输入指令“`sudo systemctl stop start_app_node.service`”，并按下回车，关闭手机 app 服务。

```
jetauto@jetauto-desktop:~$ sudo systemctl stop start_app_node.service
```

4) 额外创建三个命令栏标签，右键命令栏，并选择“Open Tab”。需要重复这个操作三次。



5) 在第一个命令标签输入命令：“**roslaunch jetauto\_peripherals astrapro.launch**”，启动深度相机初始化节点。

```
jetauto@jetauto-desktop:~$ roslaunch jetauto_peripherals astrapro.launch
```

6) 在第二个命令标签输入命令：“**roslaunch jetauto\_controller jetauto\_controller.launch**”，启动底盘控制节点。

```
jetauto@jetauto-desktop:~$ roslaunch jetauto_controller jetauto_controller.launch
```

7) 在第三个命令标签输入命令：

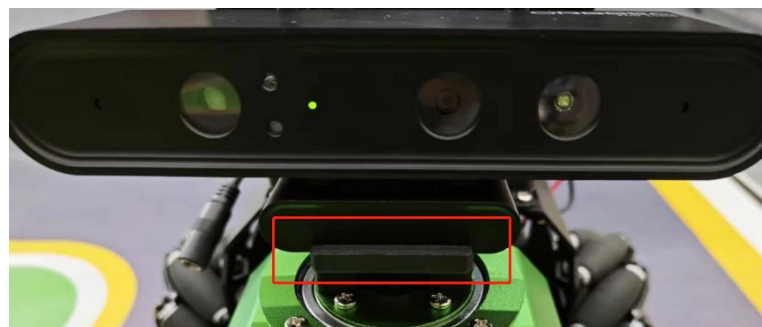
“**roslaunch jetauto\_peripherals teleop\_key\_control.launch robot\_name:="/>"**”，启动键盘控制节点。

```
jetauto@jetauto-desktop:~$ roslaunch jetauto_peripherals teleop_key_control.launch robot_name:="/>"
```

8) 在第四个命令标签输入命令：“**cd jetauto\_software/lab\_tool/ && python3 main.py**”，启动颜色阈值调节工具。

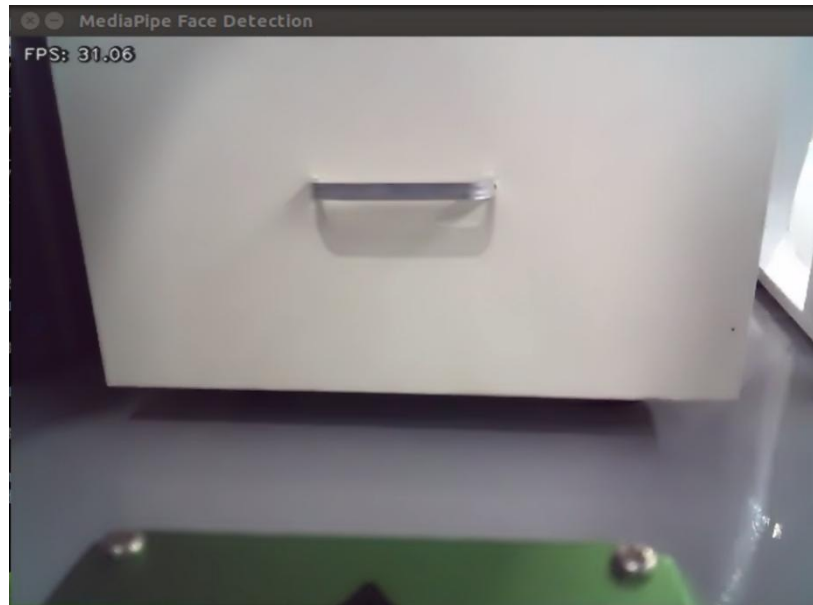
```
jetauto@jetauto-desktop:~$ cd jetauto_software/lab_tool/ && python3 main.py
```

9) 将深度相机扣在相机支架上。



10) 扭动深度相机的关节，调节深度相机的角度，使相机照到机器人机身。

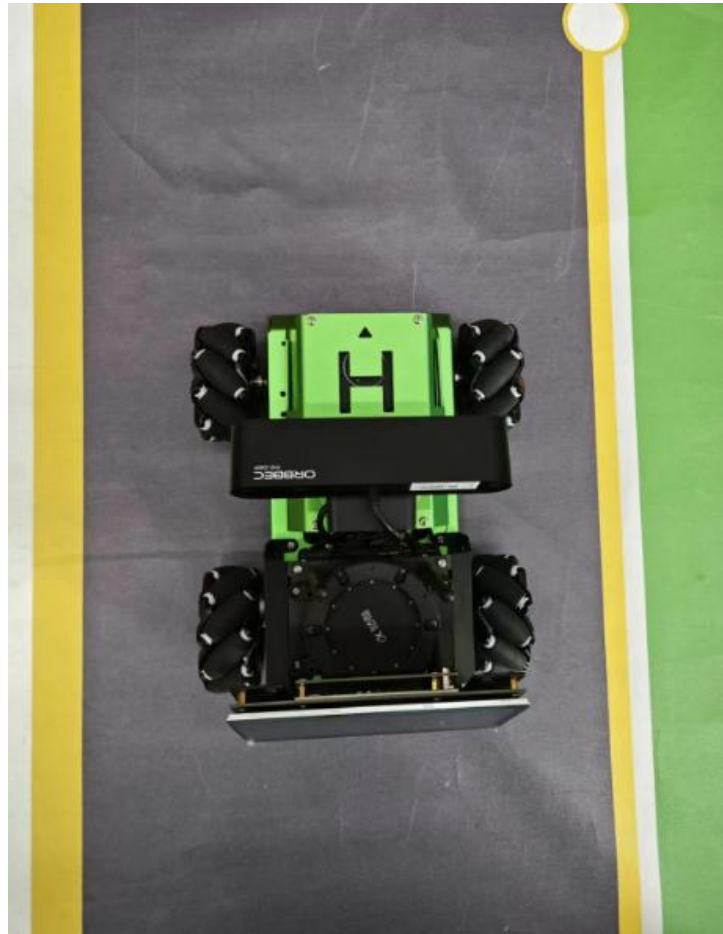




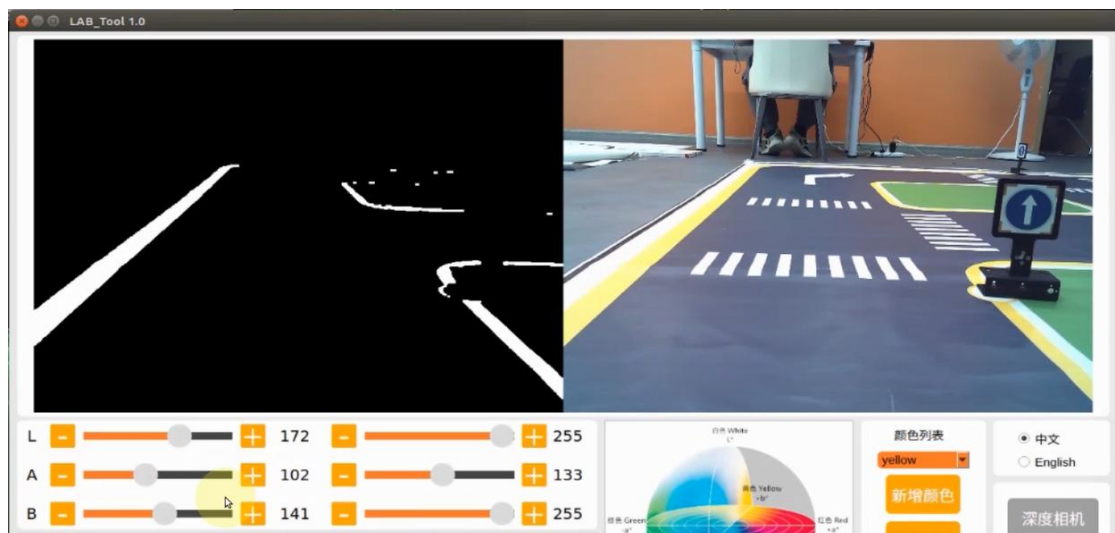
11) 将深度相机回调一点，使深度相机刚好照不到机器人机身。



12) 将机器人放置在地图上，使其在地图中道路上居中。



13) 开始调节颜色阈值，将颜色列表内的颜色切换成“yellow”，然后根据“第9章 ROS+OpenCV 视觉识别标准课程第1课 颜色阈值的调节”调节阈值。



14) 调节完成后，将机器人的窗口移动到如下图一样的位置，然后选择第三个命令栏标签，控制小车在地图上运动。观察小车能否识别到地图上的黄线，若可以则不需要调节，若无法识别某处的黄线，或者识别除黄线外的其他物体，则需要调节颜色阈值。直到，地图上

的黄线能够完全识别且不受其他物体影响。

```
/home/jetauto/jetauto_ws/src/jetauto_peripherals/launch/teleop_key_control.launch http:
/home/jetauto/jetauto_ws/... x /home/jetauto/jetauto_ws/... x /home/jetauto/jetauto_ws/... x
//home/jetauto/jetauto_ws/src/jetauto_peripherals/launch/teleop_key_control.launch http://192.168.149

PARAMETERS
* /rostdistro: melodic
* /rosversion: 1.14.13

NODES
/
teleop_key_control (jetauto_peripherals/teleop_key_control.py)

ROS_MASTER_URI=http://192.168.149.1:11311

process[teleop_key_control-1]: started with pid [15705]

Control Your Jetauto!
-----
Moving around:
      w
    a   s   d
CTRL-C to quit
```

## 1.5 启动车道保持

---

注意：输入指令时需要严格区分大小写和空格，且可使用“Tab”键补齐关键词。进行此步，必须进行第四步。

---

- 1) 启动 JetAuto，将其连接至远程控制软件 NoMachine。



- 2) 点击系统桌面的左侧图标，打开命令行终端。

- 3) 输入指令“**sudo systemctl stop start\_app\_node.service**”，并按下回车，关闭手机 app 服务。

```
jetauto@jetauto-desktop:~$ sudo systemctl stop start_app_node.service
```

- 1) 将小车放置在地图道路上后，在命令栏内输入指令“**roslaunch jetauto\_example self\_driving.launch only\_line\_follow:=true**”，这里我们在指令后加入“**only\_line\_follow:=true**”默认为 **flase**，这里我们将参数设置为 **true**，此时小车仅启动车道保持玩法。

```
jetauto@jetauto-desktop:~$ roslaunch jetauto_example self_driving.launch only_l  
ine_follow:=true
```

4) 如需关闭此玩法，可在终端界面按下“**Ctrl+C**”。若关闭失败，请反复尝试。

5) 玩法体验完毕后，可通过指令或重启机器人来开启手机 APP 服务。

若未开启手机 APP 服务，则 APP 相关功能会失效。（若机器人重启，手机 APP 服务将会自动开启）输入指令“**sudo systemctl start start\_app\_node.service**”重启手机 APP 服务，等待蜂鸣器滴的一声即服务启动完成。

```
jetauto@jetauto-desktop:~$ sudo systemctl stop start_app_node.service
```

## 1.6 实现效果

---

注意：机器人放置在地图中后，一定需要调整相机的角度，确保回传画面中不会出现机器人的机体部分的前提下尽量朝下，如果出现巡线失败，请参照开头的准备工作进行检查，特别注意调整相机角度。

---

启动玩法后，将机器人放置在地图道路中，机器人会识别道路边缘的黄线，调整自身的位置，保持在道路中间。



## 1.7 程序分析

该程序的源代码位于：

/home/jetauto\_ws/src/jetauto\_example/scripts/self\_driving/self\_driving.py

```
4 import cv2
5 import math
6 import yaml
7 import rospy
8 import threading
9 import numpy as np
10 import jetauto_sdk.pid as pid
11 import jetauto_sdk.misc as misc
12 from sensor_msgs.msg import Image
13 import geometry_msgs.msg as geo_msg
14 from jetauto_interfaces.msg import ObjectsInfo
15 from hiwonder_servo_msgs.msg import MultiRawIdPosDur
16 from hiwonder_servo_controllers.bus_servo_control import set_servos
17
18 def get_yaml_data(yaml_file):
19     yaml_file = open(yaml_file, 'r', encoding='utf-8')
20     file_data = yaml_file.read()
21     yaml_file.close()
22
23     data = yaml.load(file_data, Loader=yaml.FullLoader)
24
25     return data
26
27 lab_data = get_yaml_data("/home/jetauto/jetauto_software/lab_tool/lab_config.yaml")
28
29 class LineFollower:
30     def __init__(self, color):
31         self.target_color = color
32         self.rois = ((450, 480, 0, 320, 0.7), (390, 420, 0, 320, 0.3), (330, 360, 0, 320, 0.1))
33         self.weight_sum = 1.0
34
35     @staticmethod
36     def get_area_max_contour(contours, threshold=100):
37         """
38         获取最大面积对应的轮廓
39         :param contours:
40         :param threshold:
41         :return:
42         """
43         contour_area = zip(contours, tuple(map(lambda c: math.fabs(cv2.contourArea(c)), contours)))
44         contour_area = tuple(filter(lambda c_a: c_a[1] > threshold, contour_area))
45         if len(contour_area) > 0:
```

