Part 1 Lane Keeping

NOTE：
1) Because different lighting conditions can affect the accuracy of color recognition, please refer to "9. Standard ROS + OpenCV Vision Recognition Part/Part 1 Color Threshold Adjustment" to adjust the color channel accordingly.

2) Unfold and place the map on the ground ensuring that it is not folded.

1. Program Logic

Initially, the program will **access the live camera screen** to **apply image erosion and dilation** to **improve the image quality**.

Subsequently, the **Yolov5 model will be invoked** to **compare the captured image with the target image.**

Finally, the program will **adjust the movement direction of JetAuto** to **ensure that the robot stays in the middle of the lane.**

The source code of this program is stored in
/home/jetauto_ws/src/jetauto_example/scripts/self_driving/self_driving.py

```python
#!/usr/bin/env python3
# encoding: utf-8
# @data:2022/11/07
# @author:aiden
# 无人驾驶
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
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

# 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")


# 负责线路跟踪逻辑
class LineFollower:
    def __init__(self, color): # color 用于指定要跟随的线的颜色
        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

    @staticmethod # 标记下面的方法为静态方法，意味着这个方法不需要访问类的实例即可调
用。


    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)))
        # tuple(map(lambda c: math.fabs(cv2.contourArea(c)：生成一个包含每个轮廓
的面积的元组
        # zip(contours, tuple()): 将原始轮廓列表和计算出的面积组合在一起，形成一个
包含 (轮廓, 面积) 对

        contour_area = tuple(filter(lambda c_a: c_a[1] > threshold, 
contour_area))
        # 筛选出面积大于 threshold 的轮廓


        # 找出面积最大的轮廓
        if len(contour_area) > 0:
            max_c_a = max(contour_area, key=lambda c_a: c_a[1])
            # key=lambda c_a: c_a[1] 指定 max() 函数使用 (轮廓, 面积) 对中的面积作
为比较的键值。

            return max_c_a # 将是面积最大的 (轮廓, 面积) 对
        return None
    # 用于获取图像中最大的 y 坐标
    def get_max_y(self, image):
        h, w = image.shape[:2] # 获取图像的高度 (h) 和宽度 (w)
        roi = image[:, int(w/2):w] # 定义感兴趣区域 (ROI) ,即图像的右半部分
        flip_binary = cv2.flip(roi, 0) # 进行垂直翻转 (参数 0 表示沿 x 轴翻转)
        max_y = cv2.minMaxLoc(flip_binary)[-1][1] # 用于找到图像 (或 ROI 区域) 中
的最小和最大灰度值及其位置。它返回四个值：最小值、最大值、最小值位置、最大值位置。
        # [-1][1] 取最大值的位置的 Y 坐标
        return h - max_y # 将坐标转换回原始图像的坐标系统


    # 用于获取图像中最大的 x 坐标
    def get_max_x(self, image):
        h, w = image.shape[:2] # 获取图像的高度 (h) 和宽度 (w)
        roi = image[int(h/2):h, 0:int(w/2)] # 定义感兴趣的区域, 即图像左下角的一半。
        #cv2.imshow('roi', roi)
        #rotate_binary = cv2.rotate(roi, cv2.ROTATE_90_COUNTERCLOCKWISE)
        flip_binary = cv2.flip(roi, 1) # 进行水平翻转
        #cv2.imshow('1', flip_binary)
        (x, y) = cv2.minMaxLoc(flip_binary)[-1] # [-1] 取最大值的位置的坐标 (X, Y)
        return 320 - x, y + 240 # 将坐标转换回原始图像的坐标系统

    # 将输入图像转换为二值图像
    def get_binary(self, image):
        img_lab = cv2.cvtColor(image, cv2.COLOR_RGB2LAB)  # rgb 转 lab
        img_blur = cv2.GaussianBlur(img_lab, (3, 3), 3)  # 高斯模糊去噪
        mask = cv2.inRange(img_blur,
tuple(lab_data['lab']['Stereo'][self.target_color]['min']),
tuple(lab_data['lab']['Stereo'][self.target_color]['max']))  # 二值化
        eroded = cv2.erode(mask, cv2.getStructuringElement(cv2.MORPH_RECT, (3,
3)))  # 腐蚀
        dilated = cv2.dilate(eroded, cv2.getStructuringElement(cv2.MORPH_RECT,
(3, 3)))  # 膨胀

        return dilated

    # 用于实际执行线跟随的逻辑
    def __call__(self, image, result_image): # image (原始图像) # result_image
 (用于显示结果的图像)
        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]]  # 从 image 中截取 roi 部分
            contours = cv2.findContours(blob, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_TC89_L1)[-2]  # 查找每个区域中的轮廓。
```

```
            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))  # 获取最小外接矩形的四个顶点
坐标
                for j in range(4):
                    box[j, 1] = box[j, 1] + roi[0] # 将最小外接矩形的坐标转换为原
始图像坐标系中的坐标
                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) # 将每个最小外接矩形的中心点 x 坐标
添加到列表 center_x 中。
            else:
                center_x.append(-1) # 如果未检测到有效的轮廓，则将 -1 添加到
center_x 中，以便后续处理。


        for i in range(len(center_x)): # 遍历保存每个 ROI 中心点横坐标的列表
center_x。
            if center_x[i] != -1: # 检查当前 ROI 的中心点是否有效
                if center_x[i] > max_center_x:
                    max_center_x = center_x[i] # 找到所有 ROI 中心点中的最大横坐
标值
                centroid_sum += center_x[i] * self.rois[i][-1] # 根据不同 ROI 的
权重，对每个 ROI 中心点的横坐标进行加权求和

        if centroid_sum == 0: # 表示未找到有效的中心点，可能是因为图像中没有检测到目
标。
            return result_image, None, max_center_x # 函数返回原始图像
result_image，中心点位置为 None，以及最大中心点横坐标 max_center_x。

        center_pos = centroid_sum / self.weight_sum  # 按比重，计算中心点位置 (x
坐标)
        angle = math.degrees(-math.atan((center_pos - (w / 2.0)) / (h / 2.0)))
# 计算偏转角度
```

```
        return result_image, angle, max_center_x

class LineFollowingNode:
    def __init__(self, name):
        rospy.init_node(name, anonymous=True)   # 初始化 ROS 节点
        self.name = name        # 节点名称
        self.is_running = True  # 标志位，用于控制程序的运行状态
        self.pid = pid.PID(0.01, 0.0, 0.0)   # 创建一个 PID 控制器对象
        self.center = None      # 用于存储目标位置的中心坐标
        self.stop = False       # 标志位，表示机器人是否需要停止。
        self.red = False        # 标志位，表示是否检测到红灯
        self.park = False       # 标志位，表示是否检测到停车标志
        self.green = False      # 标志位，表示是否检测到绿灯
        self.image = None       # 存储从摄像头接收到的图像
        self.min_distance = 0   # 存储距离值
        self.right_area = -1    # 存储右侧区域的面积
        self.park_pos = (0, 0)  # 存储停车位置
        self.count_red = 0      # 计数器，用于计算某些事件发生的次数
        self.count_green = 0
        self.count_turn = 0
        self.count_go = 0
        self.objects_info = []  # 存储目标检测结果
        self.colors = common.Colors() # 存储颜色信息
        self.start_turn = False # 标志位，表示转向是否开始
        self.start_park = False # 标志位，表示停车是否开始
        self.turn_right = False # 标志位，表示右转是否开始
        self.start_slow_down = False # 标志位，表示减速是否开始
        self.count_cross_walk = 0    # 计数器，用于计算某些事件发生的次数
        self.cross_walk_distance = 0.4     # 存储通过人行横道时的距离

        self.follower = LineFollower("yellow")  # 创建了一个 LineFollower 类的实
例，用于处理线路跟踪任务

        self.classes = rospy.get_param('/yolov5_node/classes') # 从 ROS 参数服务
器中获取目标类别信息。

        # 用于发布底盘控制
        self.mecanum_pub = rospy.Publisher('/jetauto_controller/cmd_vel',
geo_msg.Twist, queue_size=1)  # 底盘控制
        # 创建了一个用于向/jetauto_controller/cmd_vel 主题发送 Twist 消息的 ROS 发布
器，这些消息控制着机器人的运动。队列大小为 1 意味着在消息能够实际发送到网络上之前，可以
在队列中累积的消息的数量,意味着如果新的消息被发布但旧的消息还没来得及发送，旧的消息会被
丢弃。


        # 用于发布图像处理结果
        self.result_publisher = rospy.Publisher(self.name + '/image_result', 
Image, queue_size=1)  # 图像处理结果发布
        # 创建了一个用于向特定主题发布消息类型为 Image 的图像处理结果的 ROS 发布器, 让
其他的 ROS 节点可以订阅并接收到最新的图像数据


        # 用于发布舵机控制消息
        self.joints_pub = 
rospy.Publisher('servo_controllers/port_id_1/multi_id_pos_dur', 
MultiRawIdPosDur, queue_size=1)  # 舵机控制
        # 创建了一个用于向 servo_controllers/port_id_1/multi_id_pos_dur 主题发送
MultiRawIdPosDur 消息的 ROS 发布器, 这些消息控制机器人的舵机。
        # servo_controllers/port_id_1/multi_id_pos_dur: 该主题通常与特定的硬件控
制协议或接口配置相关, 用于机器人中多个舵机的同时控制。
        # MultiRawIdPosDur: 为了特定的应用定义的自定义消息类型, 包含了舵机的 ID、目
标位置和动作持续时间等信息

        # 从 ROS 的参数服务器中获取参数-深度摄像头的名称
        depth_camera = rospy.get_param('/depth_camera/camera_name', 
'camera')  # 获取参数

        # 创建了一个 ROS 订阅者, 用于接收来自深度摄像头的原始 RGB 图像数据
        rospy.Subscriber('/%s/rgb/image_raw'%depth_camera, Image, 
self.image_callback)  # 摄像头订阅
        # '/%s/rgb/image_raw' % depth_camera: 该主题通常用于传输原始的 RGB 图像数据
        # Image: 图像消息的类型, 代表订阅的数据格式
        # self.image_callback: 接收到图像数据时调用的函数

        # 创建了一个 ROS 订阅者, 用于接收 YOLOv5 目标检测节点发布的物体检测信息
        rospy.Subscriber('/yolov5/object_detect', ObjectsInfo, 
self.get_object_callback)
        # '/yolov5/object_detect': 这是订阅的 ROS 主题代表了 YOLOv5 目标检测节点发布
的物体检测信息
        # ObjectsInfo: 代表了订阅主题所传输的数据的类型, 用于表示物体检测信息
        # self.get_object_callback: 当订阅到新的物体检测信息时, 会调用
self.get_object_callback 函数。


        # 等待'/yolov5_node/start'参数被设置为真, 然后才继续执行后续的代码。
        while not rospy.is_shutdown():
            try:
                if rospy.get_param('/yolov5_node/start'):
                    break
            except:
                rospy.sleep(0.1)

        # 等待两个特定的 ROS 参数都被设置为真, 然后才继续执行后续的代码。
        while not rospy.is_shutdown():
            try:
                if rospy.get_param('/hiwonder_servo_manager/running') and 
rospy.get_param('/joint_states_publisher/running'):
                    break
            except:
                rospy.sleep(0.1)

        set_servos(self.joints_pub, 1000, ((5, 500), ))  # 舵机中位

        # 向 ROS 中的底盘控制话题发布一个空的运动控制命令，以停止机器人的运动。
        self.mecanum_pub.publish(geo_msg.Twist())

        # 调用实例的 image_proc 方法，进行图像处理和分析操作
        self.image_proc()


    # 将从 ROS 图像话题接收到的原始图像数据转换为 RGB 图像
    def image_callback(self, ros_image):

        # 将 ROS 图像消息中的原始数据转换为 NumPy 数组，并将其解释为 RGB 图像
        rgb_image = np.ndarray(shape=(ros_image.height, ros_image.width, 3), 
dtype=np.uint8, buffer=ros_image.data) # 原始 RGB 画面
        self.image = rgb_image


    # park_action 方法通过发布 Twist 消息来控制底盘进行一系列的线性运动，从而实现停车
动作。
    def park_action(self):
        twist = geo_msg.Twist() # 创建一个 Twist 消息对象
        twist.linear.x = 0.15 # 将 twist 对象的线速度设置为 0.15，表示向前移动
        self.mecanum_pub.publish(twist) # 发布 Twist 消息，控制底盘进行线性运动，使
车辆向前移动。
        rospy.sleep(0.25/0.15) # 使程序暂停一段时间，确保车辆移动到目标位置。
        self.mecanum_pub.publish(geo_msg.Twist()) # 发布空的 Twist 消息，停止底盘
的运动。
        twist = geo_msg.Twist() # 重新创建一个 Twist 消息对象
        twist.linear.y = -0.15 # 将 twist 对象的线速度设置为-0.15，表示向左移动
        self.mecanum_pub.publish(twist) # 发布 Twist 消息，控制底盘进行线性运动，使
车辆向左移动。
        rospy.sleep(0.25/0.15) # 同样暂停一段时间，确保车辆完成左移动作
        self.mecanum_pub.publish(geo_msg.Twist()) # 发布空的 Twist 消息，停止底盘
的运动。


    # 主要是一个图像处理的循环，根据条件不断地对图像进行处理，并根据处理结果来控制底盘
的运动和机器人的行为
    def image_proc(self):

        # 检查是否存在图像数据
        if self.image is not None:
            # 如果节点正在运行，进入循环