## 1.7.1 基础配置

### ◆ 导入所需库

```
import os
import cv2
import math
import yaml
import rospy
import threading
import numpy as np
import jetauto_sdk.pid as pid
import jetauto_sdk.misc as misc
from std_srvs.srv import Trigger
import jetauto_sdk.common as common
from sensor_msgs.msg import Image
import geometry_msgs.msg as geo_msg
from jetauto_interfaces.msg import ObjectsInfo
from hiwonder_servo_msgs.msg import MultiRawIdPosDur
from hiwonder_servo_controllers.bus_servo_control import set_servos
```

### ◆ 读取颜色阈值设置文件

通过 get\_yaml\_data 模块获取颜色阈值的配置文件。

```
def get_yaml_data(yaml_file):
    yaml_file = open(yaml_file, 'r', encoding='utf-8')
    file_data = yaml_file.read()
    yaml_file.close()

    data = yaml.load(file_data, Loader=yaml.FullLoader)

    return data

lab_data = get_yaml_data("/home/jetauto/jetauto_software/lab_tool/lab_config.yaml")
```

## ◆ 订阅节点

初始化巡线类：

```
self.follower = LineFollower("yellow")
```

发布底盘控制参数以及图像处理结果：

```
self.mecanum_pub = rospy.Publisher('/jetauto_controller/cmd_vel', geo_msg.Twist, queue_size=1) # 底盘控制
self.result_publisher = rospy.Publisher(self.name + '/image_result', Image, queue_size=1) # 图像处理结果发布
```

订阅摄像头节点：

```
rospy.Subscriber('/%s/rgb/image_raw'%depth_camera, Image, self.image_callback) # 摄像头订阅
```

## 1.7.2 巡线初始化

设置目标颜色并且设置三个 ROI 区域。

```
def __init__(self, color):
    self.target_color = color
    self.rois = ((450, 480, 0, 320, 0.7), (390, 420, 0, 320, 0.2), (330, 360, 0, 320, 0.1))
    self.weight_sum = 1.0
```

## 1.7.3 得到颜色最大面积

LineFollower 内的 get\_area\_max\_contour 函数，用于获取回传画面中识别到最大面积的黄色。需要将轮廓参数输入。

```
def get_area_max_contour(contours, threshold=100):
    """
    获取最大面积对应的轮廓
    :param contours:
    :param threshold:
    :return:
    """
    contour_area = zip(contours, tuple(map(lambda c: math.fabs(cv2.contourArea(c)), contours)))
    contour_area = tuple(filter(lambda c_a: c_a[1] > threshold, contour_area))
    if len(contour_area) > 0:
        max_c_a = max(contour_area, key=lambda c_a: c_a[1])
        return max_c_a
    return None
```





第一个参数 “img\_lab” 是输入图像；

第二个参数 “(3,3)” 是高斯卷积核的大小，卷积核的高度和宽度都必须为正数和奇数；

第三个参数 “3” 是水平方向的高斯核标准偏差。

#### ◆ 二值化处理

采用 cv2 库中的 inRange()函数对图像进行二值化处理。

```
74 mask = cv2.inRange(img_blur, tuple(lab_data['lab']['Stereo'][self.target_color]['min']),  
75 tuple(lab_data['lab']['Stereo'][self.target_color]['max'])) # 二值化
```

第一个参数 “img\_blur” 是输入图像；

第二个参数 “tuple(lab\_data['lab'][self.target\_color]['min'])” 是颜色阈值的最小值；

第三个参数 “tuple(lab\_data['lab'][self.target\_color]['max'])” 是颜色阈值的最大值。

当某像素点的 RGB 值处于颜色阈值范围内，该点被赋值为 “1”，否则赋值为 “0”。

#### ◆ 腐蚀膨胀处理

对图像进行腐蚀和膨胀处理，平滑图像内的轮廓边缘，便于后续查找目标轮廓。

```
75 eroded = cv2.erode(mask, cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))) # 腐蚀  
76 dilated = cv2.dilate(eroded, cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))) # 膨胀
```

erode()函数用于对图像进行腐蚀操作。括号内的参数含义如下：

第一个参数 “mask” 是输入图像；

第二个参数 “cv2.getStructuringElement(cv2.MORPH\_RECT, (3, 3))” 是决定操作性质的结构元素或内核。其中，括号内的第一个参数是内核形状，第二个参数是内核尺寸。

dilate()函数用于对图像进行膨胀操作。此函数括号内参数的含义与 erode()函数的相同。



## 1.7.7 巡线回调函数

```
def __call__(self, image, result_image):
    centroid_sum = 0
    h, w = image.shape[:2]
    max_center_x = -1
    center_x = []
    for roi in self.rois:
        blob = image[roi[0]:roi[1], roi[2]:roi[3]] # 截取roi
        contours = cv2.findContours(blob, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_TC89_L1)[-2] # 找轮廓
        max_contour_area = self.get_area_max_contour(contours, 30) # 获取最大面积对应轮廓
        if max_contour_area is not None:
            rect = cv2.minAreaRect(max_contour_area[0]) # 最小外接矩形
            box = np.int0(cv2.boxPoints(rect)) # 四个角
            for j in range(4):
                box[j, 1] = box[j, 1] + roi[0]
            cv2.drawContours(result_image, [box], -1, (0, 255, 255), 2) # 画出四个点组成的矩形

            # 获取矩形对角点
            pt1_x, pt1_y = box[0, 0], box[0, 1]
            pt3_x, pt3_y = box[2, 0], box[2, 1]
            # 线的中心点
            line_center_x, line_center_y = (pt1_x + pt3_x) / 2, (pt1_y + pt3_y) / 2

            cv2.circle(result_image, (int(line_center_x), int(line_center_y)), 5, (0, 0, 255), -1) # 画出中心点
            center_x.append(line_center_x)
        else:
            center_x.append(-1)
    for i in range(len(center_x)):
        if center_x[i] != -1:
            if center_x[i] > max_center_x:
                max_center_x = center_x[i]
            centroid_sum += center_x[i] * self.rois[i][-1]
    if centroid_sum == 0:
        return result_image, None, max_center_x
    center_pos = centroid_sum / self.weight_sum # 按比重计算中心点
    angle = math.degrees(-math.atan((center_pos - (w / 2.0)) / (h / 2.0)))
    return result_image, angle, max_center_x
```

### ◆ 提取ROI区域

对图像进行区域的提取，便于后续的图像对比判断。

```
85         for roi in self.rois:
86             blob = image[roi[0]:roi[1], roi[2]:roi[3]] # 截取roi
```

blob()函数用于对图像进行 ROI 区域的截取。括号内的参数含义如下：

第一个参数“roi[0]:roi[1]”是相机从获取画面中截取的 roi 区域的 x 轴最大值和最小值

第二个参数“roi[2]:roi[3]”是相机从获取画面中截取的 roi 区域的 y 轴最大值和最小值

### ◆ 获取最大面积轮廓

通过调用 cv2 库中的 findContours()函数，查找图像内目标识别颜色的最大轮廓。

```
87         contours = cv2.findContours(blob, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_TC89_L1)[-2] # 找轮廓
```

第一个参数“blob”是通过 ROI 截取所得的图像；

第二个参数“cv2.RETR\_EXTERNAL”是轮廓检索模式；

第三个参数“cv2.CHAIN\_APPROX\_TC89\_L1”是轮廓近似方法。

### ◆ 获取最大面积轮廓

1、通过面积来得到最小外接矩形的四个角。

```
rect = cv2.minAreaRect(max_contour_area[0]) # 最小外接矩形
```

2、计算矩形四个角的坐标点。

```
box = np.int0(cv2.boxPoints(rect)) # 四个角
```

3、使用 ROI 来将所需巡线的矩形框出来。

```
for j in range(4):  
    box[j, 1] = box[j, 1] + roi[0]  
cv2.drawContours(result_image, [box], -1, (0, 255, 255), 2) # 画出四个点组成的矩形
```

4、根据矩形的坐标点来得到矩形对角点，通过对角点计算线的中心点坐标，然后将中心点画出来。

```
# 获取矩形对角点  
pt1_x, pt1_y = box[0, 0], box[0, 1]  
pt3_x, pt3_y = box[2, 0], box[2, 1]  
# 线的中心点  
line_center_x, line_center_y = (pt1_x + pt3_x) / 2, (pt1_y + pt3_y) / 2  
  
cv2.circle(result_image, (int(line_center_x), int(line_center_y)), 5, (0, 0, 255), -1) # 画出中心点  
center_x.append(line_center_x)
```

5、将得到的最大 center\_x 与 roi 的比例相乘，得到重心点。

```
for i in range(len(center_x)):  
    if center_x[i] != -1:  
        if center_x[i] > max_center_x:  
            max_center_x = center_x[i]  
            centroid_sum += center_x[i] * self.rois[i][-1]
```

6、通过重心点计算得到中心点，通过中心点与画面中点的偏差计算偏差角度。

```
if centroid_sum == 0:  
    return result_image, None, max_center_x  
center_pos = centroid_sum / self.weight_sum # 按比重计算中心点  
angle = math.degrees(-math.atan((center_pos - (w / 2.0)) / (h / 2.0)))
```

## 1.7.8 巡线判定

1、仅需关注红框内内容，默认情况下，小车的速度为 0.15 米/S。

```

def image_proc(self):
    if self.image is not None:
        while self.is_running:
            twist = geo_msg.Twist()
            binary_image = self.follower.get_binary(self.image)
            if 600 < self.park_pos[0] and 150 < self.park_pos[1] and not self.start_park:
                self.mecanum_pub.publish(geo_msg.Twist())
                self.start_park = True
                threading.Thread(target=self.park_action).start()
                self.stop = True
            if 0 < self.park_pos[0]:
                self.park = True
            if 1500 < self.right_area:
                self.turn_right = True
            if 440 < self.min_distance < 460:
                self.count_cross_walk += 1
                if self.count_cross_walk == 5:
                    self.count_cross_walk = 0
                    self.start_slow_down = True
                    self.count_slow_down = rospy.get_time()
            else:
                self.count_cross_walk = 0
            if self.start_slow_down:
                if self.red:
                    self.mecanum_pub.publish(geo_msg.Twist())
                    self.stop = True
                elif self.green:
                    twist.linear.x = 0.1
                    self.stop = False
                else:
                    twist.linear.x = 0.1
                    if rospy.get_time() - self.count_slow_down > self.cross_walk_distance/twist.linear.x:
                        self.start_slow_down = False
            else:
                twist.linear.x = 0.15
            trv:

```

2、若偏离线过大时，机器人的角速度会被设置成-0.45 米每秒，用于大幅靠近黄线。若检测到的直行次数超过了六次，则关闭 YOLOv5 识别的功能，将机器人 PID 的目标点设置成 80，并实时更新角速度，并发布出去。

```

result_image, angle, x = self.follower(binary_image, self.image.copy())
if x != -1 and not self.stop:
    if x > 200:
        if self.turn_right:
            self.start_turn = True
            twist.angular.z = -0.45
    else:
        if self.start_turn:
            self.count_turn += 1
        else:
            self.count_turn = 0
        if self.count_turn > 6:
            self.count_turn = 0
            self.start_turn = False
            self.turn_right = False
            self.pid.SetPoint = 80
            self.pid.update(x)
            twist.angular.z = misc.set_range(self.pid.output, -0.8, 0.8)
        self.mecanum_pub.publish(twist)
    else:
        self.pid.clear()

```

## 2.无人驾驶之路标检测

本节课是通过指令来实现对路标进行识别的。

### 2.1 准备工作

**注意：**体验此玩法时应保证在光照均匀的室内环境进行，并且避免灯以直射的方式照射地图，防止出现误识别的情况发生。

- 1) 开始前需要铺设好地图，确保地图平整无褶皱，道路畅通无障碍物，具体铺设地图

可以前往上级目录“2.AI 视觉无人驾驶玩法\1.无人驾驶道具摆放及调试视频”查看学习。

2) 本节的的路标模型为 yolov5 训练的模型，相关的 yolov5 内容可以前往“11 十章 ROS+机器学习课程”查看了解。

3) 建议摆放在道路的正中间，方便识别！

## 2.2 实验原理

首先，获取摄像头的实时画面，对图像进行腐蚀、膨胀等操作。

然后，通过 Yolov5 调用模型，与目标画面图像进行比较。

最后，根据比较结果，执行对应的路标动作。

该程序的源代码位于：

**/home/Jetauto\_ws/src/Jetauto\_example/scripts/yolov5\_detect/yolov5\_trt.py**

```
69 class YoLov5TRT(object):
70     """
71     description: A YOLOv5 class that warps TensorRT ops, preprocess and postprocess ops.
72     """
73
74     def __init__(self, engine_file_path, plugin, classes, conf_thresh=0.8, iou_threshold=0.4):
75         self.CONF_THRESH = conf_thresh
76         self.IOU_THRESHOLD = iou_threshold
77
78         PLUGIN_LIBRARY = plugin
79         self.engine_file_path = engine_file_path
80
81         # load labels
82         self.categories = classes
83
84         ctypes.CDLL(PLUGIN_LIBRARY)
85
86         # Create a Context on this device,
87         self.ctx = cuda.Device(0).make_context()
88         stream = cuda.Stream()
89         TRT_LOGGER = trt.Logger(trt.Logger.INFO)
90         runtime = trt.Runtime(TRT_LOGGER)
91
92         # Deserialize the engine from file
93         with open(self.engine_file_path, "rb") as f:
94             engine = runtime.deserialize_cuda_engine(f.read())
95             context = engine.create_execution_context()
96
97         host_inputs = []
98         cuda_inputs = []
99         host_outputs = []
100         cuda_outputs = []
101         bindings = []
```

## 2.3 实现流程

1、我们需要制作好我们需要使用的数据集，需要收集直行、右转、停车、红色红绿灯、绿色红绿灯、斑马线。

2、使用 labelimg 对图像进行标定，框出图像中我们需要训练的内容。

3、将数据的格式修改后，对数据集进行切分。

4、将数据集输入 Yolov5 内进行训练，得到可以识别交通表示相关标志的模型。

得到的新模型就可以对我们训练的内容进行识别，机器人会做出相应的反馈动作。若想深入了解可以参考路径“第 11 章 ROS+机器学习课程\机器学习实战应用”进行学习。

这里我们会使用机器人内有提前训练好，用于识别交通相关标志的 Yolov5 模型。将我们摄像头得到的回传画面输入到 Yolov5 模型内，模型会在回传画面中将交通相关标志给框选出来，并且标注其的置信度。

## 2.4 实验步骤

---

注意：下面的步骤仅开启在回传画面中检测路标的效果，不会执行对应的动作，想直接体验无人驾驶玩法的用户可跳过本节课，前往与本节课同目录下的“[6.无人驾驶之综合应用](#)”查看。

---

输入指令时需要严格区分大小写和空格，且可使用“Tab”键补齐关键词。

---

1) 启动机器人，将其连接至远程控制软件 NoMachine。

2) 双击系统桌面的图标，打开命令行终端。

3) 输入指令“**sudo systemctl stop start\_app\_node.service**”，并按下回车，关闭手机 app 服务。

```
jetauto@jetauto-desktop:~$ sudo systemctl stop start_app_node.service
```

4) 输入指令“**roscd hiwonder\_example/scripts/yolov5\_detect/**”，进入程序目录。

```
jetauto@jetauto-desktop:~$ roscd jetauto_example/scripts/yolov5_detect
```

5) 输入指令“**python3 yolov5\_trt.py**”，启动玩法程序。

```
jetauto@jetauto-desktop:~/jetauto_ws/src/jetauto_example/scripts/yolov5_detect$ python3 yolov5_trt_7_0.py
```

6) 放置路标于摄像头前，会识别出路标并标识于画面中，如需关闭此玩法，可在终端界面按下“**Ctrl+C**”。若关闭失败，请反复尝试。

7) 玩法体验完毕后，可通过指令或重启机器人来开启手机 APP 服务。

若未开启手机 APP 服务，则 APP 相关功能会失效。（若机器人重启，手机 APP 服务将会

自动开启) 输入指令 “`sudo systemctl start start_app_node.service`” 重启手机 APP 服务，等待蜂鸣器滴的一声即服务启动完成。

```
jetauto@jetauto-desktop:~$ sudo systemctl start start_app_node.service
```

注意：如果模型识别不到交通相关标志，我们则需要将置信度减小；如果模型误识别交通相关标志，我们则需要将置信度增大。

1) 输入指令 “`cd jetauto_ws/src/jetauto_example/scripts/self_driving`”，进入玩法程序路径。

```
jetauto@jetauto-desktop:~$ cd jetauto_ws/src/jetauto_example/scripts/self_driving
```

2) 输入指令 “`vim self_driving.launch`”，打开玩法程序。

```
vim self_driving.launch
```

红框中为置信度的值，可修改来调整目标检测的效果。

```
/usr/bin/zsh 80x24
1 <?xml version="1.0"?>
2 <launch>
3   <arg name="only_line_follow" default="false"/>
4   <include file="$(find hiwonder_peripherals)/launch/depth_cam.launch"/>
5   <!-- 底盘驱动(chassis driver)-->
6   <include file="$(find hiwonder_controller)/launch/hiwonder_controller.launch"/>
7
8   <arg name="node_name" default="yolov5" />
9   <rosparam param="$(arg node_name)/classes">['go', 'right', 'park', 'red', 'green', 'crosswalk']</rosparam>
10  <node unless="$(arg only_line_follow)" pkg="hiwonder_example" type="yolov5_node.py" name="$(arg node_name)" output="screen">
11    <param name="use_depth_cam" value="true"/>
12    <param name="engine" value="traffic_signs_640s_7_0.engine"/>
13    <param name="lib" value="libmyplugins_640.so"/>
14    <param name="conf_thresh" value="0.8"/>
15  </node>
```

## 2.5 功能实现

注意：机器人放置在地图中后，需要调整相机的角度，确保回传画面中不会出现机器人的机体部分的前提下尽量朝下。

启动玩法后，将机器人放置在地图道路中，当机器人中识别到路标后，会对应将识别到的路标框选出来并且标注出与学习过的模型置信度最大的。





## 2.6 程序简要分析

巡线部分请前往“第一课 无人驾驶之车道保持”学习，该程序的源代码位于：

`/home/jetauto_ws/src/jetauto_example/scripts/self_driving/self_driving.py`

```

4  import cv2
5  import math
6  import yaml
7  import rospy
8  import threading
9  import numpy as np
10 import jetauto_sdk.pid as pid
11 import jetauto_sdk.misc as misc
12 from sensor_msgs.msg import Image
13 import geometry_msgs.msg as geo_msg
14 from jetauto_interfaces.msg import ObjectsInfo
15 from hiwonder_servo_msgs.msg import MultiRawIdPosDur
16 from hiwonder_servo_controllers.bus_servo_control import set_servos
17
18 def get_yaml_data(yaml_file):
19     yaml_file = open(yaml_file, 'r', encoding='utf-8')
20     file_data = yaml_file.read()
21     yaml_file.close()
22
23     data = yaml.load(file_data, Loader=yaml.FullLoader)
24
25     return data
26
27 lab_data = get_yaml_data("/home/jetauto/jetauto_software/lab_tool/lab_config.yaml")
28
29 class LineFollower:
30     def __init__(self, color):
31         self.target_color = color
32         self.rois = ((450, 480, 0, 320, 0.7), (390, 420, 0, 320, 0.3), (330, 360, 0, 320, 0.1))
33         self.weight_sum = 1.0
34
35     @staticmethod
36     def get_area_max_contour(contours, threshold=100):
37         """
38         获取最大面积对应的轮廓
39         :param contours:
40         :param threshold:
41         :return:
42         """
43         contour_area = zip(contours, tuple(map(lambda c: math.fabs(cv2.contourArea(c)), contours)))
44         contour_area = tuple(filter(lambda c_a: c_a[1] > threshold, contour_area))
45         if len(contour_area) > 0:

```

### 2.6.1 基础通信部分

得到交通标识的类别:

```
self.classes = rospy.get_param('/yolov5_node/classes')
```

得到 yolov5 的检测结果:

```
rospy.Subscriber('/yolov5/object_detect', ObjectsInfo, self.get_object_callback)
```

### 2.6.2 停车

停车时, 运行 “**park\_action**” 函数内容。初始化 twist, 将线速度设置为 0.15m/s, 发布 twist 并运行 (0.25/0.15) S, 运行完毕后, 重复初始化并将角速度设置成-0.15 米每秒, 发布 twist 并运行 (0.25/0.15) S。

```
def park_action(self):  
    twist = geo_msg.Twist()  
    twist.linear.x = 0.15  
    self.mecanum_pub.publish(twist)  
    rospy.sleep(0.25/0.15)  
    twist = geo_msg.Twist()  
    twist.linear.y = -0.15  
    self.mecanum_pub.publish(twist)  
    rospy.sleep(0.25/0.15)  
    self.mecanum_pub.publish(geo_msg.Twist())
```

在主函数内, 若检测到停车标识则启动停车线程, 当停车标识到一定大小的时候, 开始运行 “**park\_action**” 函数的内容。

```
def image_proc(self):  
    if self.image is not None:  
        while self.is_running:  
            twist = geo_msg.Twist()  
            binary_image = self.follower.get_binary(self.image)  
            if 600 < self.park_pos[0] and 150 < self.park_pos[1] and not self.start_park:  
                self.mecanum_pub.publish(geo_msg.Twist())  
                self.start_park = True  
                threading.Thread(target=self.park_action).start()  
                self.stop = True  
            if 0 < self.park_pos[0]:
```

### 2.6.3 右转

主函数内, 若识别到右转交通标识牌, 其面积大于 1500, 则启动右转。

```
self.park = True  
if 1500 < self.right_area:  
    self.turn_right = True  
if 110 < self.min_distance < 160:
```

若启动右转, 则从巡线类的 “get\_max\_y” 函数, 得到巡线需要的 y 值, 重设 roi, 并对其区域画线。

若 self.turn\_right 参数为 true, 则开始左转, 反之, 接着巡线。



```

if self.turn_right:
    y = self.follower.get_max_y(binary_image)
    roi = [(0, y), (640, y), (640, 0), (0, 0)]
    cv2.fillPoly(binary_image, [np.array(roi)], [0, 0, 0])
    min_x = cv2.minMaxLoc(binary_image)[-1][0]
    cv2.line(binary_image, (min_x, y), (640, y), (255, 255, 255), 10)
elif self.stop:

```

```

result_image, angle, x = self.follower(binary_image, self.image.copy())
if x != -1 and not self.stop:
    if x > 200:
        if self.turn_right:
            self.start_turn = True
            twist.angular.z = -0.45
        else:
            if self.start_turn:
                self.count_turn += 1
            else:
                self.count_turn = 0
            if self.count_turn > 6:
                self.count_turn = 0
                self.start_turn = False
                self.turn_right = False
            self.pid.SetPoint = 80
            self.pid.update(x)
            twist.angular.z = misc.set_range(self.pid.output, -0.8, 0.8)
        self.mecanum_pub.publish(twist)
    else:
        self.pid.clear()
        self.follower = None

```

#### 2.6.4 斑马线和红绿灯

若连续五次检测到斑马线的面积大于 440 小于 460，则启动开启斑马线相关程序。

```

if 440 < self.min_distance < 460:
    self.count_cross_walk += 1
    if self.count_cross_walk == 5:
        self.count_cross_walk = 0
        self.start_slow_down = True
        self.count_slow_down = rospy.get_time()

```

若启动斑马线相关程序，则开始检测面前是否有红灯或绿灯，若有红灯则小车停止，若有绿灯小车则以 0.1m/s 的速度行进，四秒过后将斑马线程序关闭。

```

if self.start_slow_down:
    if self.red:
        self.mecanum_pub.publish(gemo_msg.Twist())
        self.stop = True
    elif self.green:
        twist.linear.x = 0.1
        self.stop = False
    else:
        twist.linear.x = 0.1
        if rospy.get_time() - self.count_slow_down > self.cross_walk_distance/twist.linear.x:
            self.start_slow_down = False
else:
    twist.linear.x = 0.15