            while self.is_running:
                # 创建一个 Twist 消息对象，用于控制底盘的运动
                twist = geo_msg.Twist()
                # 利用 self.follower 对象的 get_binary 方法处理原始图像，获取二值化后
的图像。
                binary_image = self.follower.get_binary(self.image)
                # 检查是否满足停车条件，并且停车动作未开始。
                if 600 < self.park_pos[0] and 150 < self.park_pos[1] and not
self.start_park:
                    # 发布空的 Twist 消息，停止底盘的运动。
                    self.mecanum_pub.publish(geo_msg.Twist())

                    # 标记停车动作已开始
                    self.start_park = True

                    # 启动一个新的线程执行停车动作，避免阻塞主线程。
                    threading.Thread(target=self.park_action).start()
                    # 这样做的好处是，即使 self.park_action 方法中有耗时的操作，也不
会阻塞主线程的执行，提高了程序的并发性和响应速度。

                    # 设置停止标志，停止底盘运动。
                    self.stop = True

                # 如果 self.park_pos 元组的第一个元素大于 0，则设置 self.park 为 True，
标记正在停车。
                if 0 < self.park_pos[0]:
                    self.park = True

                # 如果 self.right_area 大于 1500，则设置 self.turn_right 为 True，标
记需要右转。
                if 1500 < self.right_area:
                    self.turn_right = True

                # 用于检测是否接近斑马线，并触发减速操作。
                if 440 < self.min_distance < 460:
                    self.count_cross_walk += 1 # 如果机器人检测到在斑马线附近，则
增加计数器

                    # 表示机器人连续检测到了接近斑马线的情况时
                    if self.count_cross_walk == 5:

                        # 将计数器重置为 0，以便下一次重新计数。
                        self.count_cross_walk = 0

                        # 一旦确认机器人接近斑马线，表示需要开始减速操作
                        self.start_slow_down = True
                        # 记录开始减速的时间
                        self.count_slow_down = rospy.get_time()
                else:
                    self.count_cross_walk = 0

                # 当机器人需要减速或在特定条件下调整其速度时的逻辑

                # 开始减速标志为真时
                if self.start_slow_down:

                    # 红灯信号（self.red）为真时
                    if self.red:

                        # 发布一个空的 Twist 消息到机器人的控制话题上，这将停止机器人
的所有移动。这是因为默认的 Twist 消息的线性和角速度都是 0。
                        self.mecanum_pub.publish(geo_msg.Twist())

                        # 设置 self.stop 标志为真，表示机器人应该保持静止状态
                        self.stop = True

                    # 绿灯信号（self.green）为真时
                    elif self.green:

                        # 设置机器人的线性速度为 0.1 米/秒，这可能是一个较慢的速度，
用于在绿灯时安全通过。
                        twist.linear.x = 0.1

                        # 清除停止标志，允许机器人移动。
                        self.stop = False

                    # 如果不是红灯也不是绿灯，即可能是黄灯或无交通灯控制情况。
                    else:

                        # 同样将速度设置为 0.1 米/秒，以保持较慢的速度。
                        twist.linear.x = 0.1

                        # 自从减速开始以来的时间是否已经足够让机器人通过斑马线区域
                        if rospy.get_time() - self.count_slow_down >
self.cross_walk_distance/twist.linear.x:

                    # 如果已经过的时间足够，则停止减速模式，允许机器人恢复正
常速度
                    self.start_slow_down = False

                # 如果不需要减速，则直接将机器人的线性速度设置为正常行驶速度 0.15 米/
秒
                else:
```

```
                    twist.linear.x = 0.15


                try:
                    # 如果机器人需要向右转
                    if self.turn_right:

                        # 从 binary_image（二值图像）中获取最大的 y 值，通常这是用来
识别图像中特定特征的垂直位置
                        y = self.follower.get_max_y(binary_image)

                        # 定义一个矩形区域（Region of Interest, ROI），这个区域从
图像的最大 y 值开始，到图像顶部
                        roi = [(0, y), (640, y), (640, 0), (0, 0)]

                        # 将定义的 roi 区域内的像素设置为黑色 [0, 0, 0]
                        cv2.fillPoly(binary_image, [np.array(roi)], [0, 0, 0])

                        # 使用 OpenCV 的 minMaxLoc 函数找出处理后的 binary_image
中亮度最低（即值最小，这里为黑色区域）的 x 坐标。这个坐标代表了图像中未被遮挡部分的最左
侧边缘，可能用于确定导航的起始点
                        min_x = cv2.minMaxLoc(binary_image)[-1][0]

                        # 这行代码在 binary_image 上绘制一条白色的线，从 (min_x, y)
到 (640, y)，线的宽度为 10 像素。这样的线条可能用于可视化或标记某种导航路径，比如机器人
应该遵循的道路边界或车道线。
                        cv2.line(binary_image, (min_x, y), (640, y), (255, 255,
255), 10)

                    # 如果机器人处于泊车模式
                    elif self.park:

                        # 从 binary_image 中获取最大的 x 值，用于定位最近的障碍或泊车
位置。

                        x, y = self.follower.get_max_x(binary_image)
                        #print(x, y)
                        #line_x = self.follower(binary_image,
self.image.copy())[-1]

                        # 在图像中绘制一条从(0, 450)到(x, y)的白色线，可能用于指示
泊车路线。
                        cv2.line(binary_image, (0, 450), (x, y), (255, 255,
255), 10)

                    result_image, angle, x = self.follower(binary_image,
self.image.copy())
                    # 使用处理后的 binary_image 和原始图像的副本，通过某种算法（可能
是路径跟踪或特征识别）来获取处理结果图像、角度和 x 位置
```

```
                    # 如果 x 不等于-1（有效的位置）并且机器人没有停止
                    if x != -1 and not self.stop:

                        if x > 200:
                            if self.turn_right:
                                # 表示开始执行右转动作。
                                self.start_turn = True
                            # 将机器人的角速度设置为 -0.45, 表示向右转动。
                            twist.angular.z = -0.45
                        else:
                            if self.start_turn:

                                # 如果机器人当前不需要右转, 但之前开始了右转动作,
那么将 self.count_turn 增加 1
                                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

                            # 这行代码设置了一个 PID 控制器的目标值为 80, 这个值通常
代表着期望的中心位置
                            self.pid.SetPoint = 80

                            # 这行代码更新了 PID 控制器的输出, 以便根据当前目标位置
x 来调整机器人的行为
                            self.pid.update(x)

                            # 这行代码将 PID 控制器的输出限制在 -0.8 到 0.8 之间,
并将结果赋给机器人的角速度, 以确保机器人的转向动作在可控范围内
                            twist.angular.z = misc.set_range(self.pid.output, -
0.8, 0.8)
                        #self.mecanum_pub.publish(twist)
                    else:
                        # 如果检测到的目标位置 x 为 -1, 或者机器人被命令停止, 那么清
除 PID 控制器的历史数据, 以便在下一次有效的目标检测时重新开始控制。
                        self.pid.clear()

                except Exception as e:
                    # 在发生异常时, 将结果图像设置为原始图像, 避免因错误处理导致的程
序崩溃
                    result_image = self.image
                    # 使用 ROS 的日志系统记录异常, 便于调试和监控系统状态
                    rospy.logerr(str(e))


                # 将 RGB 格式的图像 result_image 转换为 BGR 格式。在 OpenCV 中, 大多
数图像操作都基于 BGR 格式。
                bgr_image = cv2.cvtColor(result_image, cv2.COLOR_RGB2BGR)
                '''
                for i in self.objects_info:
                    color = self.colors(self.classes.index(i.class_name), True)
                    common.plot_one_box(
                    i.box,
                    bgr_image,
                    color=color,
                    label="{}:{:.2f}".format(
                        i.class_name, i.score
                    ),
                )
                '''
                # 将绘制了物体框的图像显示在名为 'result' 的窗口中。
                cv2.imshow('result', bgr_image)

                key = cv2.waitKey(1)
                # 如果用户按下了键盘上的某个按键 (按键码不等于-1)
                if key != -1:
                    # 将 self.is_running 设置为 False, 这将导致程序退出循环, 停止
运行
                    self.is_running = False
            #ros_image.data = result_image.tostring()
            #self.result_publisher.publish(ros_image)
        else:
            rospy.sleep(0.01)
        self.mecanum_pub.publish(geo_msg.Twist())


    # 用于处理从 '/yolov5/object_detect' 话题接收到的物体检测信息
    def get_object_callback(self, msg):

        # 将从消息中提取的物体检测信息存储在对象的 objects_info 属性中
        self.objects_info = msg.objects

        # 如果未检测到任何物体, 将执行以下操作
        if self.objects_info == []:

            # 等属性都将被重置为默认值或空值
            self.red = False
            self.green = False
            self.right_area = -1
            self.park_x = -1
            self.center = None
            self.min_distance = 0

        # 如果检测到了物体，将执行以下操作
        else:

            # 用于存储最小距离
            min_distance = 0
            # 遍历 self.objects_info 列表中的每个物体信息。
            for i in self.objects_info:

                # 获取当前物体的类别名称。
                class_name = i.class_name

                # 计算当前物体的中心坐标，并将其存储在 self.center 中。这里使用物体
边界框的左上角和右下角坐标来计算物体中心
                self.center = (int((i.box[0] + i.box[2])/2), int((i.box[1] +
i.box[3])/2))

                # 这行代码检查当前检测到的物体是否是右侧物体
                if class_name == 'right':

                    # 计算了右侧物体的面积。它通过使用物体边界框的宽度和高度来计算
                    self.right_area = abs((i.box[0] - i.box[2])*(i.box[1] -
i.box[3]))

                # 如果物体类别是 'crosswalk'
                if class_name == 'crosswalk':
                    # 检查当前物体的中心点的 y 坐标是否大于 min_distance，其中
min_distance 是一个变量，用于存储已知横穿斑马线物体中心点的最大 y 坐标。如果当前物体
的中心点 y 坐标更大，则更新 min_distance。
                    if self.center[1] > min_distance:
                        min_distance = self.center[1]

                # 如果物体类别是 'park'
                if class_name == 'park':
                    # 将停车点的中心坐标存储在 self.park_pos 中
                    self.park_pos = self.center

                # 如果物体类别分别是 'red' 和 'green'
                if class_name == 'red':

                    # 表示检测到了红色物体
                    self.red = True
                    # 这个计数器用于跟踪连续检测到的红色物体数量
                    self.count_red = 0
                # 如果当前物体不是红色的
                else:
                    self.count_red += 1
                if self.count_red == 2:
                    self.count_red = 0
                    self.red = False

                if class_name == 'green':
                    self.green = True
                    self.count_green = 0
                else:
                    self.count_green += 1
                if self.count_green == 2:
                    self.count_green = 0
                    self.green = False

            self.min_distance = min_distance

if __name__ == "__main__":
    LineFollowingNode('line_following')
```

2. Steps to Operate

1) Start JetAuto, and connect it to NoMachine.