```

### 3.无人驾驶之红绿灯识别

本玩法是通过指令来实现摄像头对红绿灯的识别。



### 3.1 准备工作

---

**注意：**体验此玩法时应保证在光照均匀的室内环境进行，并且避免灯以直射的方式照射地图，防止出现误识别的情况发生。

---

- 1) 开始前需要铺设好地图，确保地图平整无褶皱，道路畅通无障碍物，具体铺设地图可以前往与本节同目录下的“地图铺设及道具安装”查看学习。
- 2) 开始前需要铺设好地图，确保地图平整无褶皱，道路畅通无障碍物，具体铺设地图可以前往上级目录“2.AI 视觉无人驾驶玩法\1.无人驾驶道具摆放及调试视频”查看学习。
- 3) 本节的红绿灯模型为 yolov5 训练的模型，相关的 yolov5 内容可以前往“第 11 章 ROS+机器学习课程”查看了解。
- 4) 建议摆放在道路的正中间，方便识别！

### 3.2 实现原理

首先，获取摄像头的实时画面，对图像进行腐蚀、膨胀等操作。

然后，通过 Yolov5 调用模型，与目标画面图像进行比较。

最后，根据比较结果，执行对应的路标动作。

该程序的源代码位于：

**/home/Jetauto\_ws/src/Jetauto\_example/scripts/yolov5\_detect/yolov5\_trt.py**

```
69 class YoLov5TRT(object):
70     """
71     description: A YOLOv5 class that warps TensorRT ops, preprocess and postprocess ops.
72     """
73
74     def __init__(self, engine_file_path, plugin, classes, conf_thresh=0.8, iou_threshold=0.4):
75         self.CONF_THRESH = conf_thresh
76         self.IOU_THRESHOLD = iou_threshold
77
78         PLUGIN_LIBRARY = plugin
79         self.engine_file_path = engine_file_path
80
81         # load labels
82         self.categories = classes
83
84         ctypes.CDLL(PLUGIN_LIBRARY)
85
86         # Create a Context on this device,
87         self.ctx = cuda.Device(0).make_context()
88         stream = cuda.Stream()
89         TRT_LOGGER = trt.Logger(trt.Logger.INFO)
90         runtime = trt.Runtime(TRT_LOGGER)
91
92         # Deserialize the engine from file
93         with open(self.engine_file_path, "rb") as f:
94             engine = runtime.deserialize_cuda_engine(f.read())
95         context = engine.create_execution_context()
96
97         host_inputs = []
98         cuda_inputs = []
99         host_outputs = []
100         cuda_outputs = []
101         bindings = []
```

### 3.3 实现流程

1、我们需要制作好我们需要使用的数据集，需要收集直行、右转、停车、红色红绿灯、绿色红绿灯、斑马线。

4、使用 `labelimg` 对图像进行标定，框出图像中我们需要训练的内容。

5、将数据的格式修改后，对数据集进行切分。

4、将数据集输入 YOLOv5 内进行训练，得到可以识别交通表示相关标志的模型。

得到的新模型就可以对我们训练的内容进行识别，机器人会做出相应的反馈动作。若想深入了解可以参考路径“**第 11 章 ROS+机器学习课程\机器学习实战应用**”进行学习。

这里我们会使用机器人内有提前训练好，用于识别交通相关标志的 YOLOv5 模型。将我们摄像头得到的回传画面输入到 YOLOv5 模型内，模型会在回传画面中将交通相关标志给框选出来，并且标注其的置信度。

### 3.4 实验步骤

---

**注意：**下面的步骤仅开启在回传画面中检测路标的效果，不会执行对应的动作，想直接体验无人驾驶玩法的用户可跳过本节课，前往与本节课同目录下的“[6.无人驾驶之综合应用](#)”查看。

---

---

输入指令时需要严格区分大小写和空格，且可使用“Tab”键补齐关键词。

---

1) 启动机器人，将其连接至远程控制软件 NoMachine。

2) 双击系统桌面的图标，打开命令行终端。

3) 输入指令“**sudo systemctl stop start\_app\_node.service**”，并按下回车，关闭手机 app 服务。

```
jetauto@jetauto-desktop:~$ sudo systemctl stop start_app_node.service
```

4) 输入指令“**roscd hiwonder\_example/scripts/yolov5\_detect/**”，进入程序目录。

```
jetauto@jetauto-desktop:~$ roscd jetauto_example/scripts/yolov5_detect
```

5) 输入指令“**python3 yolov5\_trt.py**”，启动玩法程序。

```
jetauto@jetauto-desktop:~/jetauto_ws/src/jetauto_example/scripts/yolov5_detect$ python3 yolov5_trt_7_0.py
```

6) 放置路标于摄像头前，会识别出路标并标识于画面中，如需关闭此玩法，可在终端界面按下“**Ctrl+C**”。若关闭失败，请反复尝试。

7) 玩法体验完毕后，可通过指令或重启机器人来开启手机 APP 服务。

若未开启手机 APP 服务，则 APP 相关功能会失效。（若机器人重启，手机 APP 服务将会自动开启）输入指令“**sudo systemctl start start\_app\_node.service**”重启手机 APP 服务，等待蜂鸣器滴的一声即服务启动完成。

```
jetauto@jetauto-desktop:~$ sudo systemctl start start_app_node.service
```

---

注意：如果模型识别不到交通相关标志，我们则需要将置信度减小；如果模型误识别交通相关标志，我们则需要将置信度增大。

3) 输入指令“**cd ros\_ws/src/hiwonder\_example/scripts/self\_driving**”，进入程序路径。

```
cd ros_ws/src/hiwonder_example/scripts/self_driving
```

4) 输入指令“**vim self\_driving.launch**”，打开玩法程序。



```
vim self_driving.launch
```

红框中为置信度的值，可修改来调整目标检测的效果。

```
1 <?xml version="1.0"?>
2 <launch>
3   <arg name="only_line_follow" default="false" />
4   <include file="$(find hiwonder_peripherals)/launch/depth_cam.launch" />
5   <!-- 底盘驱动(chassis driver)-->
6   <include file="$(find hiwonder_controller)/launch/hiwonder_controller.launch" />
7
8   <arg name="node_name" default="yolov5" />
9   <rosparam param="/$(arg node_name)/classes">['go', 'right', 'park', 'red', 'green', 'crosswalk']</rosparam>
10  <node unless="$(arg only_line_follow)" pkg="hiwonder_example" type="yolov5_node.py" name="$(arg node_name)" output="screen">
11    <param name="use_depth_cam" value="true" />
12    <param name="engine" value="traffic_signs_640s_7_0.engine" />
13    <param name="lib" value="libmyplugins_640.so" />
14    <param name="conf_thresh" value="0.8" />
15  </node>
```

### 3.5 功能实现

注意：机器人放置在地图中后，需要调整相机的角度，确保回传画面中不会出现机器人的机体部分，否则可能会影响玩法实现。

启动玩法后，将机器人放置在地图道路中，当机器人前进到红绿灯处，会识别信号灯的颜色，红色会停止，绿色则减速前进。



### 3.6 参数调节说明

我们再进行无人驾驶玩法时，若小车无法识别到红绿灯，则需要前往“[2.无人驾驶之路标检测](#)”学习。

该程序的源代码位于：

`/home/jetauto_ws/src/jetauto_example/scripts/self_driving/self_driving.py`

```
4 import cv2
5 import math
6 import yaml
7 import rospy
8 import threading
9 import numpy as np
10 import jetauto_sdk.pid as pid
11 import jetauto_sdk.misc as misc
12 from sensor_msgs.msg import Image
13 import geometry_msgs.msg as geo_msg
14 from jetauto_interfaces.msg import ObjectsInfo
15 from hiwonder_servo_msgs.msg import MultiRawIdPosDur
16 from hiwonder_servo_controllers.bus_servo_control import set_servos
17
18 def get_yaml_data(yaml_file):
19     yaml_file = open(yaml_file, 'r', encoding='utf-8')
20     file_data = yaml_file.read()
21     yaml_file.close()
22
23     data = yaml.load(file_data, Loader=yaml.FullLoader)
24
25     return data
26
27 lab_data = get_yaml_data("/home/jetauto/jetauto_software/lab_tool/lab_config.yaml")
28
29 class LineFollower:
30     def __init__(self, color):
31         self.target_color = color
32         self.rois = ((450, 480, 0, 320, 0.7), (390, 420, 0, 320, 0.3), (330, 360, 0, 320, 0.1))
33         self.weight_sum = 1.0
34
35     @staticmethod
36     def get_area_max_contour(contours, threshold=100):
37         """
38         获取最大面积对应的轮廓
39         :param contours:
40         :param threshold:
41         :return:
42         """
43         contour_area = zip(contours, tuple(map(lambda c: math.fabs(cv2.contourArea(c)), contours)))
44         contour_area = tuple(filter(lambda c_a: c_a[1] > threshold, contour_area))
45         if len(contour_area) > 0:
```

若启动斑马线相关程序，则开始检测面前是否有红灯或绿灯，若有红灯则小车停止，若有绿灯小车则以 0.1m/s 的速度前进，四秒过后将斑马线程序关闭。。

```
if self.start_slow_down:
    if self.red:
        self.mecanum_pub.publish(geo_msg.Twist())
        self.stop = True
    elif self.green:
        twist.linear.x = 0.1 |
        self.stop = False
    else:
        twist.linear.x = 0.1
        if rospy.get_time() - self.count_slow_down > self.cross_walk_distance/twist.linear.x:
            self.start_slow_down = False
else:
    twist.linear.x = 0.15
```

若想添加或修改识别到红绿灯后的内容，可以在下列判断内添加对应的代码。

```
if self.red:
    self.mecanum_pub.publish(geo_msg.Twist())
    self.stop = True
elif self.green:
    twist.linear.x = 0.1 |
    self.stop = False
```

## 4.无人驾驶之转向决策

本节课是通过指令来实现检测识别转向路标的。

### 4.1 准备工作

---

**注意：体验此玩法时应保证在光照均匀的室内环境进行，并且避免灯以直射的方式照射地图，防止出现误识别的情况发生。**

---

1) 开始前需要铺设好地图，确保地图平整无褶皱，道路畅通无障碍物，具体铺设地图可以前往上级目录“2.AI 视觉无人驾驶玩法\1.无人驾驶道具摆放及调试视频”查看学习。

2) 本节的的路标模型为 yolov5 训练的模型，相关的 yolov5 内容可以前往“第 10 章 ROS+机器学习课程”查看了解。

3) 建议摆放在道路的正中间，方便识别！

### 4.2 实验原理

首先，获取摄像头的实时画面，对图像进行腐蚀、膨胀等操作。

然后，通过 Yolov5 调用模型，与目标画面图像进行比较。

最后，根据比较结果，识别转向路标，让小车向指定方向前进。

该程序的源代码位于：

**/home/Jetauto\_ws/src/Jetauto\_example/scripts/yolov5\_detect/yolov5\_trt.py**

```
4 import cv2
5 import math
6 import yaml
7 import rosp
8 import threading
9 import numpy as np
10 import jetauto_sdk.pid as pid
11 import jetauto_sdk.misc as misc
12 from sensor_msgs.msg import Image
13 import geometry_msgs.msg as geo_msg
14 from jetauto_interfaces.msg import ObjectsInfo
15 from hiwonder_servo_msgs.msg import MultiRawIdPosDur
16 from hiwonder_servo_controllers.bus_servo_control import set_servos
17
18 def get_yaml_data(yaml_file):
19     yaml_file = open(yaml_file, 'r', encoding='utf-8')
20     file_data = yaml_file.read()
21     yaml_file.close()
22
23     data = yaml.load(file_data, Loader=yaml.FullLoader)
24
25     return data
26
27 lab_data = get_yaml_data("/home/jetauto/jetauto_software/lab_tool/lab_config.yaml")
28
29 class LineFollower:
30     def __init__(self, color):
31         self.target_color = color
32         self.rois = ((450, 480, 0, 320, 0.7), (390, 420, 0, 320, 0.3), (330, 360, 0, 320, 0.1))
33         self.weight_sum = 1.0
34
35     @staticmethod
```

## 4.3 实现流程

1、我们需要制作好我们需要使用的数据集，需要收集直行、右转、停车、红色红绿灯、绿色红绿灯、斑马线。

2、使用 labelimg 对图像进行标定，框出图像中我们需要训练的内容。

3、将数据的格式修改后，对数据集进行切分。

4、将数据集输入 YOLOv5 内进行训练，得到可以识别交通表示相关标志的模型。

得到的新模型就可以对我们训练的内容进行识别，机器人会做出相应的反馈动作。若想深入了解可以参考路径“第 11 章 ROS+机器学习课程\机器学习实战应用”进行学习。

这里我们会使用机器人内有提前训练好，用于识别交通相关标志的 YOLOv5 模型。将我们摄像头得到的回传画面输入到 YOLOv5 模型内，模型会在回传画面中将交通相关标志给框选出来，并且标注其的置信度。

## 4.4 实验步骤

---

注意：

1、下面的步骤仅开启在回传画面中检测转向路标的效果，不会执行对应的动作，想直接体验无人驾驶玩法的用户可跳过本节课，前往与本节课同目录下的“[6.无人驾驶之综合应用](#)”查看。

2、输入指令时需要严格区分大小写和空格，且可使用“Tab”键补齐关键词。

---

1) 启动机器人，将其连接至远程控制软件 NoMachine。

2) 双击系统桌面的图标，打开命令行终端。

3) 输入指令“`sudo systemctl stop start_app_node.service`”，并按下回车，关闭手机 app 服务。

```
jetauto@jetauto-desktop:~$ sudo systemctl stop start_app_node.service
```

4) 输入指令“`roscd hiwonder_example/scripts/yolov5_detect/`”，进入程序目录。



```
jetauto@jetauto-desktop:~$ roscd jetauto_example/scripts/yolov5_detect
```

- 5) 输入指令“`python3 yolov5_trt.py`”，启动玩法程序。

```
jetauto@jetauto-desktop:~/jetauto_ws/src/jetauto_example/scripts/yolov5_detect$ python3 yolov5_trt_7_0.py
```

- 6) 放置路标于摄像头前，会识别出路标并标识于画面中，如需关闭此玩法，可在终端界面按下“`Ctrl+C`”。若关闭失败，请反复尝试。

- 7) 玩法体验完毕后，可通过指令或重启机器人来开启手机 APP 服务。

若未开启手机 APP 服务，则 APP 相关功能会失效。（若机器人重启，手机 APP 服务将会自动开启）输入指令“`sudo systemctl start start_app_node.service`”重启手机 APP 服务，等待蜂鸣器滴的一声即服务启动完成。

```
jetauto@jetauto-desktop:~$ sudo systemctl start start_app_node.service
```

注意：如果模型识别不到交通相关标志，我们则需要将置信度减小；如果模型误识别交通相关标志，我们则需要将置信度增大。

- 1) 输入指令“`cd ros_ws/src/hiwonder_example/scripts/self_driving`”，进入程序路径。

```
cd ros_ws/src/hiwonder_example/scripts/self_driving
```

- 2) 输入指令“`vim self_driving.launch`”，打开玩法程序。

```
vim self_driving.launch
```

红框中为置信度的值，可修改来调整目标检测的效果。

```
/usr/bin/zsh 80x24
1 <?xml version="1.0"?>
2 <launch>
3   <arg name="only_line_follow" default="false"/>
4   <include file="$(find hiwonder_peripherals)/launch/depth_cam.launch"/>
5   <!--底盘驱动(chassis driver)-->
6   <include file="$(find hiwonder_controller)/launch/hiwonder_controller.launch"/>
7
8   <arg name="node_name" default="yolov5" />
9   <rosparam param="$(arg node_name)/classes">['go', 'right', 'park', 'red', 'green', 'crosswalk']</rosparam>
10  <node unless="$(arg only_line_follow)" pkg="hiwonder_example" type="yolov5_node.py" name="$(arg node_name)" output="screen">
11    <param name="use_depth_cam" value="true"/>
12    <param name="engine" value="traffic_signs_640s_7_0.engine"/>
13    <param name="lib" value="libmyplugins_640.so"/>
14    <param name="conf_thresh" value="0.8"/>
15  </node>
```

## 4.5 功能实现

---

注意：机器人放置在地图中后，需要调整相机的角度，确保回传画面中不会出现机器人的机体部分，否则可能会影响玩法实现。

---

机器人在识别到右转交通标识牌后会进行右转。



## 4.6 参数调节说明

我们再进行无人驾驶玩法时，若小车无法识别到右转交通标识牌，则需要前往 [2.无人驾驶之路标检测](#)”学习。

该程序的源代码位于：

`/home/jetauto_ws/src/jetauto_example/scripts/self_driving/self_driving.py`

```

4 import cv2
5 import math
6 import yaml
7 import rospy
8 import threading
9 import numpy as np
10 import jetauto_sdk.pid as pid
11 import jetauto_sdk.misc as misc
12 from sensor_msgs.msg import Image
13 from geometry_msgs.msg as geo_msg
14 from jetauto_interfaces.msg import ObjectsInfo
15 from hiwonder_servo_msgs.msg import MultiRawIdPosDur
16 from hiwonder_servo_controllers.bus_servo_control import set_servos
17
18 def get_yaml_data(yaml_file):
19     yaml_file = open(yaml_file, 'r', encoding='utf-8')
20     file_data = yaml_file.read()
21     yaml_file.close()
22
23     data = yaml.load(file_data, Loader=yaml.FullLoader)
24
25     return data
26
27 lab_data = get_yaml_data("/home/jetauto/jetauto_software/lab_tool/lab_config.yaml")
28
29 class LineFollower:
30     def __init__(self, color):
31         self.target_color = color
32         self.rois = ((450, 480, 0, 320, 0.7), (390, 420, 0, 320, 0.3), (330, 360, 0, 320, 0.1))
33         self.weight_sum = 1.0
34
35     @staticmethod
36     def get_area_max_contour(contours, threshold=100):
37         """
38         获取最大面积对应的轮廓
39         :param contours:
40         :param threshold:
41         :return:
42         """
43         contour_area = zip(contours, tuple(map(lambda c: math.fabs(cv2.contourArea(c)), contours)))
44         contour_area = tuple(filter(lambda c_a: c_a[1] > threshold, contour_area))
45         if len(contour_area) > 0:

```

主函数内，若识别到右转交通标识牌，其面积大于 1500，则启动右转。

```

self.park = True
if 1500 < self.right_area:
    self.turn_right = True
if 110 < self.min_distance < 150:

```

若启动右转，则从巡线类的“get\_max\_y”函数，得到巡线需要的 y 值，重设 roi，并对其区域画线。

```

if self.turn_right:
    y = self.follower.get_max_y(binary_image)
    roi = [(0, y), (640, y), (640, 0), (0, 0)]
    cv2.fillPoly(binary_image, [np.array(roi)], [0, 0, 0])
    min_x = cv2.minMaxLoc(binary_image)[-1][0]
    cv2.line(binary_image, (min_x, y), (640, y), (255, 255, 255), 10)
elif self.park:

```

若 self.turn\_right 参数为 true，则开始左转，反之，接着巡线。

```

result_image, angle, x = self.follower(binary_image, self.image.copy())
if x != -1 and not self.stop:
    if x > 200:
        if self.turn_right:
            self.start_turn = True
            twist.angular.z = -0.45
        else:
            if self.start_turn:
                self.count_turn += 1
            else:
                self.count_turn = 0
            if self.count_turn > 6:
                self.count_turn = 0
                self.start_turn = False
                self.turn_right = False
            self.pid.SetPoint = 80
            self.pid.update(x)
            twist.angular.z = misc.set_range(self.pid.output, -0.8, 0.8)
        self.mecanum_pub.publish(twist)
    else:
        self.pid.clear()

```

若想要在识别到右转交通标识后，添加或修改功能可以在下列代码内修改或添加。

```

if x > 200:
    if self.turn_right:
        self.start_turn = True
        twist.angular.z = -0.45

```

## 5.无人驾驶之自主泊车

本节课是通过指令来实现检测识别泊车路标的。

### 5.1 准备工作

**注意：**体验此玩法时应保证在光照均匀的室内环境进行，并且避免灯以直射的方式照射地图，防止出现误识别的情况发生。

1) 开始前需要铺设好地图，确保地图平整无褶皱，道路畅通无障碍物，具体铺设地图可以前往与本节同目录下的“地图铺设及道具安装”查看学习。

2) 开始前需要铺设好地图，确保地图平整无褶皱，道路畅通无障碍物，具体铺设地图可以前往上级目录“2.AI 视觉无人驾驶玩法\1.无人驾驶道具摆放及调试视频”查看学习。

3) 本节的红绿灯模型为 yolov5 训练的模型，相关的 yolov5 内容可以前往“第 11 章 ROS+机器学习课程”查看了解。

4) 建议摆放在道路的正中间，方便识别！

### 5.2 实现原理

首先，获取摄像头的实时画面，对图像进行腐蚀、膨胀等操作。

然后，通过 YOLOv5 调用模型，与目标画面图像进行比较。

最后，根据比较结果，识别到泊车路标，让小车自动停入车位。

该程序的源代码位于：

`/home/Jetauto_ws/src/Jetauto_example/scripts/yolov5_detect/yolov5_trt.py`

```
4 import cv2
5 import math
6 import yaml
7 import rospy
8 import threading
9 import numpy as np
10 import jetauto_sdk.pid as pid
11 import jetauto_sdk.misc as misc
12 from sensor_msgs.msg import Image
13 import geometry_msgs.msg as geo_msg
14 from jetauto_interfaces.msg import ObjectsInfo
15 from hiwonder_servo_msgs.msg import MultiRawIdPosDur
16 from hiwonder_servo_controllers.bus_servo_control import set_servos
17
18 def get_yaml_data(yaml_file):
19     yaml_file = open(yaml_file, 'r', encoding='utf-8')
20     file_data = yaml_file.read()
21     yaml_file.close()
22
23     data = yaml.load(file_data, Loader=yaml.FullLoader)
24
25     return data
26
27 lab_data = get_yaml_data("/home/jetauto/jetauto_software/lab_tool/lab_config.yaml")
28
29 class LineFollower:
30     def __init__(self, color):
31         self.target_color = color
32         self.rois = ((450, 480, 0, 320, 0.7), (390, 420, 0, 320, 0.3), (330, 360, 0, 320, 0.1))
33         self.weight_sum = 1.0
34
35     @staticmethod
36     def get_area_max_contour(contours, threshold=100):
37         """
38         获取最大面积对应的轮廓
39         :param contours:
40         :param threshold:
41         :return:
42         """
43         contour_area = zip(contours, tuple(map(lambda c: math.fabs(cv2.contourArea(c)), contours)))
44         contour_area = tuple(filter(lambda c_a: c_a[1] > threshold, contour_area))
45         if len(contour_area) > 0:
```

## 5.3 实现流程

- 1、我们需要制作好我们需要使用的数据集，需要收集直行、右转、停车、红色红绿灯、绿色红绿灯、斑马线。
- 2、使用 `labelimg` 对图像进行标定，框出图像中我们需要训练的内容。
- 3、将数据的格式修改后，对数据集进行切分。
- 4、将数据集输入 YOLOv5 内进行训练，得到可以识别交通表示相关标志的模型。

得到的新模型就可以对我们训练的内容进行识别，机器人会做出相应的反馈动作。若想深入了解可以参考路径“第 10 章 ROS+机器学习课程\机器学习实战应用”进行学习。



这里我们会使用机器人内有提前训练好，用于识别交通相关标志的 YOLOv5 模型。将我们摄像头得到的回传画面输入到 YOLOv5 模型内，模型会在回传画面中将交通相关标志给框选出来，并且标注其置信度。

## 5.4 实验步骤

---

注意：

1、下面的步骤仅开启在回传画面中检测转向路标的效果，不会执行对应的动作，想直接体验无人驾驶玩法的用户可跳过本节课，前往与本节课同目录下的“[6.无人驾驶之综合应用](#)”查看。

---

2、输入指令时需要严格区分大小写和空格，且可使用“Tab”键补齐关键词。

---

1) 启动机器人，将其连接至远程控制软件 NoMachine。

2) 双击系统桌面的图标，打开命令行终端。

3) 输入指令“**sudo systemctl stop start\_app\_node.service**”，并按下回车，关闭手机 app 服务。

```
jetauto@jetauto-desktop:~$ sudo systemctl stop start_app_node.service
```

4) 输入指令“**roscd hiwonder\_example/scripts/yolov5\_detect/**”，进入程序目录。

```
jetauto@jetauto-desktop:~$ roscd jetauto_example/scripts/yolov5_detect
```