2) Open the terminal.

3) Enter the command "**sudo systemctl stop start_app_node.service**" and press Enter to turn off app service.

4) Enter the command "**roslaunch jetauto_example self_driving.launch**" to enable autonomous driving service.

5) Open a new terminal, then input the command "**roscd jetauto_example/scripts/self_driving/**" to enter the directory containing programs.

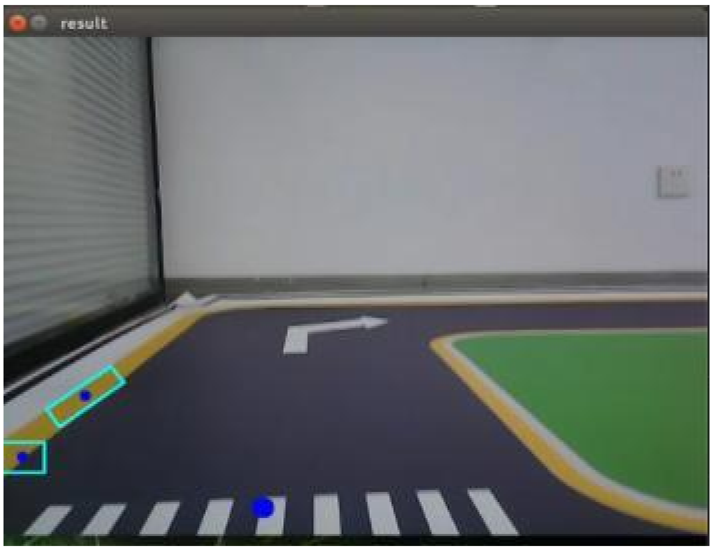
6) Enter the command "**python3 self_driving.py**" to run the program.

7) If you want to terminate the running program, press "**Ctrl+C**" on the terminal. If the program cannot be stopped, please try again.

3. Program Outcome

NOTE: remember to adjust robot's camera angle after placing it on the map to ensure the live camera feed will not include the robot's body. Otherwise, game performance will be affected.

After starting the game, place the robot on the road, and it will automatically detect the yellow line at the edge of the road. The robot will then adjust its position based on the detection results.



4. Program Analysis

4.1 Basic Configuration

◆ **Read setting file of color threshold**

Adopt the module "**get_yaml_data**" to obtain the setting file of color threshold.

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

◆ **Subscribe to the camera node**

Obtain the live camera feed through subscribing to the message released by camera node.

```
# 从 ROS 的参数服务器中获取参数-深度摄像头的名称
depth_camera = rospy.get_param('/depth_camera/camera_name', 'camera') # 获取参数

# 创建了一个 ROS 订阅者, 用于接收来自深度摄像头的原始 RGB 图像数据
rospy.Subscriber('/%s/rgb/image_raw'%depth_camera, Image,
self.image_callback)  # 摄像头订阅
```

◆ **Subscribe to yolov5 model detection node**

Invoke the detection model to obtain the detection result through subscribing the message released by yolov5 node.

```
# 创建了一个 ROS 订阅者, 用于接收 YOLOv5 目标检测节点发布的物体检测信息
rospy.Subscriber('/yolov5/object_detect', ObjectsInfo, self.get_object_callback)
```

4.2 Process Image

◆ **Gaussian Blurring**

Employ the function "**GaussianBlur()**" in cv2 library to apply Gaussian filter to process the image in order to reduce the noise in the image.

```
img_blur = cv2.GaussianBlur(img_lab, (3, 3), 3)  # 高斯模糊去噪
```

The first parameter “**img_lab**” is the input image.

The second parameter “**(3, 3)**” refers to the size of the Gaussian convolution kernel. Both the height and width of the convolution kernel must be positive number and odd number.

The third parameter “**3**” is the standard deviation in horizontal direction.

◆ **Perform binaryzation**

Employ the function “**inRange()**” in cv2 library to perform binaryzation on the image.

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

The first parameter “**img_blur**” is the input image.

The second parameter, represented by the code '**tuple(lab_data['lab'][self.target_color]['min'])**', specifies the minimum color threshold.

The third parameter, represented by the code '**tuple(lab_data['lab'][self.target_color]['max'])**', is the maximum color threshold.

The RGB value of a pixel above a certain threshold are set to 1 and values below the threshold are set to 0.

◆ **Perform erosion and dilation**

To make it easier to search for the target contour, the image is first subjected to erosion and dilation operations, which smooth out its edges.

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

The function **erode()** is used to perform erosion on the image. The meanings of the parameters enclosed in the parenthesis is as follows.

The first parameter “**mask**” is the input image.

“**cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))**” is the structuring element or kernel that decides the nature of the operation. The parameter in the parenthesis refers to the kernel shape and the second parameter is the kernel size.

The function **dilate()** is applied for image dilation. The meanings of the parameters are the same as **erode().**

◆ **Obtain the largest contour**

Utilize the function “**findContours()**” in cv2 library to search for the maximum contour in target color within the image.

```
contours = cv2.findContours(blob, cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_TC89_L1)[-
2]  # 查找每个区域中的轮廓。
```

The first parameter “**blob**” is the input image.

The second parameter “**cv2.RETR_EXTERNAL**” decides the contour retrieving mode.

The third parameter “**cv2.CHAIN_APPROX_TC89_L1**” specifies the contour approximation method.

4.3 Mark the Image

◆ Mark the Object

Adopt the module “**drawContours**” to mark the target object on the live camera feed.

```
cv2.drawContours(result_image, [box], -1, (0, 255, 255), 2)  # 画出四个点组成的矩形
```

The parameters in the parenthesis is as follows.

The first parameter “**result_image**” is the input image.

The second parameter “**[box]**” is the coordinate of the four corners of the drawn rectangle.

The third parameter “**-1**” represents all the contour can be drawn.

The fourth parameter “**(0, 255, 255)**” is the color of the rectangle.

The fifth parameter “**2**” is the width of the line.

4.4 Perform Action

◆ Set yellow to the target color

Set the instance object of LineFollower, and initialize the followed color to yellow.

```
self.follower = LineFollower("yellow")  # 创建了一个 LineFollower 类的实例, 用于处理线路跟踪任务
```

◆ Calibrate the position of the image relative to the yellow line

```
# 从 binary_image（二值图像）中获取最大的 y 值, 通常这是用来识别图像中特定特征的垂直位置
y = self.follower.get_max_y(binary_image)
```

Retrieve the height and width of the image, and divide the width in half to create image segments. Next, flip the segmented image over once the segmentation process is complete. Calculate the difference between the height of the yellow line image and subtract it from the difference between the height of the original video to determine the offset of the yellow line.

```
# 用于获取图像中最大的 y 坐标
    def get_max_y(self, image):
        h, w = image.shape[:2] # 获取图像的高度 (h) 和宽度 (w)
        roi = image[:, int(w/2):w] # 定义感兴趣区域 (ROI) ,即图像的右半部分
        flip_binary = cv2.flip(roi, 0) # 进行垂直翻转 (参数 0 表示沿 x 轴翻转)
        max_y = cv2.minMaxLoc(flip_binary)[-1][1] # 用于找到图像 (或 ROI 区域) 中的最小和最大灰度值及其位置。它返回四个值：最小值、最大值、最小值位置、最大值位置。
        # [-1][1] 取最大值的位置的 Y 坐标
        return h - max_y # 将坐标转换回原始图像的坐标系统
```

◆ **Change the movement direction according to the offset**

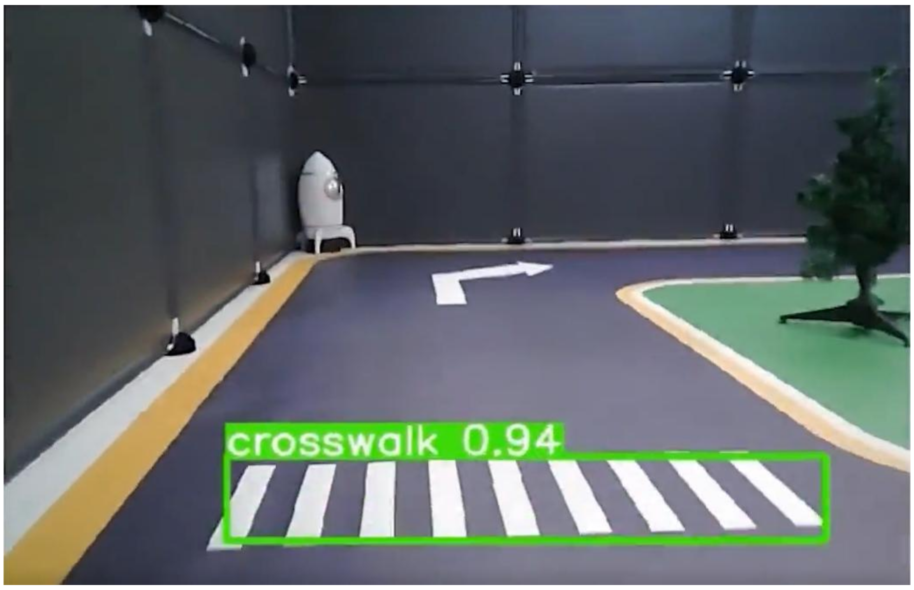
Define the anchor point, then update the data of pid. Lastly, adjust the path taking the threshold into consideration.

```
                            # 这行代码设置了一个 PID 控制器的目标值为 80，这个值通常
代表着期望的中心位置
                            self.pid.SetPoint = 80

                            # 这行代码更新了 PID 控制器的输出，以便根据当前目标位置
x 来调整机器人的行为
                            self.pid.update(x)

                            # 这行代码将 PID 控制器的输出限制在 -0.8 到 0.8 之间，
并将结果赋给机器人的角速度，以确保机器人的转向动作在可控范围内
                            twist.angular.z = misc.set_range(self.pid.output, -
0.8, 0.8)
```

Part 2 Road Sign Detection

1. Program Goal

The robot will be trained to recognize road signs and give a response based on the recognition result. For example, the robot will slow down when recognizing the zebra crossing, and restore normal speed after crossing the zebra crossing.



2. Preparation and Precaution

1) Unfold the map and lay it flat on the ground, making sure there are no obstructions in the way. For detailed instructions on how to position the map, please refer to the "**Place the Map and Assemble Tools (Must Read!!!)**" guide included in the same folder.

2) The model trained by yolov5 is used in this program.

3) Please operate in well-lit environment and avoid direct rays of light toprevent the robot from incorrect recognition.

3. Program Logic

To begin with, create a dataset that includes images of road signs indicating going straight, turning right, stopping, red light, green light, and zebra crossings. After collecting the images, use labelimg to label and frame the images needed for training. Next, convert the format of the collected data and segment the dataset.

Next, import the dataset into Yolov5 for training to get the model that can recognize the images trained previously. JetAuto will employ this model to recognize road signs and perform corresponding action based on the recognition result. To learn more about machine learning, please refer to the file in “**10. ROS + Machine Learning Part\ 2. Machine Learning Application**”.

JetAuto comes equipped with a pre-trained Yolov5 model capable of recognizing road signs. By transferring the live camera feed to the Yolov5 model, related road signs will be identified and the model will print the recognition confidence along with the framed image.