5) 输入指令“**python3 yolov5\_trt.py**”，启动玩法程序。

```
jetauto@jetauto-desktop:~/jetauto_ws/src/jetauto_example/scripts/yolov5_detect$  
python3 yolov5_trt_7_0.py
```

6) 放置路标于摄像头前，会识别出路标并标识于画面中，如需关闭此玩法，可在终端界面按下“**Ctrl+C**”。若关闭失败，请反复尝试。

7) 玩法体验完毕后，可通过指令或重启机器人来开启手机 APP 服务。

若未开启手机 APP 服务，则 APP 相关功能会失效。（若机器人重启，手机 APP 服务将会自动开启）输入指令“**sudo systemctl start start\_app\_node.service**”重启手机 APP 服务，等

待蜂鸣器滴的一声即服务启动完成。

```
jetauto@jetauto-desktop:~$ sudo systemctl start start_app_node.service
```

注意：如果模型识别不到交通相关标志，我们则需要将置信度减小；如果模型误识别交通相关标志，我们则需要将置信度增大。

- 1) 输入指令 “`cd ros_ws/src/hiwonder_example/scripts/self_driving`”，进入程序路径。

```
cd ros_ws/src/hiwonder_example/scripts/self_driving
```

- 2) 输入指令 “`vim self_driving.launch`”，打开玩法程序。

```
vim self_driving.launch
```

红框中为置信度的值，可修改来调整目标检测的效果。

```
1 <?xml version="1.0"?>
2 <launch>
3   <arg name="only_line_follow" default="false"/>
4   <include file="$(find hiwonder_peripherals)/launch/depth_cam.launch"/>
5   <!--底盘驱动(chassis driver)-->
6   <include file="$(find hiwonder_controller)/launch/hiwonder_controller.launch"/>
7
8   <arg name="node_name" default="yolov5" />
9   <rosparam param="/$(arg node_name)/classes">['go', 'right', 'park', 'red', 'green', 'crosswalk']</rosparam>
10  <node unless="$(arg only_line_follow)" pkg="hiwonder_example" type="yolov5_node.py" name="$(arg node_name)" output="screen">
11    <param name="use_depth_cam" value="true"/>
12    <param name="engine" value="traffic_signs_640s_7_0.engine"/>
13    <param name="lib" value="libmyplugins_640.so"/>
14    <param name="conf_thresh" value="0.8"/>
15  </node>
```

```
jetauto@jetauto-desktop:~$ sudo systemctl start start_app_node.service
```

## 5.5 功能实现

注意：机器人放置在地图中后，需要调整相机的角度，确保回传画面中不会出现机器人的机体部分，否则可能会影响玩法实现。

启动玩法后，将机器人放置在地图道路中，当机器人前进到泊车路标处，机器人会通过 yolov5 模型识别交通标识牌，根据路标指示，自动停入指定车位。



## 5.6 参数调节说明

我们再进行无人驾驶玩法时，若小车无法识别到泊车交通标识牌，则需要前往“[2.无人驾驶之路标检测](#)”学习。

该程序的源代码位于：

`/home/jetauto_ws/src/jetauto_example/scripts/self_driving/self_driving.py`

```
4 import cv2
5 import math
6 import yaml
7 import rospy
8 import threading
9 import numpy as np
10 import jetauto_sdk.pid as pid
11 import jetauto_sdk.misc as misc
12 from sensor_msgs.msg import Image
13 import geometry_msgs.msg as geo_msg
14 from jetauto_interfaces.msg import ObjectsInfo
15 from hiwonder_servo_msgs.msg import MultiRawIdPosDur
16 from hiwonder_servo_controllers.bus_servo_control import set_servos
17
18 def get_yaml_data(yaml_file):
19     yaml_file = open(yaml_file, 'r', encoding='utf-8')
20     file_data = yaml_file.read()
21     yaml_file.close()
22
23     data = yaml.load(file_data, Loader=yaml.FullLoader)
24
25     return data
26
27 lab_data = get_yaml_data("/home/jetauto/jetauto_software/lab_tool/lab_config.yaml")
28
29 class LineFollower:
30     def __init__(self, color):
31         self.target_color = color
32         self.rois = ((450, 480, 0, 320, 0.7), (390, 420, 0, 320, 0.3), (330, 360, 0, 320, 0.1))
33         self.weight_sum = 1.0
34
35     @staticmethod
36     def get_area_max_contour(contours, threshold=100):
37         """
38         获取最大面积对应的轮廓
39         :param contours:
40         :param threshold:
41         :return:
```



停车时，运行“**park\_action**”函数内容。初始化 twist，将线速度设置为 0.15m/s，发布 twist 并运行 (0.25/0.15) S，运行完毕后，重复初始化并将角速度设置成-0.15 米每秒，发布 twist 并运行 (0.25/0.15) S。若想添加或修改识别到泊车交通标识的效果，可在“**park\_action**”函数添加或修改代码。

```
def park_action(self):  
    twist = geo_msg.Twist()  
    twist.linear.x = 0.15  
    self.mecanum_pub.publish(twist)  
    rospy.sleep(0.25/0.15)  
    twist = geo_msg.Twist()  
    twist.linear.y = -0.15  
    self.mecanum_pub.publish(twist)  
    rospy.sleep(0.25/0.15)  
    self.mecanum_pub.publish(geo_msg.Twist())
```

在主函数内，若检测到停车标识则启动停车线程，当停车标识到一定大小的时候，开始运行“**park\_action**”函数的内容。

```
def image_proc(self):  
    if self.image is not None:  
        while self.is_running:  
            twist = geo_msg.Twist()  
            binary_image = self.follower.get_binary(self.image)  
            if 600 < self.park_pos[0] and 150 < self.park_pos[1] and not self.start_park:  
                self.mecanum_pub.publish(geo_msg.Twist())  
                self.start_park = True  
                threading.Thread(target=self.park_action).start()  
                self.stop = True  
            if 0 < self.park_pos[0]:
```

## 6.无人驾驶之综合应用

本节课是通过指令来实现机器人对无人驾驶玩法的综合应用，即包括车道保持、路标检测、红绿灯识别、转向决策和自动泊车。

### 6.1 准备工作

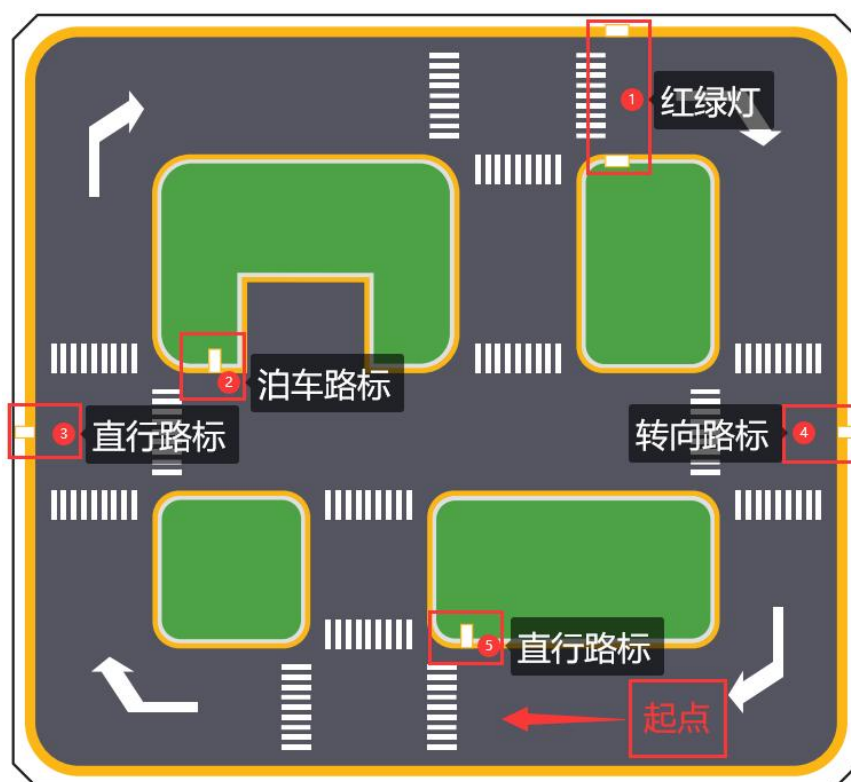
---

**注意：**体验此玩法时应保证在光照均匀的室内环境进行，并且避免灯以直射的方式照射地图，防止出现误识别的情况发生。

---

- 1) 开始前需要铺设好地图，确保地图平整无褶皱，道路畅通无障碍物，具体铺设地图可以前往上级目录“**2.AI 视觉无人驾驶玩法\1.无人驾驶道具摆放及调试视频**”查看学习。
- 2) 建议摆放在道路的正中间，方便识别！

地图需要铺设在平整的表面，并确保地图平整无褶皱，道路畅通无障碍物，所有路标和红绿灯需要摆放在地图指定位置，并面朝地图顺时针方向，路标位置和起点位置如下图：




## 6.2 调节摄像头位置并设置颜色阈值和置信度

---

注意：输入指令时需要严格区分大小写和空格，且可使用“Tab”键补齐关键词。

---

- 1) 启动 JetAuto，将其连接至远程控制软件 NoMachine。

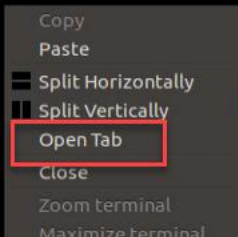
- 2) 点击系统桌面的左侧图标，打开命令行终端。

- 3) 输入指令“**sudo systemctl stop start\_app\_node.service**”，并按下回车，关闭手机 app 服务。

```
jetauto@jetauto-desktop:~$ sudo systemctl stop start_app_node.service
```

- 4) 额外创建三个命令栏标签，右键命令栏，并选择“**Open Tab**”。需要重复这个操作三次。

```
lidar: A1
camera: AstraProPlus
machine: JetAuto
host: jetauto_1
master: jetauto_1
ros_hostname: 192.168.149.1
ros_master_uri: http://192.168.149.1:11311
jetauto@jetauto-desktop:~$ sudo systemctl stop start_app_node.service
jetauto@jetauto-desktop:~$
```



5) 在第一个命令标签输入命令：“**roslaunch jetauto\_peripherals astrapro.launch**”，启动深度相机初始化节点。

```
jetauto@jetauto-desktop:~$ roslaunch jetauto_peripherals astrapro.launch
```

6) 在第二个命令标签输入命令：“**roslaunch jetauto\_controller jetauto\_controller.launch**”，启动底盘控制节点。

```
jetauto@jetauto-desktop:~$ roslaunch jetauto_controller jetauto_controller.launch
```

7) 在第三个命令标签输入命令：

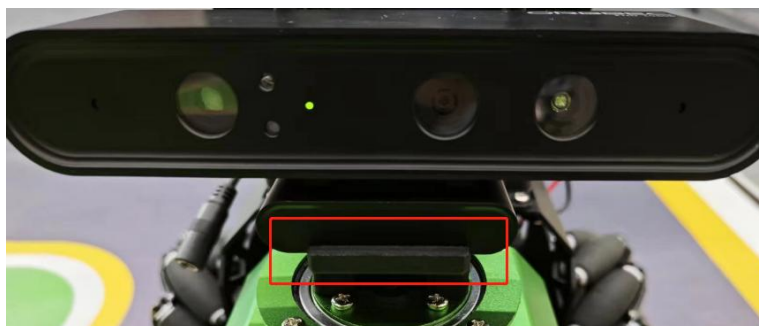
“**roslaunch jetauto\_peripherals teleop\_key\_control.launch robot\_name:="/**”，启动键盘控制节点。

```
jetauto@jetauto-desktop:~$ roslaunch jetauto_peripherals teleop_key_control.launch robot_name:= "/"
```

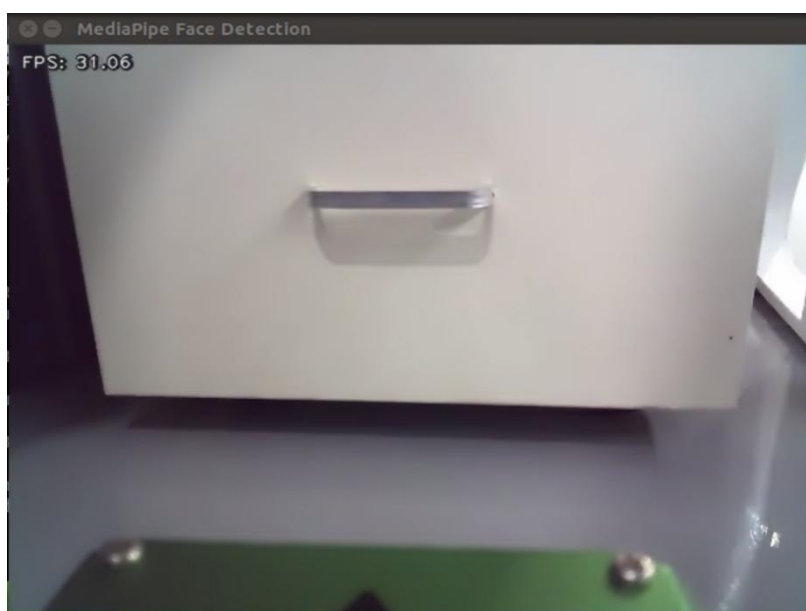
8) 在第四个命令标签输入命令：“**cd jetauto\_software/lab\_tool/ && python3 main.py**”，启动颜色阈值调节工具。

```
jetauto@jetauto-desktop:~$ cd jetauto_software/lab_tool/ && python3 main.py
```

9) 将深度相机扣在相机支架上。



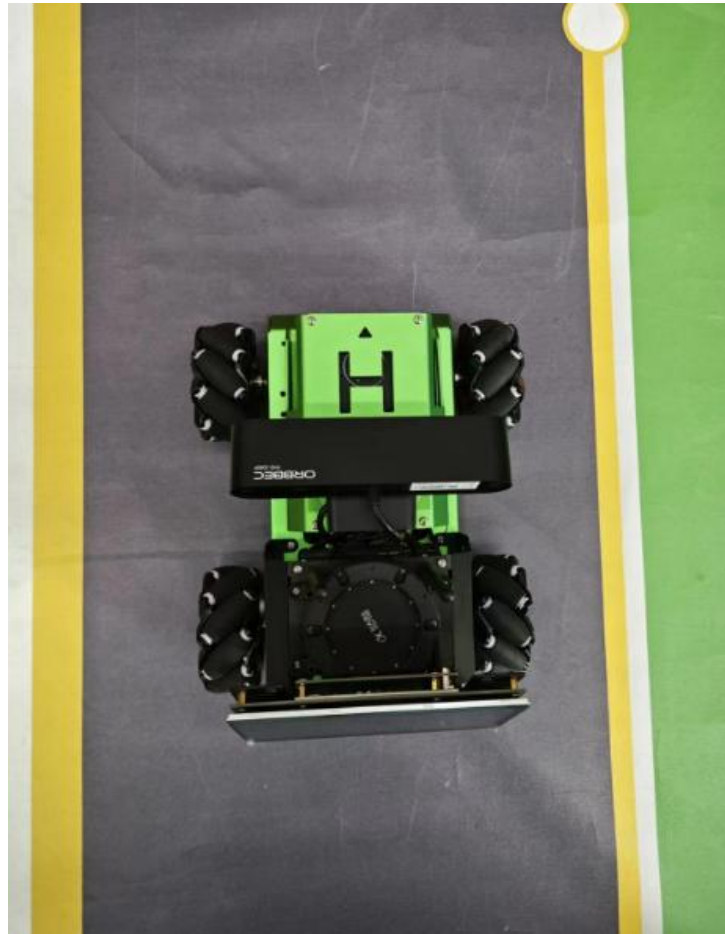
10) 扭动深度相机的关节，调节深度相机的角度，使相机照到机器人机身。



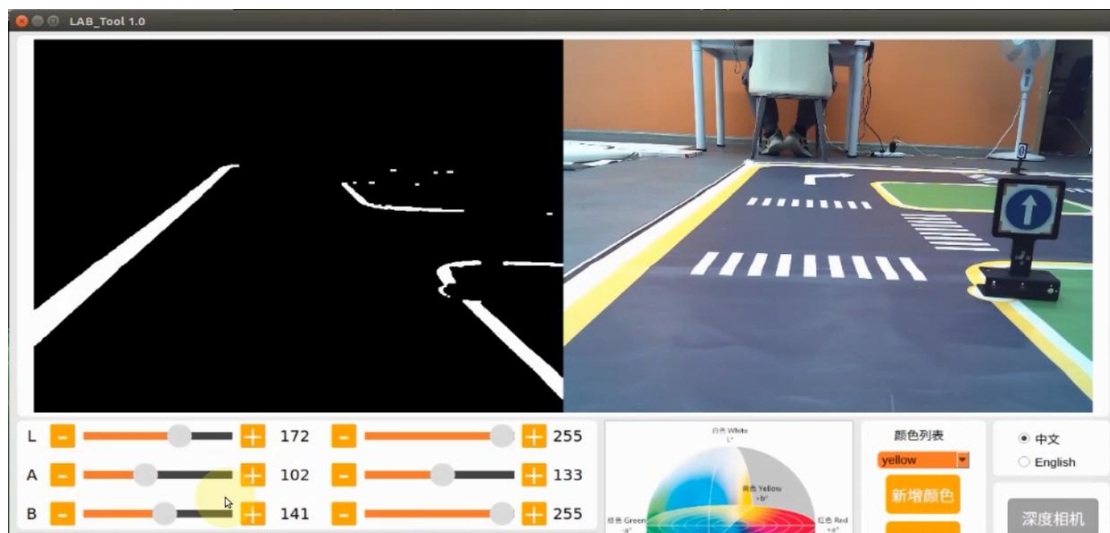
11) 将深度相机回调一点，使深度相机刚好照不到机器人机身。



12) 将机器人放置在地图上，使其在地图中道路上居中。



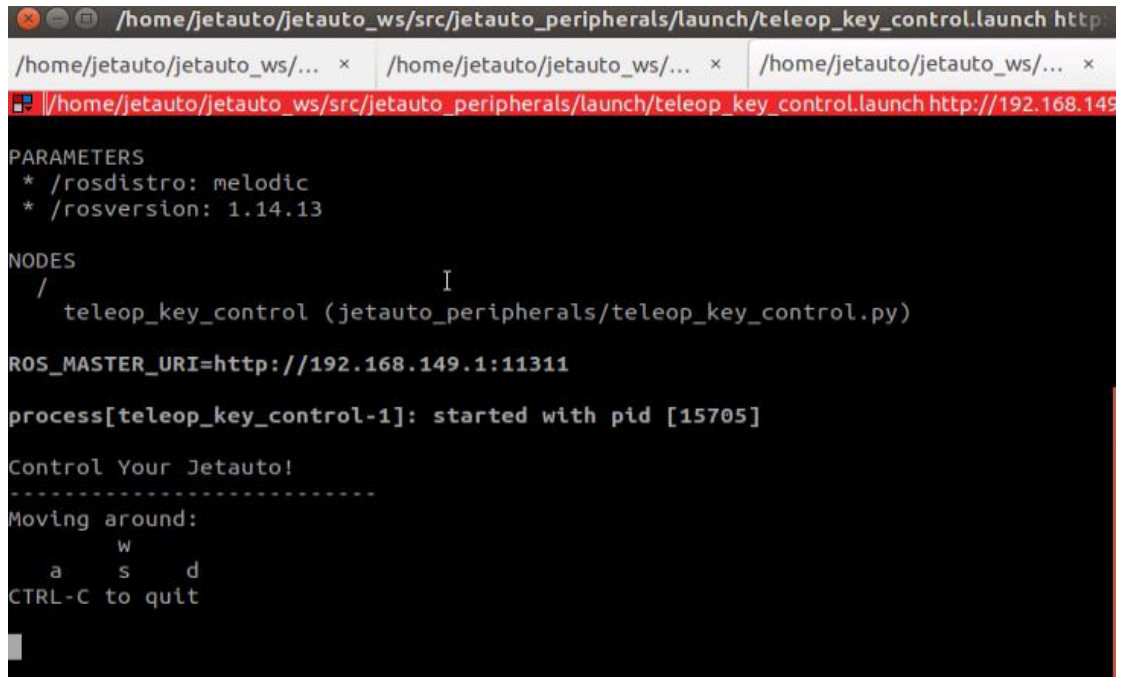
13) 开始调节颜色阈值，将颜色列表内的颜色切换成“yellow”，然后根据“第 9 章 ROS+OpenCV 视觉识别标准课程第 1 课 颜色阈值的调节”调节阈值。



14) 调节完成后，将机器人的窗口移动到如下图一样的位置，然后选择第三个命令栏标签，控制小车在地图上运动。观察小车能否识别到地图上的黄线，若可以则不需要调节，若无法识别某处的黄线，或者识别除黄线外的其他物体，则需要调节颜色阈值。直到，地图上



的黄线能够完全识别且不受其他物体影响。



```
/home/jetauto/jetauto_ws/src/jetauto_peripherals/launch/teleop_key_control.launch http://192.168.149.1:11311
PARAMETERS
* /rostdistro: melodic
* /rosversion: 1.14.13

NODES
/
  teleop_key_control (jetauto_peripherals/teleop_key_control.py)

ROS_MASTER_URI=http://192.168.149.1:11311

process[teleop_key_control-1]: started with pid [15705]

Control Your Jetauto!
-----
Moving around:
    W
  A   S   D
CTRL-C to quit
```

注意：如果模型识别不到交通相关标志，我们则需要将置信度减小；如果模型误识别交通相关标志，我们则需要将置信度增大。

- 1) 输入指令 “cd ros\_ws/src/hiwonder\_example/scripts/self\_driving”，进入程序路径。



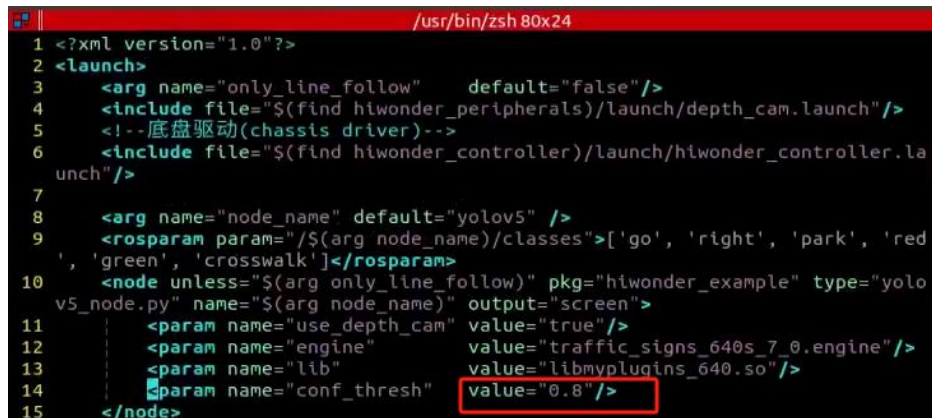
```
cd ros_ws/src/hiwonder_example/scripts/self_driving
```

- 2) 输入指令 “vim self\_driving.launch”，打开玩法程序。



```
vim self_driving.launch
```

红框中为置信度的值，可修改来调整目标检测的效果。



```
1 <?xml version="1.0"?>
2 <launch>
3   <arg name="only_line_follow" default="false"/>
4   <include file="$(find hiwonder_peripherals)/launch/depth_cam.launch"/>
5   <!--底盘驱动(chassis driver)-->
6   <include file="$(find hiwonder_controller)/launch/hiwonder_controller.launch"/>
7
8   <arg name="node_name" default="yolov5" />
9   <rosparam param="$(arg node_name)/classes">['go', 'right', 'park', 'red', 'green', 'crosswalk']</rosparam>
10  <node unless="$(arg only_line_follow)" pkg="hiwonder_example" type="yolo_v5_node.py" name="$(arg node_name)" output="screen">
11    <param name="use_depth_cam" value="true"/>
12    <param name="engine" value="traffic_signs_640s_7_0.engine"/>
13    <param name="lib" value="libmyplugins_640.so"/>
14    <param name="conf_thresh" value="0.8"/>
15  </node>
```