The source code of this program is located in
/home/jetauto_ws/src/jetauto_example/scripts/self_driving/self_driving.py

4. Steps to Operate

1) Start JetAuto, and then connect it to NoMachine.

2) Open command-line terminal.

3) Enter the command “**sudo systemctl stop start_app_node.service**” and press Enter key to turn off app service.

4) Run the command “**roslaunch jetauto_example self_driving.launch**” to enable autonomous driving service.

5) If you want to terminate the running program, use shortcut “**Ctrl+C**”. If the program fails to stop, please try again.

5. Program Outcome

Place the robot on the road. Then the robot will start going forward following the line and recognizing road signs. The trained Yolov5 model can recognize 6 types of road signs, including go straight, turn right, stop, red light, green light and zebra crossing.

6. Parameter Analysis

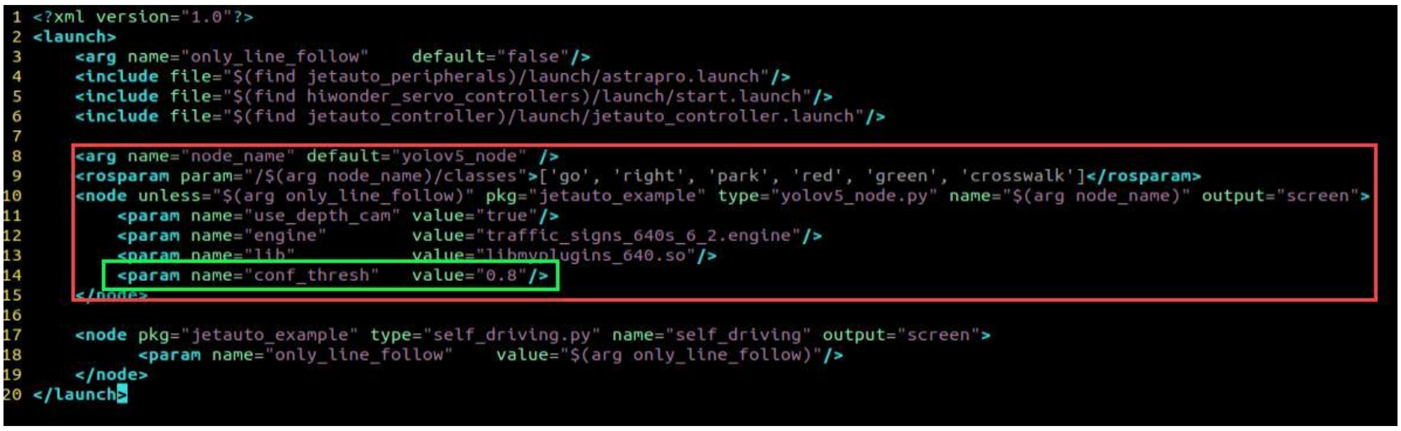
If the autonomous driving model fails to recognize or incorrectly recognizes a road sign, you can adjust the "**conf_thresh**" parameter in the "**self_driving.launch**" file. This parameter determines the confidence threshold for JetAuto to take the appropriate action after recognizing a road sign.

1) To initiate autonomous driving, first, lift JetAuto off the ground by placing it on a box or similar object. Next, run the command "**roslaunch jetauto_example self_driving.launch**" to start the self-driving module.

2) Open a new terminal, then execute the command “**rqt_image_view**” to open rqt to view how yolov5 model recognize.

3) Enter the command "**vim ~/jetauto_ws/src/jetauto_example/scripts/self_driving/self_driving.launch**" and press Enter key to check the file "**self_driving.launch**" through vim.

4) The content highlighted within a red frame utilizes the YOLOv5 model node, and to adjust its performance, you must modify the value located within the green frame to an appropriate value.



```
<?xml version="1.0"?>
<launch>
    <arg name="only_line_follow"    default="false"/>
    <include file="$(find jetauto_peripherals)/launch/astrapro.launch"/>
    <include file="$(find hiwonder_servo_controllers)/launch/start.launch"/>
    <include file="$(find jetauto_controller)/launch/jetauto_controller.launch"/>

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

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

Decrease the value of the parameter when the model cannot recognize the road signs.

Increase the value of the parameter if the model incorrectly recognize the road signs.

Press "**ESC**" key, and input "**：wq**" to save the value and close the file. After that, follow the steps in "**4.Steps to Operate**" to operate again.

7. Program Analysis

7.1 Basic Configuration

◆ **Read setting file of color threshold**

Adopt the module "**get_yaml_data**" to obtain the setting file of color threshold.

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

◆ **Subscribe to the camera node**

Obtain the live camera feed through subscribing to the message released by camera node.

```
# 从 ROS 的参数服务器中获取参数-深度摄像头的名称
depth_camera = rospy.get_param('/depth_camera/camera_name', 'camera') # 获取参数

# 创建了一个 ROS 订阅者, 用于接收来自深度摄像头的原始 RGB 图像数据
rospy.Subscriber('/%s/rgb/image_raw'%depth_camera, Image,
self.image_callback)  # 摄像头订阅
```

◆ **Subscribe to yolov5 model detection node**

Invoke the detection model to obtain the detection result through subscribing the message released by yolov5 node.

```
# 创建了一个 ROS 订阅者, 用于接收 YOLOv5 目标检测节点发布的物体检测信息
rospy.Subscriber('/yolov5/object_detect', ObjectsInfo, self.get_object_callback)
```

7.2 Process Image

◆ **Gaussian Blurring**

Employ the function “**GaussianBlur()**” in cv2 library to apply Gaussian filter to process the image in order to reduce the noise in the image.

```
img_blur = cv2.GaussianBlur(img_lab, (3, 3), 3)  # 高斯模糊去噪
```

The first parameter “**img_lab**” is the input image.

The second parameter “**(3, 3)**” refers to the size of the Gaussian convolution kernel. Both the height and width of the convolution kernel must be positive number and odd number.

The third parameter “**3**” is the standard deviation in horizontal direction.

◆ **Perform binaryzation**

Employ the function “**inRange()**” in cv2 library to perform binaryzation on the image.

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

The first parameter “**img_blur**” is the input image.

The second parameter, represented by the code '**tuple(lab_data['lab'][self.target_color]['min'])**', specifies the minimum color threshold.

The third parameter, represented by the code '**tuple(lab_data['lab'][self.target_color]['max'])**', is the maximum color threshold.

The RGB value of a pixel above a certain threshold are set to 1 and values below the threshold are set to 0.

◆ **Perform erosion and dilation**

To make it easier to search for the target contour, the image is first subjected to erosion and dilation operations, which smooth out its edges.

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

The function **erode()** is used to perform erosion on the image. The meanings of the parameters enclosed in the parenthesis is as follows.

The first parameter “**mask**” is the input image.

“**cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))**” is the structuring element or kernel that decides the nature of the operation. The parameter in the parenthesis refers to the kernel shape and the second parameter is the kernel size.

The function **dilate()** is applied for image dilation. The meanings of the parameters are the same as **erode().**

◆ **Obtain the largest contour**

Utilize the function "**findContours()**" in cv2 library to search for the maximum contour in target color within the image.

```
contours = cv2.findContours(blob, cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_TC89_L1)[-2]  # 查找每个区域中的轮廓。
```

The first parameter "**blob**" is the input image.

The second parameter "**cv2.RETR_EXTERNAL**" decides the contour retrieving mode.

The third parameter "**cv2.CHAIN_APPROX_TC89_L1**" specifies the contour approximation method.

7.3 Mark the Image

◆ **Mark the Object**

Adopt the module "**drawContours**" to mark the target object on the live camera feed.

```
cv2.drawContours(result_image, [box], -1, (0, 255, 255), 2)  # 画出四个点组成的矩形
```

The parameters in the parenthesis is as follows.

The first parameter "**result_image**" is the input image.

The second parameter "**[box]**" is the coordinate of the four corners of the drawn rectangle.

The third parameter "**-1**" represents all the contour can be drawn.

The fourth parameter "**(0, 255, 255)**" is the color of the rectangle.

The fifth parameter "**2**" is the width of the line.

7.4 Perform Action

◆ **Match with corresponding action**

Find the image matching with the road sign through yolov5 target detection node.

```
# 创建了一个 ROS 订阅者, 用于接收 YOLOv5 目标检测节点发布的物体检测信息
        rospy.Subscriber('/yolov5/object_detect', ObjectsInfo, self.get_object_callback)
        # '/yolov5/object_detect': 这是订阅的 ROS 主题代表了 YOLOv5 目标检测节点发布的物体检测信息
        # ObjectsInfo: 代表了订阅主题所传输的数据的类型, 用于表示物体检测信息
        # self.get_object_callback: 当订阅到新的物体检测信息时, 会调用 self.get_object_callback 函数。
```

JetAuto will perform corresponding actions based on the obtained road signs. If JetAuto receives the image labeled with "**right**", it will turn right and go forward. If it receives the image labeled with "**crosswalk**", it will slow down. If it gets the image labeled with "**park**", it will move to the specific position and parallel park.

```
# 遍历 self.objects_info 列表中的每个物体信息。
            for i in self.objects_info:

                # 获取当前物体的类别名称。
                class_name = i.class_name

                # 计算当前物体的中心坐标, 并将其存储在 self.center 中。这里使用物体边界框的左上角和右下角坐标来计算物体中心
                self.center = (int((i.box[0] + i.box[2])/2), int((i.box[1] + i.box[3])/2))

                # 这行代码检查当前检测到的物体是否是右侧物体
                if class_name == 'right':

                    # 计算了右侧物体的面积。它通过使用物体边界框的宽度和高度来计算
                    self.right_area = abs((i.box[0] - i.box[2])*(i.box[1] - i.box[3]))

                # 如果物体类别是 'crosswalk'
                if class_name == 'crosswalk':
                    # 检查当前物体的中心点的 y 坐标是否大于 min_distance, 其中 min_distance 是一个变量, 用于存储已知横穿斑马线物体中心点的最大 y 坐标。如果当前物体的中心点 y 坐标更大, 则更新 min_distance。
                    if self.center[1] > min_distance:
                        min_distance = self.center[1]

                # 如果物体类别是 'park'
                if class_name == 'park':
                    # 将停车点的中心坐标存储在 self.park_pos 中
                    self.park_pos = self.center

                # 如果物体类别分别是 'red' 和 'green'
                if class_name == 'red':

                    # 表示检测到了红色物体
                    self.red = True
                    # 这个计数器用于跟踪连续检测到的红色物体数量
                    self.count_red = 0
```

7.5 Yolov5 Model Detection

The source codes of this program are located in **/home/jetauto_ws/src/jetauto_example/scripts/self_driving/yolov5_trt.py**

```python
"""
An example that uses TensorRT's Python api to make inferences.
"""
import ctypes
import time
import cv2
import random
import numpy as np
import pycuda.autoinit
import pycuda.driver as cuda
import tensorrt as trt

class Colors:
    # Ultralytics color palette https://ultralytics.com/
    def __init__(self):
        # hex = matplotlib.colors.TABLEAU_COLORS.values()
        hex = ('FF3838', 'FF9D97', 'FF701F', 'FFB21D', 'CFD231', '48F90A',
'92CC17', '3DDB86', '1A9334', '00D4BB',
               '2C99A8', '00C2FF', '344593', '6473FF', '0018EC', '8438FF',
'520085', 'CB38FF', 'FF95C8', 'FF37C7')
        self.palette = [self.hex2rgb('#' + c) for c in hex]
        self.n = len(self.palette)

    def __call__(self, i, bgr=False):
        c = self.palette[int(i) % self.n]
        return (c[2], c[1], c[0]) if bgr else c

    @staticmethod
    def hex2rgb(h):  # rgb order (PIL)
        return tuple(int(h[1 + i:1 + i + 2], 16) for i in (0, 2, 4))

colors = Colors()  # create instance for 'from utils.plots import colors'

def plot_one_box(x, img, color=None, label=None, line_thickness=None):
    """
    description: Plots one bounding box on image img,
                 this function comes from YoLov5 project.
    param:
        x:      a box likes [x1,y1,x2,y2]
        img:    a opencv image object
        color:  color to draw rectangle, such as (0,255,0)
        label:  str
        line_thickness: int
    return:
        no return

    """
    tl = (
        line_thickness or round(0.002 * (img.shape[0] + img.shape[1]) / 2) + 1
    )  # line/font thickness
    color = color or [random.randint(0, 255) for _ in range(3)]
    c1, c2 = (int(x[0]), int(x[1])), (int(x[2]), int(x[3]))
    cv2.rectangle(img, c1, c2, color, thickness=tl, lineType=cv2.LINE_AA)
    if label:
        tf = max(tl - 1, 1)  # font thickness
        t_size = cv2.getTextSize(label, 0, fontScale=tl / 3, thickness=tf)[0]
        c2 = c1[0] + t_size[0], c1[1] - t_size[1] - 3
        cv2.rectangle(img, c1, c2, color, -1, cv2.LINE_AA)  # filled
        cv2.putText(
            img,
            label,
            (c1[0], c1[1] - 2),
            0,
            tl / 3,
            [225, 255, 255],
            thickness=tf,
            lineType=cv2.LINE_AA,
        )


class YoLov5TRT(object):
    """
    description: A YOLOv5 class that warps TensorRT ops, preprocess and
postprocess ops.
    """

    def __init__(self, engine_file_path, plugin, classes, conf_thresh=0.8,
iou_threshold=0.4):
        self.CONF_THRESH = conf_thresh
        self.IOU_THRESHOLD = iou_threshold

        PLUGIN_LIBRARY = plugin
        self.engine_file_path = engine_file_path

        # load labels
        self.categories = classes

        ctypes.CDLL(PLUGIN_LIBRARY)

        # Create a Context on this device,
        self.ctx = cuda.Device(0).make_context()
        stream = cuda.Stream()
        TRT_LOGGER = trt.Logger(trt.Logger.INFO)
        runtime = trt.Runtime(TRT_LOGGER)

        # Deserialize the engine from file
        with open(self.engine_file_path, "rb") as f:
            engine = runtime.deserialize_cuda_engine(f.read())
        context = engine.create_execution_context()

        host_inputs = []
        cuda_inputs = []
        host_outputs = []
        cuda_outputs = []
        bindings = []

        for binding in engine:
            print('bingding:', binding, engine.get_binding_shape(binding))
            size = trt.volume(engine.get_binding_shape(binding)) *
engine.max_batch_size
            dtype = trt.nptype(engine.get_binding_dtype(binding))
            # Allocate host and device buffers
            host_mem = cuda.pagelocked_empty(size, dtype)
            cuda_mem = cuda.mem_alloc(host_mem.nbytes)
            # Append the device buffer to device bindings.
            bindings.append(int(cuda_mem))
            # Append to the appropriate list.
            if engine.binding_is_input(binding):
                self.input_w = engine.get_binding_shape(binding)[-1]
                self.input_h = engine.get_binding_shape(binding)[-2]
                host_inputs.append(host_mem)
                cuda_inputs.append(cuda_mem)
            else:
                host_outputs.append(host_mem)
                cuda_outputs.append(cuda_mem)

        # Store
        self.stream = stream
        self.context = context
        self.engine = engine
        self.host_inputs = host_inputs
        self.cuda_inputs = cuda_inputs
        self.host_outputs = host_outputs
        self.cuda_outputs = cuda_outputs
        self.bindings = bindings
        self.batch_size = engine.max_batch_size

    def infer(self, raw_image_generator):
        # Make self the active context, pushing it on top of the context stack.
        self.ctx.push()
        # Restore
        stream = self.stream
        context = self.context
        engine = self.engine
        host_inputs = self.host_inputs
        cuda_inputs = self.cuda_inputs
        host_outputs = self.host_outputs
        cuda_outputs = self.cuda_outputs
        bindings = self.bindings
        # Do image preprocess
        batch_image_raw = []
        batch_origin_h = []
        batch_origin_w = []
        batch_input_image = np.empty(shape=[self.batch_size, 3, self.input_h,
self.input_w])

        input_image, image_raw, origin_h, origin_w =
self.preprocess_image(raw_image_generator)
        batch_image_raw.append(image_raw)
        batch_origin_h.append(origin_h)
        batch_origin_w.append(origin_w)
        np.copyto(batch_input_image[0], input_image)

        batch_input_image = np.ascontiguousarray(batch_input_image)

        # Copy input image to host buffer
        np.copyto(host_inputs[0], batch_input_image.ravel())
        start = time.time()
        # Transfer input data  to the GPU.
        cuda.memcpy_htod_async(cuda_inputs[0], host_inputs[0], stream)
        # Run inference.
        context.execute_async(batch_size=self.batch_size, bindings=bindings,
stream_handle=stream.handle)
        # Transfer predictions back from the GPU.
        cuda.memcpy_dtoh_async(host_outputs[0], cuda_outputs[0], stream)
        # Synchronize the stream
        stream.synchronize()
        end = time.time()
        # Remove any context from the top of the context stack, deactivating it.
        self.ctx.pop()
        # Here we use the first row of output in that batch_size = 1
        output = host_outputs[0]
        # Do postprocess
        boxes = []
        scores = []
        classid = []
        for i in range(self.batch_size):
            result_boxes, result_scores, result_classid = self.post_process(
                output[i * 6001: (i + 1) * 6001], batch_origin_h[i],
batch_origin_w[i]
            )
            # Draw rectangles and labels on the original image
            for j in range(len(result_boxes)):
                '''
                box = result_boxes[j]
                color = colors(int(result_classid[j]), True)
                plot_one_box(
                    box,
                    batch_image_raw[i],
                    color=color,
                    label="{}:{:.2f}".format(
                        self.categories[int(result_classid[j])],
result_scores[j]
                    ),
                )
                '''
                boxes.extend([result_boxes[j]])
                scores.append(result_scores[j])
                classid.append(int(result_classid[j]))
        #print(int(1/(end - start)))
        return boxes, scores, classid

    def destroy(self):
        # Remove any context from the top of the context stack, deactivating it.
        self.ctx.pop()

    def get_raw_image(self, image_path_batch):
        """
        description: Read an image from image path
        """
        for img_path in image_path_batch:
            yield cv2.imread(img_path)

    def get_raw_image_zeros(self, image_path_batch=None):
        """
        description: Ready data for warmup
        """
        for _ in range(self.batch_size):
            yield np.zeros([self.input_h, self.input_w, 3], dtype=np.uint8)

    def preprocess_image(self, raw_bgr_image):
        """
        description: Convert BGR image to RGB,
                     resize and pad it to target size, normalize to [0,1],
                     transform to NCHW format.
        param:
            input_image_path: str, image path
        return:
            image:  the processed image
            image_raw: the original image
            h: original height
            w: original width
        """
        image_raw = raw_bgr_image
        h, w, c = image_raw.shape
        image = cv2.cvtColor(image_raw, cv2.COLOR_BGR2RGB)
        # Calculate widht and height and paddings
        r_w = self.input_w / w
        r_h = self.input_h / h
        if r_h > r_w:
            tw = self.input_w
            th = int(r_w * h)
            tx1 = tx2 = 0
            ty1 = int((self.input_h - th) / 2)
            ty2 = self.input_h - th - ty1
        else:
            tw = int(r_h * w)
            th = self.input_h
            tx1 = int((self.input_w - tw) / 2)
            tx2 = self.input_w - tw - tx1
            ty1 = ty2 = 0
        # Resize the image with long side while maintaining ratio
        image = cv2.resize(image, (tw, th))
        # Pad the short side with (128,128,128)
        image = cv2.copyMakeBorder(
            image, ty1, ty2, tx1, tx2, cv2.BORDER_CONSTANT, None, (128, 128, 
128)
        )
        image = image.astype(np.float32)
        # Normalize to [0,1]
        image /= 255.0
        # HWC to CHW format:
        image = np.transpose(image, [2, 0, 1])
        # CHW to NCHW format
        image = np.expand_dims(image, axis=0)
        # Convert the image to row-major order, also known as "C order":
        image = np.ascontiguousarray(image)
        return image, image_raw, h, w

    def xywh2xyxy(self, origin_h, origin_w, x):
        """
        description:    Convert nx4 boxes from [x, y, w, h] to [x1, y1, x2, y2]
where xy1=top-left, xy2=bottom-right
        param:
            origin_h:   height of original image
            origin_w:   width of original image
            x:          A boxes numpy, each row is a box [center_x, center_y, w,
h]
        return:
            y:          A boxes numpy, each row is a box [x1, y1, x2, y2]
        """
        y = np.zeros_like(x)
        r_w = self.input_w / origin_w
        r_h = self.input_h / origin_h
        if r_h > r_w:
            y[:, 0] = x[:, 0] - x[:, 2] / 2
            y[:, 2] = x[:, 0] + x[:, 2] / 2
            y[:, 1] = x[:, 1] - x[:, 3] / 2 - (self.input_h - r_w * origin_h) /
2
            y[:, 3] = x[:, 1] + x[:, 3] / 2 - (self.input_h - r_w * origin_h) /
2
            y /= r_w
        else:
            y[:, 0] = x[:, 0] - x[:, 2] / 2 - (self.input_w - r_h * origin_w) /
2
            y[:, 2] = x[:, 0] + x[:, 2] / 2 - (self.input_w - r_h * origin_w) /
2
            y[:, 1] = x[:, 1] - x[:, 3] / 2
            y[:, 3] = x[:, 1] + x[:, 3] / 2
            y /= r_h

        return y

    def post_process(self, output, origin_h, origin_w):
        """
        description: postprocess the prediction
        param:
            output:     A numpy likes [num_boxes,cx,cy,w,h,conf,cls_id,
cx,cy,w,h,conf,cls_id, ...]
            origin_h:   height of original image
            origin_w:   width of original image
        return:
            result_boxes: finally boxes, a boxes numpy, each row is a box [x1,
y1, x2, y2]
            result_scores: finally scores, a numpy, each element is the score
correspoing to box
            result_classid: finally classid, a numpy, each element is the
classid correspoing to box
```

```
        """
        # Get the num of boxes detected
        num = int(output[0])
        # Reshape to a two dimentional ndarray
        pred = np.reshape(output[1:], (-1, 6))[:num, :]
        # Do nms
        boxes = self.non_max_suppression(pred, origin_h, origin_w,
conf_thres=self.CONF_THRESH, nms_thres=self.IOU_THRESHOLD)
        result_boxes = boxes[:, :4] if len(boxes) else np.array([])
        result_scores = boxes[:, 4] if len(boxes) else np.array([])
        result_classid = boxes[:, 5] if len(boxes) else np.array([])
        return result_boxes, result_scores, result_classid

    def bbox_iou(self, box1, box2, x1y1x2y2=True):
        """
        description: compute the IoU of two bounding boxes
        param:
            box1: A box coordinate (can be (x1, y1, x2, y2) or (x, y, w, h))
            box2: A box coordinate (can be (x1, y1, x2, y2) or (x, y, w,
h))
            x1y1x2y2: select the coordinate format
        return:
            iou: computed iou
        """
        if not x1y1x2y2:
            # Transform from center and width to exact coordinates
            b1_x1, b1_x2 = box1[:, 0] - box1[:, 2] / 2, box1[:, 0] + box1[:, 2]
/ 2
            b1_y1, b1_y2 = box1[:, 1] - box1[:, 3] / 2, box1[:, 1] + box1[:, 3]
/ 2
            b2_x1, b2_x2 = box2[:, 0] - box2[:, 2] / 2, box2[:, 0] + box2[:, 2]
/ 2
            b2_y1, b2_y2 = box2[:, 1] - box2[:, 3] / 2, box2[:, 1] + box2[:, 3]
/ 2
        else:
            # Get the coordinates of bounding boxes
            b1_x1, b1_y1, b1_x2, b1_y2 = box1[:, 0], box1[:, 1], box1[:, 2],
box1[:, 3]
            b2_x1, b2_y1, b2_x2, b2_y2 = box2[:, 0], box2[:, 1], box2[:, 2],
box2[:, 3]