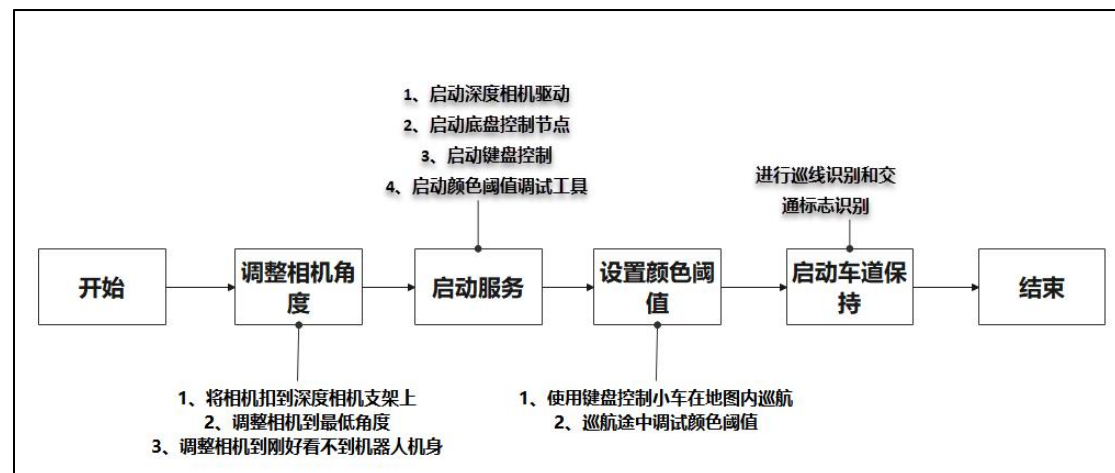
## 6.3 实现流程

首先，载入 yolov5 训练好的模型文件和所需的库文件，并获取摄像头的实时画面，对图像进行腐蚀、膨胀等操作。

然后，识别图像中目标颜色线段，获取目标图像的大小、中心点等信息。并通过 Yolov5 调用模型，与目标画面图像进行比较。

最后，根据目标图像中心点的偏移量比较，调整前进的方向，让机器人保持在道路中间。同时在地图的前进过程中，根据识别的不同路标信息，执行对应的动作。

实现流程图如下：



该程序的源代码位于：

`/home/Jetauto_ws/src/Jetauto_example/scripts/yolov5_detect/yolov5_trt.py`

```

4 import cv2
5 import math
6 import yaml
7 import rosp
8 import threading
9 import numpy as np
10 import jetauto_sdk.pid as pid
11 import jetauto_sdk.misc as misc
12 from sensor_msgs.msg import Image
13 import geometry_msgs.msg as geo_msg
14 from jetauto_interfaces.msg import ObjectsInfo
15 from hiwonder_servo_msgs.msg import MultiRawIdPosDur
16 from hiwonder_servo_controllers.bus_servo_control import set_servos
17
18 def get_yaml_data(yaml_file):
19     yaml_file = open(yaml_file, 'r', encoding='utf-8')
20     file_data = yaml_file.read()
21     yaml_file.close()
22
23     data = yaml.load(file_data, Loader=yaml.FullLoader)
24
25     return data
26
27 lab_data = get_yaml_data("/home/jetauto/jetauto_software/lab_tool/lab_config.yaml")
28
29 class LineFollower:
30     def __init__(self, color):
31         self.target_color = color
32         self.rois = ((450, 480, 0, 320, 0.7), (390, 420, 0, 320, 0.3), (330, 360, 0, 320, 0.1))
33         self.weight_sum = 1.0
34
35     @staticmethod
36     def get_area_max_contour(contours, threshold=100):
37         """
38         获取最大面积对应的轮廓
39         :param contours:
40         :param threshold:
41         :return:
42         """
43         contour_area = zip(contours, tuple(map(lambda c: math.fabs(cv2.contourArea(c)), contours)))
44         contour_area = tuple(filter(lambda c_a: c_a[1] > threshold, contour_area))
45         if len(contour_area) > 0:

```

## 6.4 实验步骤

注意：输入指令时需要严格区分大小写和空格，且可使用“Tab”键补齐关键词。

1) 启动机器人，使用远程控制软件 NoMachine 连接机器人。

2) 双击系统桌面的图标 ，打开命令行终端。

3) 输入指令“`sudo systemctl stop start_app_node.service`”，并按下回车，关闭手机 APP 服务。

```
jetauto@jetauto-desktop:~$ sudo systemctl stop start_app_node.service
```

4) 输入指令“`roslaunch jetauto_example self_driving.launch`”，启动无人驾驶服务。

```
jetauto@jetauto-desktop:~$ roslaunch jetauto_example self_driving.launch
```

5) 等待服务启动完毕，当出现下图信息时，说明已经启动完成。

```
[03/28/2023-23:23:21] [TRT] [I] [MemUsageChange] Init CUDA: CPU +224, GPU +0, now: CPU 269, GPU 3426 (MiB)
[03/28/2023-23:23:21] [TRT] [I] Loaded engine size: 22 MiB
[03/28/2023-23:23:22] [TRT] [I] [MemUsageChange] Init cuBLAS/cuBLASLt: CPU +158, GPU +101, now: CPU 456, GPU 3580 (MiB)
[03/28/2023-23:23:26] [TRT] [I] [MemUsageChange] Init cuDNN: CPU +241, GPU +287, now: CPU 697, GPU 3867 (MiB)
[03/28/2023-23:23:26] [TRT] [I] [MemUsageChange] TensorRT-managed allocation in engine deserialization: CPU +0, GPU +21, now: CPU 0, GPU 21 (MiB)
[03/28/2023-23:23:26] [TRT] [I] [MemUsageChange] Init cuBLAS/cuBLASLt: CPU +0, GPU +0, now: CPU 674, GPU 3845 (MiB)
[03/28/2023-23:23:26] [TRT] [I] [MemUsageChange] Init cuDNN: CPU +0, GPU -1, now: CPU 674, GPU 3844 (MiB)
[03/28/2023-23:23:26] [TRT] [I] [MemUsageChange] TensorRT-managed allocation in IExecutionContext creation: CPU +0, GPU +39, now: CPU 0, GPU 60 (MiB)
binding: data (3, 640, 640)
binding: prob (38001, 1, 1)
[ WARN] [1680017006.464306027]: sub num:2
[INFO] [1680017006.558791]: start yolov5 detect
```

6) 如需关闭此玩法，可在终端界面按下“**Ctrl+C**”。若关闭失败，请反复尝试。

7) 玩法体验完毕后，可通过指令或重启机器人来开启手机 APP 服务。

8) 若未开启手机 APP 服务，则 APP 相关功能会失效。（若机器人重启，手机 APP 服务将会自动开启）输入指令“**sudo systemctl start start\_app\_node.service**”重启手机 APP 服务，等待蜂鸣器滴的一声即服务启动完成。

```
jetauto@jetauto-desktop:~$ sudo systemctl start start_app_node.service
```

## 6.5 功能实现

### ◆ 车道保持

启动玩法后，小车会通过巡线，识别道路边缘的黄线。根据黄线笔直或弯曲的状态，执行前进和转弯的动作，保证自身在车道内。

### ◆ 红绿灯识别

当小车遇到红绿灯时，若为红灯则停在原地，若为绿灯则前进。在前进通过斑马线时，当小车检测到斑马线，会自动放慢速度缓慢前进。

### ◆ 转向路标和泊车路标

当在小车前进的过程中检测到前方的交通标识牌，则会根据交通标识牌做出相应的动作。若为右转，则小车右转并前进，若为泊车，则会进行侧方停车。

## 6.6 Launch 文件说明

在我们前面的 1-5 课对我们小车在地图上行驶的各个功能进行分部说明，这里我们可以

通过 Launch 文件来对各个部分如何组合进行理解。

```
1 <?xml version="1.0"?>
2 <launch>
3   <arg name="only_line_follow" default="false"/>
4   <include file="$(find jetauto_peripherals)/launch/astapro.launch"/>
5   <include file="$(find hiwonder_servo_controllers)/launch/start.launch"/>
6   <include file="$(find jetauto_controller)/launch/jetauto_controller.launch"/>
7
8   <arg name="node_name" default="yolov5_node" />
9   <rosparam param="/$(arg node_name)/classes">['go', 'right', 'park', 'red', 'green', 'crosswalk']</rosparam>
10  <node unless="$(arg only_line_follow)" pkg="jetauto_example" type="yolov5_node.py" name="$(arg node_name)" output="screen">
11    <param name="use_depth_cam" value="true"/>
12    <param name="engine" value="traffic_signs_640s_6_2.engine"/>
13    <param name="lib" value="libmyplugins_640.so"/>
14    <param name="conf_thresh" value="0.8"/>
15  </node>
16
17  <node pkg="jetauto_example" type="self_driving.py" name="self_driving" output="screen">
18    <param name="only_line_follow" value="$(arg only_line_follow)"/>
19  </node>
20 </launch>
```

### 6.6.1 启动设备

```
3   <arg name="only_line_follow" default="false"/>
4   <include file="$(find jetauto_peripherals)/launch/astapro.launch"/>
5   <include file="$(find hiwonder_servo_controllers)/launch/start.launch"/>
6   <include file="$(find jetauto_controller)/launch/jetauto_controller.launch"/>
```

参数“**only\_line\_follow**”其作用为控制小车是否只保留巡线功能，其的默认值定义为“**false**”即不但可以进行巡线还可以进行交通标识的识别。

文件“**astapro.launch**”深度相机服务，在进行玩法前开启。

文件“**start.launch**”舵机控制服务，在进行玩法前开启。

文件“**jetauto\_controller.launch**”底盘控制服务，在进行玩法前开启。

### 6.1.2 启动 Yolov5 节点

```
8   <arg name="node_name" default="yolov5_node" />
9   <rosparam param="/$(arg node_name)/classes">['go', 'right', 'park', 'red', 'green', 'crosswalk']</rosparam>
10  <node unless="$(arg only_line_follow)" pkg="jetauto_example" type="yolov5_node.py" name="$(arg node_name)" output="screen">
11    <param name="use_depth_cam" value="true"/>
12    <param name="engine" value="traffic_signs_640s_6_2.engine"/>
13    <param name="lib" value="libmyplugins_640.so"/>
14    <param name="conf_thresh" value="0.8"/>
15  </node>
```

参数“**node\_name**”，其默认设置成“**yolov5\_node**”，设置 yolov5 检测节点。

参数“**param="/\$(arg node\_name)/classes"**”为 yolov5 模型的类别文件，['go', 'right', 'park', 'red', 'green', 'crosswalk']为其能识别的类别。

第 10 行设置 yolov5 节点，其启动“**yolov5\_node.py**”文件并对内部参数进行设置：

参数“**use\_depth\_cam**”，是否使用深度相机，默认使用深度相机。

参数“**engine**”，使用的模型路径，默认使用“**traffic\_signs\_640s\_6\_2.engine**”。

参数“**lib**”，使用的模型编译文件，默认使用“**libmyplugins\_640.so**”。

参数“**conf\_thresh**”，使用识别结果进行反馈的阈值，默认设置为“**0.8**”。

### 6.1.3 启动无人驾驶

```
17 <node pkg="jetauto_example" type="self_driving.py" name="self_driving" output="screen">
18   <param name="only_line_follow" value="$(arg only_line_follow)"/>
19 </node>
20 </launch>
```

第 17 行使用"self\_driving.py"启动无人驾驶玩法节点，其的参数"only\_line\_follow"，会接入最开始设置好的值，默认为“false”。

## 无人驾驶玩法常见问题

1.机器人在巡线行驶时效果不佳，走起来歪歪扭扭的。

根据实际场景的光线环境进行颜色阈值调节，具体的颜色阈值的调节可以参考“[13 ROS+OpenCV 课程](#)”进行参考学习。

2.机器人在转弯时，转弯半径过大或过小。

1)检查相机角度是否调整正确，具体的调节可以参考“[1.4 调节摄像头位置并设置颜色阈值](#)”进行参考学习。

2)修改巡线处理代码.

输入指令“cd /home/Jetauto\_ws/src/Jetauto\_example/scripts/self\_driving”，进入玩法程序路径。

```
cd ros_ws/src/hiwonder_example/scripts/self_driving
```

输入指令“vim self\_driving.py”，打开玩法程序。

```
vim self_driving.py
```

红框中为车道中心点，可修改来调整转弯效果，将值调小，机器人会提前转弯，将值调大，机器人会延后转弯。

```

236         if x != -1 and not self.stop:
237             if x > 200:
238                 if self.turn_right:
239                     self.start_turn = True
240                     twist.angular.z = -0.45
241             else:
242                 if self.start_turn:
243                     self.count_turn += 1
244                 else:
245                     self.count_turn = 0
246                 if self.count_turn > 6:
247                     self.count_turn = 0

```

3.泊车位置不理想。

可以修改泊车处理函数或者修改泊车操作开始的位置，具体的操作可以参考“[5.5 功能实现](#)”进行参考学习。

4. 交通标志识别不准确。

修改目标检测置信度，具体的操作可以参考“[6.2 调节摄像头位置并设置颜色阈值和置信度](#)”进行参考学习。