        # Get the coordinates of the intersection rectangle
        inter_rect_x1 = np.maximum(b1_x1, b2_x1)
        inter_rect_y1 = np.maximum(b1_y1, b2_y1)
        inter_rect_x2 = np.minimum(b1_x2, b2_x2)
        inter_rect_y2 = np.minimum(b1_y2, b2_y2)
        # Intersection area
        inter_area = np.clip(inter_rect_x2 - inter_rect_x1 + 1, 0, None) * \
                     np.clip(inter_rect_y2 - inter_rect_y1 + 1, 0, None)
        # Union Area
        b1_area = (b1_x2 - b1_x1 + 1) * (b1_y2 - b1_y1 + 1)
        b2_area = (b2_x2 - b2_x1 + 1) * (b2_y2 - b2_y1 + 1)

        iou = inter_area / (b1_area + b2_area - inter_area + 1e-16)

        return iou

    def non_max_suppression(self, prediction, origin_h, origin_w,
conf_thres=0.5, nms_thres=0.4):
        """
        description: Removes detections with lower object confidence score than
'conf_thres' and performs
        Non-Maximum Suppression to further filter detections.
        param:
            prediction: detections, (x1, y1, x2, y2, conf, cls_id)
            origin_h: original image height
            origin_w: original image width
            conf_thres: a confidence threshold to filter detections
            nms_thres: a iou threshold to filter detections
        return:
            boxes: output after nms with the shape (x1, y1, x2, y2, conf,
cls_id)
        """
        # Get the boxes that score > CONF_THRESH
        boxes = prediction[prediction[:, 4] >= conf_thres]
        # Trandform bbox from [center_x, center_y, w, h] to [x1, y1, x2, y2]
        boxes[:, :4] = self.xywh2xyxy(origin_h, origin_w, boxes[:, :4])
        # clip the coordinates
        boxes[:, 0] = np.clip(boxes[:, 0], 0, origin_w -1)
        boxes[:, 2] = np.clip(boxes[:, 2], 0, origin_w -1)
        boxes[:, 1] = np.clip(boxes[:, 1], 0, origin_h -1)
        boxes[:, 3] = np.clip(boxes[:, 3], 0, origin_h -1)
        # Object confidence
        confs = boxes[:, 4]
        # Sort by the confs
        boxes = boxes[np.argsort(-confs)]
        # Perform non-maximum suppression
        keep_boxes = []
        while boxes.shape[0]:
            large_overlap = self.bbox_iou(np.expand_dims(boxes[0, :4], 0),
boxes[:, :4]) > nms_thres
            label_match = boxes[0, -1] == boxes[:, -1]
            # Indices of boxes with lower confidence scores, large IOUs and
matching labels
```

```
            invalid = large_overlap & label_match
            keep_boxes += [boxes[0]]
            boxes = boxes[~invalid]
        boxes = np.stack(keep_boxes, 0) if len(keep_boxes) else np.array([])
        return boxes

if __name__ == "__main__":
    classes = ['go', 'right', 'park', 'red', 'green', 'crosswalk']
    # a YoLov5TRT instance
    yolov5_wrapper = YoLov5TRT('./traffic_signs_640s_6_2.engine',
'./libmyplugins_640.so', classes)
    try:
        cap = cv2.VideoCapture("/dev/astrapro")
        # cap = cv2.VideoCapture("/dev/usb_cam")
        while True:
            t = time.time()
            ret, frame = cap.read()
            if ret:
                boxes, scores, classid = yolov5_wrapper.infer(frame)
                for box, cls_conf, cls_id in zip(boxes, scores, classid):
                    color = colors(cls_id, True)
                    plot_one_box(
                    box,
                    frame,
                    color=color,
                    label="{}:{:.2f}".format(
                        classes[cls_id], cls_conf
                    ),
                )
                cv2.imshow('frame', frame)
                key = cv2.waitKey(1)
                if key != -1:
                    break
    finally:
        # destroy the instance
        yolov5_wrapper.destroy()
```

Display the recognition result on the screen through the function “**plot_one_box**”. The meanings of the parameters are as follows.

The first parameter “**x**” is the coordinate of the upper left corner and lower right corner of the rectangle. The rectangle is used to frame the recognized image.

The second parameter “**img**” is the input image.

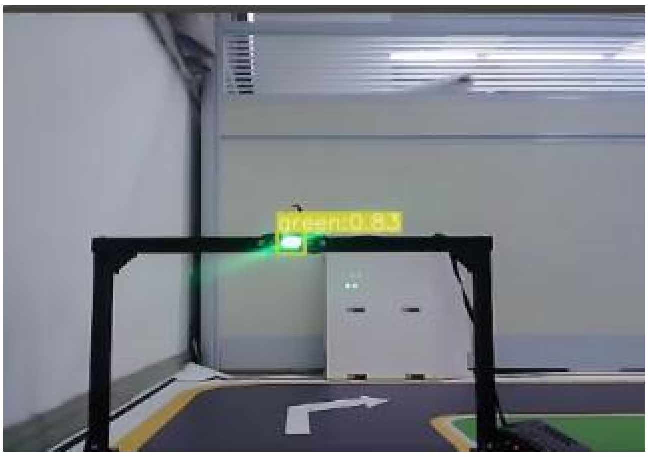
The third parameter “**color**” is the color of the rectangle which is based on the types of the image.

The fourth parameter, '**label**', contains the labeled content which includes the type of the image and its corresponding confidence level."

Part 3 Traffic Light Recognition

1. Program Goal

In this section, the robot will recognize road signs and provide corresponding feedback based on the recognition result. For example, if the robot recognizes a green light, it will continue driving. If it recognizes a red light, the robot will stop and wait.



2. Preparation and Precautions

1) Unfold the map and lay it flat on the ground, making sure there are no obstructions in the way. For detailed instructions on how to position the map, please refer to the "**Place the Map and Assemble Tools (Must Read!!!)**" guide included in the same folder.

2) After installing the traffic lights, download the program to the CoreX controller. For specific download methods, please refer to the file stored in "**Place the Map and Assemble Tools (Must Read!!!)\ Download Traffic Light Program**".

3) The traffic light model used in this section is a model trained by YOLOv5. To learn more about YOLOv5, please refer to chapter 10.

3. Program Logic

To begin with, create a dataset that includes images of road signs indicating going straight, turning right, stopping, red light, green light, and zebra crossings. After collecting the images, use labelimg to label and frame the images needed for training. Next, convert the format of the collected data and segment the dataset.

Next, import the dataset into Yolov5 for training to get the model that can recognize the images trained previously. JetAuto will employ this model to recognize road signs and perform corresponding action based on the recognition result. To learn more about machine learning, please refer to the file in “**10. ROS + Machine Learning Part\ 2. Machine Learning Application**”.

JetAuto comes equipped with a pre-trained Yolov5 model capable of recognizing road signs. By transferring the live camera feed to the Yolov5 model, related road signs will be identified and the model will print the recognition confidence along with the framed image.

The source code of this program is located in
/home/jetauto_ws/src/jetauto_example/scripts/self_driving/self_driving.py

4. Steps to Operate
1) Start JetAuto, and then connect it to NoMachine.

2) Open command-line terminal.

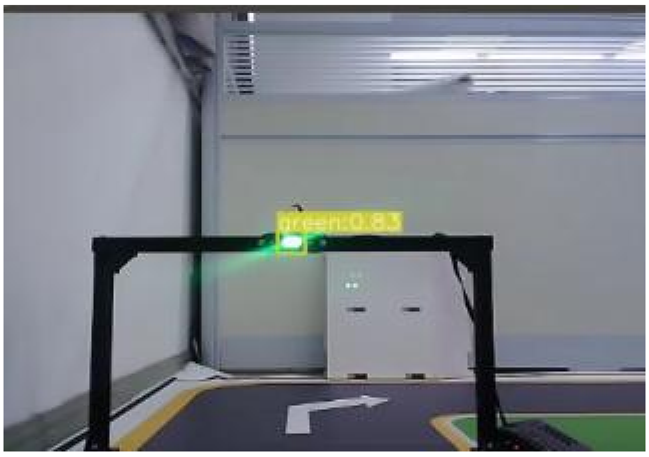
3) Enter the command "**sudo systemctl stop start_app_node.service**" and press Enter key to turn off app service.

4) Run the command "**roslaunch jetauto_example self_driving.launch**" to enable autonomous driving service.

5) If you want to terminate the running program, use shortcut "**Ctrl+C**". If the program fails to stop, please try again.

5. Program Outcome

Place the robot on the road. Then the robot will start moving to the traffic light and recognize the traffic light. When recognizing a red light, it will stop and wait. When recognizing a green light, it will go forward.



6. Parameter Analysis

If JetAuto fails to recognize or incorrectly recognizes the traffic light, please adjust the parameters according to "**Session 6 Parameter Analysis**" in "**Part 2 Road Sign Detection**".

7. Program Analysis

7.1 Basic Configuration

◆ **Read setting file of color threshold**

Adopt the module "**get_yaml_data**" to obtain the setting file of color threshold.

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

◆ **Subscribe to the camera node**

Obtain the live camera feed through subscribing to the message released by camera node.

```
# 从 ROS 的参数服务器中获取参数-深度摄像头的名称
depth_camera = rospy.get_param('/depth_camera/camera_name', 'camera') # 获取参数

# 创建了一个 ROS 订阅者, 用于接收来自深度摄像头的原始 RGB 图像数据
```

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

◆ **Subscribe to yolov5 model detection node**

Invoke the detection model to obtain the detection result through subscribing the message released by yolov5 node.

```
# 创建了一个 ROS 订阅者, 用于接收 YOLOv5 目标检测节点发布的物体检测信息
rospy.Subscriber('/yolov5/object_detect', ObjectsInfo, self.get_object_callback)
```

7.2 Process Image

◆ **Gaussian Blurring**

Employ the function "**GaussianBlur()**" in cv2 library to apply Gaussian filter to process the image in order to reduce the noise in the image.

```
img_blur = cv2.GaussianBlur(img_lab, (3, 3), 3)  # 高斯模糊去噪
```

The first parameter "**img_lab**" is the input image.

The second parameter "**(3, 3)**" refers to the size of the Gaussian convolution kernel. Both the height and width of the convolution kernel must be positive number and odd number.

The third parameter "**3**" is the standard deviation in horizontal direction.

◆ **Perform binaryzation**

Employ the function "**inRange()**" in cv2 library to perform binaryzation on the image.

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

The first parameter "**img_blur**" is the input image.

The second parameter, represented by the code '**tuple(lab_data['lab'][self.target_color]['min'])**', specifies the minimum color threshold.

The third parameter, represented by the code '**tuple(lab_data['lab'][self.target_color]['max'])**', is the maximum color threshold.

The RGB value of a pixel above a certain threshold are set to 1 and values below the threshold are set to 0.

◆ **Perform erosion and dilation**

To make it easier to search for the target contour, the image is first subjected to erosion and dilation operations, which smooth out its edges.

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

The function **erode()** is used to perform erosion on the image. The meanings of the parameters enclosed in the parenthesis is as follows.

The first parameter "**mask**" is the input image.

"**cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))**" is the structuring element or kernel that decides the nature of the operation. The parameter in the parenthesis refers to the kernel shape and the second parameter is the kernel size.

The function **dilate()** is applied for image dilation. The meanings of the parameters are the same as **erode().**

◆ **Obtain the largest contour**

Utilize the function "**findContours()**" in cv2 library to search for the maximum contour in target color within the image.

```
contours = cv2.findContours(blob, cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_TC89_L1)[-
2]  # 查找每个区域中的轮廓。
```

The first parameter "**blob**" is the input image.

The second parameter "**cv2.RETR_EXTERNAL**" decides the contour retrieving mode.

The third parameter "**cv2.CHAIN_APPROX_TC89_L1**" specifies the contour approximation method.

7.3 Mark the Image

◆ **Mark the Object**

Adopt the module "**drawContours**" to mark the target object on the live camera feed.

```
cv2.drawContours(result_image, [box], -1, (0, 255, 255), 2)  # 画出四个点组成的
矩形
```

The parameters in the parenthesis is as follows.

The first parameter "**result_image**" is the input image.

The second parameter "**[box]**" is the coordinate of the four corners of the drawn rectangle.

The third parameter "**-1**" represents all the contour can be drawn.

The fourth parameter "**(0, 255, 255)**" is the color of the rectangle.

The fifth parameter "**2**" is the width of the line.

7.4 Perform Action

◆ **Match with corresponding action**

Retrieve the image type from the recognition result, which includes the coordinates of the rectangle contour framing the image and the recognition confidence.

```
# 遍历 self.objects_info 列表中的每个物体信息。
```

```
            for i in self.objects_info:

                # 获取当前物体的类别名称。
                class_name = i.class_name
```

Match with the image labeled “**red**” or “**green**”, then set the corresponding identifier as True.

```
# 如果物体类别分别是 'red' 和 'green'
                if class_name == 'red':

                    # 表示检测到了红色物体
                    self.red = True
                    # 这个计数器用于跟踪连续检测到的红色物体数量
                    self.count_red = 0

                # 如果当前物体不是红色的
                else:
                    self.count_red += 1
                if self.count_red == 2:
                    self.count_red = 0
                    self.red = False

                if class_name == 'green':
                    self.green = True
                    self.count_green = 0
                else:
                    self.count_green += 1
                if self.count_green == 2:
                    self.count_green = 0
                    self.green = False
```

◆ **Execute the Corresponding Action**

Publish the topic to drive the motors based on the status of the signal light.

```
# 红灯信号（self.red）为真时
                    if self.red:

                        # 发布一个空的 Twist 消息到机器人的控制话题上，这将停止机器人
的所有移动。这是因为默认的 Twist 消息的线性和角速度都是 0。
                        self.mecanum_pub.publish(geo_msg.Twist())

                        # 设置 self.stop 标志为真，表示机器人应该保持静止状态
                        self.stop = True

                    # 绿灯信号（self.green）为真时
                    elif self.green:
```

```
                    # 设置机器人的线性速度为0.1 米/秒，这可能是一个较慢的速度，用于在绿灯时安全通过。
                    twist.linear.x = 0.1

                    # 清除停止标志，允许机器人移动。
                    self.stop = False
```

When it is a red light, set the robot to stop. When it is a green light, set the linear velocity in the direction of positive X=axis to 0.1m/s to enable the robot to keep going forward.When the signal light is red, instruct the robot to stop. When the signal light is green, set the linear velocity along the positive X-axis to 0.1m/s, allowing the robot to move forward.

7.5 Yolov5 Model Detection

The source codes of this program are located in **/home/jetauto_ws/src/jetauto_example/scripts/self_driving/yolov5_trt.py**

Display the recognition result on the screen through the function “**plot_one_box**”. The meanings of the parameters are as follows.

The first parameter “**x**” is the coordinate of the upper left corner and lower right corner of the rectangle. The rectangle is used to frame the recognized image.

The second parameter “**img**” is the input image.

The third parameter “**color**” is the color of the rectangle which is based on the types of the image.

The fourth parameter, '**label**', contains the labeled content which includes the type of the image and its corresponding confidence level."

The fifth parameter, “**line_thickness**”, is the width of the outline of rectangle.

Part 4 Turning Decision-Making

1. Program Goal

In general, JetAuto will go forward following the yellow line on its left. When recognizing the road sign of turning left, it will turn left. Once recognizing the road sign of turning right, it will turn right.

2. Preparation and Precautions

1) Unfold the map and lay it flat on the ground, making sure there are no obstructions in the way. For detailed instructions on how to position the map, please refer to the "**Place the Map and Assemble Tools (Must Read!!!)**" guide included in the same folder.

2) The road sign used in this section is a model trained by YOLOv5. To learn more about YOLOv5, please refer to **chapter 10**.

3) Please operate in well-lit environment, but avoid direct light rays which will cause JetAuto to incorrectly recognize.

3. Program Logic

The workflow of the program in this session is the same as “**Part 2 Road Sign Detection**”.

The source code of this program is stored in
/home/jetauto_ws/src/jetauto_example/scripts/self_driving/self_driving.py

4. Steps to Operate

1) Start JetAuto, and then connect it to NoMachine.

2) Open command-line terminal.

3) Enter the command “**sudo systemctl stop start_app_node.service**” and press Enter key to turn off app service.

4) Run the command “**roslaunch jetauto_example self_driving.launch**” to enable autonomous driving service.

5) If you want to terminate the running program, use shortcut “**Ctrl+C**”. If the program fails to stop, please try again.

5. Program Outcome

The robot will turn right when recognizing the road sign of turning right.

6. Parameter Analysis

If JetAuto fails to recognize or incorrectly recognizes the road sign of turning right, please adjust the parameters according to the below instructions.

```
# 这行代码检查当前检测到的物体是否是右侧物体
                if class_name == 'right':

                    # 计算了右侧物体的面积。它通过使用物体边界框的宽度和高度来计算
                    self.right_area = abs((i.box[0] - i.box[2])*(i.box[1] -
i.box[3]))
```

To start, it's important to have a basic understanding of the parameter '**self.right_area**' in the code '**self_driving.py**'. This parameter specifically refers to the area of the road sign indicating a right turn.

The code shown in the image above can be utilized to compute the area of the road sign indicating a right turn that has been identified by the yolov5 model. As the robot approaches the sign, the value of the parameter 'self.right_area' increases.

```
# 如果 self.right_area 大于 1500, 则设置 self.turn_right 为 True, 标记需要右转。
                if 1500 < self.right_area:
                    self.turn_right = True
```

Next, you need to set the threshold for the recognized area, for example 1500, to allow JetAuto to decide when to turn right.

If you want to let the robot turn right only when it get close to the road sign, increase this value.

However, if you want the robot to turn right when it is farther away from the traffic sign, decrease the value.

7. Program Analysis

7.1 Basic Configuration

◆ **Read setting file of color threshold**

Adopt the module “**get_yaml_data**” to obtain the setting file of color threshold.

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

◆ **Subscribe to the camera node**

Obtain the live camera feed through subscribing to the message released by camera node.

```
# 从 ROS 的参数服务器中获取参数-深度摄像头的名称
depth_camera = rospy.get_param('/depth_camera/camera_name', 'camera') # 获取参数

# 创建了一个 ROS 订阅者, 用于接收来自深度摄像头的原始 RGB 图像数据
rospy.Subscriber('/%s/rgb/image_raw'%depth_camera, Image,
self.image_callback)  # 摄像头订阅
```

◆ **Subscribe to yolov5 model detection node**

Invoke the detection model to obtain the detection result through subscribing the message released by yolov5 node.

```
# 创建了一个 ROS 订阅者, 用于接收 YOLOv5 目标检测节点发布的物体检测信息
rospy.Subscriber('/yolov5/object_detect', ObjectsInfo, self.get_object_callback)
```

7.2 Process Image

◆ **Gaussian Blurring**

Employ the function “**GaussianBlur()**” in cv2 library to apply Gaussian filter to process the image in order to reduce the noise in the image.

```
img_blur = cv2.GaussianBlur(img_lab, (3, 3), 3)  # 高斯模糊去噪
```

The first parameter “**img_lab**” is the input image.

The second parameter “**(3, 3)**” refers to the size of the Gaussian convolution kernel. Both the height and width of the convolution kernel must be positive number and odd number.

The third parameter “**3**” is the standard deviation in horizontal direction.

◆ **Perform binaryzation**

Employ the function “**inRange()**” in cv2 library to perform binaryzation on the image.

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

The first parameter “**img_blur**” is the input image.

The second parameter, represented by the code '**tuple(lab_data['lab'][self.target_color]['min'])**', specifies the minimum color threshold.

The third parameter, represented by the code '**tuple(lab_data['lab'][self.target_color]['max'])**', is the maximum color threshold.

The RGB value of a pixel above a certain threshold are set to 1 and values below the threshold are set to 0.

◆ **Perform erosion and dilation**

To make it easier to search for the target contour, the image is first subjected to erosion and dilation operations, which smooth out its edges.

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

The function **erode()** is used to perform erosion on the image. The meanings of the parameters enclosed in the parenthesis is as follows.

The first parameter "**mask**" is the input image.

"**cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))**" is the structuring element or kernel that decides the nature of the operation. The parameter in the parenthesis refers to the kernel shape and the second parameter is the kernel size.

The function **dilate()** is applied for image dilation. The meanings of the parameters are the same as **erode().**

◆ **Obtain the largest contour**

Utilize the function "**findContours()**" in cv2 library to search for the maximum contour in target color within the image.

```
contours = cv2.findContours(blob, cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_TC89_L1)[-
2]  # 查找每个区域中的轮廓。
```

The first parameter "**blob**" is the input image.

The second parameter "**cv2.RETR_EXTERNAL**" decides the contour retrieving mode.

The third parameter "**cv2.CHAIN_APPROX_TC89_L1**" specifies the contour approximation method.

7.3 Mark the Image

◆ **Mark the Object**

Adopt the module "**drawContours**" to mark the target object on the live camera feed.

```
cv2.drawContours(result_image, [box], -1, (0, 255, 255), 2)  # 画出四个点组成的
矩形
```

The parameters in the parenthesis is as follows.

The first parameter "**result_image**" is the input image.

The second parameter "**[box]**" is the coordinate of the four corners of the drawn rectangle.

The third parameter "**-1**" represents all the contour can be drawn.

The fourth parameter "**(0, 255, 255)**" is the color of the rectangle.

The fifth parameter “**2**” is the width of the line.

7.4 Perform Action

◆ Match the Recognition Result

Match the corresponding image category name to obtain the X coordinate of the center point of the image. Once the numerical value of the X coordinate falls within the turning range, instruct the robot to initiate the turning process.

```
# 这行代码检查当前检测到的物体是否是右侧物体
                if class_name == 'right':

                    # 计算了右侧物体的面积。它通过使用物体边界框的宽度和高度来计算
                    self.right_area = abs((i.box[0] - i.box[2])*(i.box[1] -
i.box[3]))

# 如果 self.right_area 大于 1500, 则设置 self.turn_right 为 True, 标记需要右转。
                if 1500 < self.right_area:
                    self.turn_right = True
```

◆ Execute corresponding action

When JetAuto recognizes the right turn traffic sign, a topic message will be sent to the driving motors to control the robot's turn to the right.

```
# 发布一个空的 Twist 消息到机器人的控制话题上, 这将停止机器人的所有移动。这是因为默认的
Twist 消息的线性和角速度都是 0。
                            self.mecanum_pub.publish(geo_msg.Twist())

if self.turn_right:
                                    # 表示开始执行右转动作。
                                    self.start_turn = True
                               # 将机器人的角速度设置为 -0.45, 表示向右转动。
                               twist.angular.z = -0.45
```

Set the angular velocity of the robot along the Z-axis to -0.45 rad/s. Negative number represents that the motor will rotate clockwise.

Part 5 Autonomous Parking

1. Program Goal

If the robot recognizes the traffic sign for parking, it will stop moving to the right and enter the parking space.

2. Preparation and Precautions

1) Unfold the map and lay it flat on the ground, making sure there are no obstructions in the way. For detailed instructions on how to position the map, please refer to the "**Place the Map and Assemble Tools (Must Read!!!)**" guide included in the same folder.

2) The road sign used in this section is a model trained by YOLOv5. To learn more about YOLOv5, please refer to **chapter 10**.

3) Please operate in well-lit environment, but avoid direct light rays which will cause JetAuto to incorrectly recognize.

3. Program Logic

The program workflow in this session is identical to that of "**Part 2 Road Sign Detection**".

The source code of this program is located in
/home/jetauto_ws/src/jetauto_example/scripts/self_driving/self_driving.py

4. Steps to Operate

1) Start JetAuto, and then connect it to NoMachine.

2) Open command-line terminal.

3) Enter the command "**sudo systemctl stop start_app_node.service**" and press Enter key to turn off app service.

4) Run the command "**roslaunch jetauto_example self_driving.launch**" to enable autonomous driving service.

5) If you want to terminate the running program, use shortcut "**Ctrl+C**". If the program fails to stop, please try again.

5. Program Outcome

After starting the game, place the robot on the road of the map. When the robot approaches the parking traffic sign, it will use yolov5 model to identify the traffic sign, and then maneuver itself to parking spot.

6. Parameter Analysis

If JetAuto fails to recognize or incorrectly recognizes the road sign of turning right, please adjust the parameters according to the below instructions.

When the robot recognizes the parking traffic sign, it may park in advance or delay. You can adjust the following parameters to set the time when it parks.

```
# park_action 方法通过发布 Twist 消息来控制底盘进行一系列的线性运动，从而实现停车动作。
    def park_action(self):
        twist = geo_msg.Twist() # 创建一个 Twist 消息对象
        twist.linear.x = 0.15 # 将 twist 对象的线速度设置为 0.15，表示向前移动
        self.mecanum_pub.publish(twist) # 发布 Twist 消息，控制底盘进行线性运动，使
车辆向前移动。
        rospy.sleep(0.25/0.15) # 使程序暂停一段时间，确保车辆移动到目标位置。
        self.mecanum_pub.publish(geo_msg.Twist()) # 发布空的 Twist 消息，停止底盘
的运动。
        twist = geo_msg.Twist() # 重新创建一个 Twist 消息对象
        twist.linear.y = -0.15 # 将 twist 对象的线速度设置为-0.15，表示向左移动
        self.mecanum_pub.publish(twist) # 发布 Twist 消息，控制底盘进行线性运动，使
车辆向左移动。
        rospy.sleep(0.25/0.15) # 同样暂停一段时间，确保车辆完成左移动作
        self.mecanum_pub.publish(geo_msg.Twist()) # 发布空的 Twist 消息，停止底盘
的运动。
```

The code “**rospy.sleep(0.25/0.15)**” positioned in the 108 th line is used to adjust the duration of going forward after the robot recognizes the parking traffic sign.

The code “**rospy.sleep(0.25/0.15)**” positioned in the 185 th line is used to enable JetAuto to park as it recognizes the parking traffic sign. When JetAuto goes forward, is is further from the parking space if it spends more time in moving to right.

7. Program Analysis

7.1 Basic Configuration

◆ **Read setting file of color threshold**

Adopt the module “**get_yaml_data**” to obtain the setting file of color threshold.

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

◆ **Subscribe to the camera node**

Obtain the live camera feed through subscribing to the message released by camera node.

```
# 从 ROS 的参数服务器中获取参数-深度摄像头的名称
depth_camera = rospy.get_param('/depth_camera/camera_name', 'camera') # 获取参数

# 创建了一个 ROS 订阅者, 用于接收来自深度摄像头的原始 RGB 图像数据
rospy.Subscriber('/%s/rgb/image_raw'%depth_camera, Image,
self.image_callback)  # 摄像头订阅
```

◆ **Subscribe to yolov5 model detection node**

Invoke the detection model to obtain the detection result through subscribing the message released by yolov5 node.

```
# 创建了一个 ROS 订阅者, 用于接收 YOLOv5 目标检测节点发布的物体检测信息
rospy.Subscriber('/yolov5/object_detect', ObjectsInfo, self.get_object_callback)
```

7.2 Process Image

◆ **Gaussian Blurring**

Employ the function “**GaussianBlur()**” in cv2 library to apply Gaussian filter to process the image in order to reduce the noise in the image.

```
img_blur = cv2.GaussianBlur(img_lab, (3, 3), 3)  # 高斯模糊去噪
```

The first parameter “**img_lab**” is the input image.

The second parameter “**(3, 3)**” refers to the size of the Gaussian convolution kernel. Both the height and width of the convolution kernel must be positive number and odd number.

The third parameter “**3**” is the standard deviation in horizontal direction.

◆ **Perform binaryzation**

Employ the function “**inRange()**” in cv2 library to perform binaryzation on the image.

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

The first parameter “**img_blur**” is the input image.

The second parameter, represented by the code '**tuple(lab_data['lab'][self.target_color]['min'])**', specifies the minimum color threshold.

The third parameter, represented by the code '**tuple(lab_data['lab'][self.target_color]['max'])**', is the maximum color threshold.

The RGB value of a pixel above a certain threshold are set to 1 and values below the threshold are set to 0.

◆ **Perform erosion and dilation**

To make it easier to search for the target contour, the image is first subjected to erosion and dilation operations, which smooth out its edges.

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

The function **erode()** is used to perform erosion on the image. The meanings of the parameters enclosed in the parenthesis is as follows.

The first parameter “**mask**” is the input image.

“**cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))**” is the structuring element or kernel that decides the nature of the operation. The parameter in the parenthesis refers to the kernel shape and the second parameter is the kernel size.

The function **dilate()** is applied for image dilation. The meanings of the parameters are the same as **erode().**

◆ **Obtain the largest contour**

Utilize the function “**findContours()**” in cv2 library to search for the maximum contour in target color within the image.

```
contours = cv2.findContours(blob, cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_TC89_L1)[-
2]  # 查找每个区域中的轮廓。
```

The first parameter “**blob**” is the input image.

The second parameter “**cv2.RETR_EXTERNAL**” decides the contour retrieving mode.

The third parameter “**cv2.CHAIN_APPROX_TC89_L1**” specifies the contour approximation method.

7.3 Mark the Image

◆ **Mark the Object**

Adopt the module “**drawContours**” to mark the target object on the live camera feed.

```
cv2.drawContours(result_image, [box], -1, (0, 255, 255), 2)  # 画出四个点组成的
矩形
```

The parameters in the parenthesis is as follows.

The first parameter “**result_image**” is the input image.

The second parameter “**[box]**” is the coordinate of the four corners of the drawn rectangle.

The third parameter “**-1**” represents all the contour can be drawn.

The fourth parameter “**(0, 255, 255)**” is the color of the rectangle.

The fifth parameter “**2**” is the width of the line.

7.4 Perform Action

◆ **Match the Recognition Result**

```
# 如果物体类别是 'park'
               if class_name == 'park':
                   # 将停车点的中心坐标存储在 self.park_pos 中
                   self.park_pos = self.center

# 检查是否满足停车条件, 并且停车动作未开始。
               if 600 < self.park_pos[0] and 150 < self.park_pos[1] and not
self.start_park:
                   # 发布空的 Twist 消息, 停止底盘的运动。
                   self.mecanum_pub.publish(geo_msg.Twist())

                   # 标记停车动作已开始
                   self.start_park = True
                   # 启动一个新的线程执行停车动作, 避免阻塞主线程。
                   threading.Thread(target=self.park_action).start()
                   # 这样做的好处是, 即使 self.park_action 方法中有耗时的操作, 也不
会阻塞主线程的执行, 提高了程序的并发性和响应速度。

                   # 设置停止标志, 停止底盘运动。
                   self.stop = True
```

To prepare for autonomous parking, match the recognition results to obtain the position of the road sign image. Next, determine if the road sign is in front of the parking space. If the position matches, initiate a thread to call the park_action action group function.

- **Execute corresponding action**

Send the message through the motor driving node “**mecanum_pub**” to drive the motor so as to control the robot to complete parking.

```
# park_action 方法通过发布 Twist 消息来控制底盘进行一系列的线性运动，从而实现停车动作。
    def park_action(self):
        twist = geo_msg.Twist() # 创建一个 Twist 消息对象
        twist.linear.x = 0.15 # 将 twist 对象的线速度设置为 0.15，表示向前移动
        self.mecanum_pub.publish(twist) # 发布 Twist 消息，控制底盘进行线性运动，使
车辆向前移动。
        rospy.sleep(0.25/0.15) # 使程序暂停一段时间，确保车辆移动到目标位置。
        self.mecanum_pub.publish(geo_msg.Twist()) # 发布空的 Twist 消息，停止底盘
的运动。
        twist = geo_msg.Twist() # 重新创建一个 Twist 消息对象
        twist.linear.y = -0.15 # 将 twist 对象的线速度设置为-0.15，表示向左移动
        self.mecanum_pub.publish(twist) # 发布 Twist 消息，控制底盘进行线性运动，使
车辆向左移动。
        rospy.sleep(0.25/0.15) # 同样暂停一段时间，确保车辆完成左移动作
        self.mecanum_pub.publish(geo_msg.Twist()) # 发布空的 Twist 消息，停止底盘
的运动。
```

The linear velocity of the motor along X-axis will be set to 0.15m/s first to allow JetAuto to maneuver to the left side of the parking space.

```
        twist.linear.x = 0.15 # 将 twist 对象的线速度设置为 0.15，表示向前移动
        self.mecanum_pub.publish(twist) # 发布 Twist 消息，控制底盘进行线性运动，使
车辆向前移动。
        rospy.sleep(0.25/0.15) # 使程序暂停一段时间，确保车辆移动到目标位置。
        self.mecanum_pub.publish(geo_msg.Twist()) # 发布空的 Twist 消息，停止底盘
的运动。
```

Next, the linear velocity of the motor along Y-axis will be set to 0.15m/s to allow JetAuto to move right to drive into the parking spot.

```
        twist.linear.y = -0.15 # 将 twist 对象的线速度设置为-0.15，表示向左移动
        self.mecanum_pub.publish(twist) # 发布 Twist 消息，控制底盘进行线性运动，使
车辆向左移动。
        rospy.sleep(0.25/0.15) # 同样暂停一段时间，确保车辆完成左移动作
        self.mecanum_pub.publish(geo_msg.Twist()) # 发布空的 Twist 消息，停止底盘
的运动。
```

Part 6 Integrated Application

1. Program Logic

To go first, obtain the live camera feed, and then perform erosion and dilation on the image.

Next, call the Yolov5 model and compare it with the target image.

Lastly, program the robot to execute the corresponding actions based on the recognition result.

The source code of this program is located in
/home/jetauto_ws/src/jetauto_example/scripts/self_driving/main.py

2. Steps to Operate

1) Start JetAuto, and then connect it to NoMachine.

2) Open command-line terminal.

3) Enter the command "**sudo systemctl stop start_app_node.service**" and press Enter key to turn off app service.

4) Run the command "**roslaunch jetauto_example self_driving.launch**" to enable autonomous driving service.

5) Open a new command-line terminal, and execute the command "**roscd jetauto_example/scripts/self_driving/**" to enter the directory containing the programs.

6) Run the command "**python3 main.py**" to start the related game program.

7) If you want to terminate the running program, use shortcut "**Ctrl+C**". If the program fails to stop, please try again.

3. Program Outcome

◆ **Lane keeping**

After starting the game, place the robot on the road of the map. Then it will recognize the yellow lines. It will move forward if the yellow lines are straight, or turn to stay in the lane if the yellow lines are curved.

◆ **Traffic light recognition**

When it is red light, JetAuto will stop. When it is green light, it will keep going. Once it detects the zebra crossing, it will slow down and go forward.

◆ **Turning Traffic Sign and Parking Sign Recognition**

When recognizing turning right traffic sign, it will turn right then keep going forward. When recognizing parking traffic sign, it will parallel park.

NOTE: JetAuto will perform the above actions in in a continuous cycle.