**Yolov8-车辆测距+前车碰撞预警(追尾预警)+车辆检测识别+车辆跟踪测速(原创算法-毕业设计)**

**前言：**

1、本项目通过yolov8和deepsort实现了一个自动驾驶领域的追尾前车碰撞预警系统，可为一些同学的课设、大作业等提供参考。分别实现了自行车、汽车、摩托车、公交车、卡车的实时目标检测、跟车距离测量、车辆间的相对速度测量、基于人脑反应时间和车辆刹停时间的碰撞预警功能。最终效果如下，红色框代表易发生碰撞追尾的高风险目标，黄色框代表中风险目标，绿色框代表低风险目标。

2、可训练自己的数据集，可以换成yolov8各种版本的权重。

**一、环境配置**

```python

pip install torch==1.7.1+cu110 torchvision==0.8.2+cu110 torchaudio===0.7.2 -f https://download.pytorch.org/whl/torch\_stable.html

pip install -r requirements.txt

```

**二、车辆检测、实时跟踪测速算法及代码解读**

1、主函数各参数含义

使用yolov8s.pt、yolov8m.pt、yolov8l.pt、yolov8x.pt预训练权重均可，也可以使用自己训练好的权重，本项目中调用的是yolov8s.pt。

2、算法实现

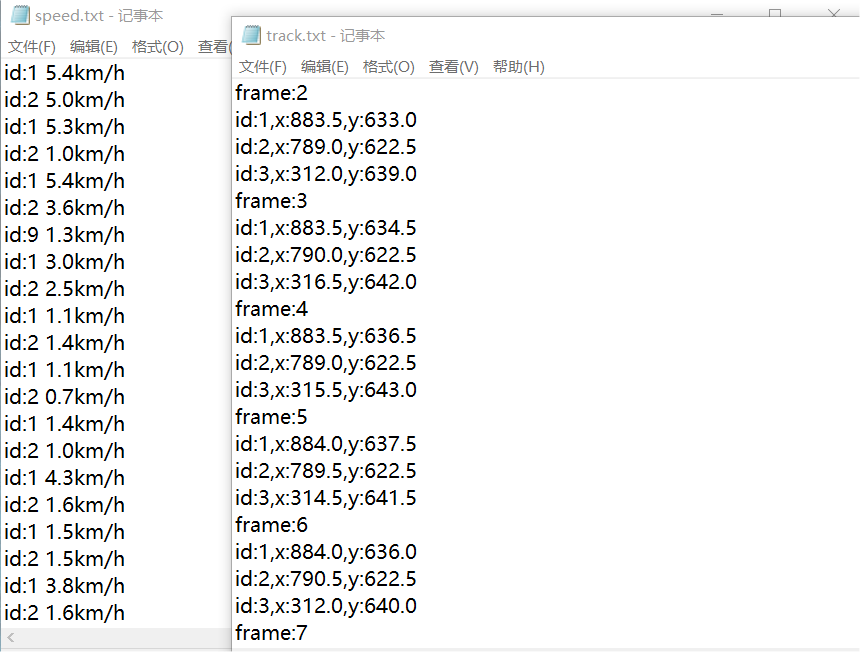
使用yolov8和deepsort分别实现车辆的目标检测、跟踪，再利用检测和跟踪的结果实时计算车速。首先使用提前设定好的车辆真实宽度和检测出来的车辆像素宽度求出真实距离和像素距离的比值，再使用每辆车的前后两帧框的中心坐标计算出两帧之间移动的像素距离。利用这个比值和像素距离做映射，就可以求出两帧之间车辆移动的真实距离。然后距离除以两帧之间的时间，就是速度了。本测速算法中将车辆真实移动距离与像素移动距离看成是线性关系，仅在监控相机轴线与车辆移动方向垂直时才能成立，并且检测出来的车辆框在空间上会产生一定形变，使得真实距离和像素距离的映射关系不准确。有兴趣的同学可以在代码中加入透视变换，将图像变成类似于遥感数据的俯瞰图，实现测速后再将图像变换为原始图像视角。

3、核心代码

我的项目将测速代码封装到了Estimated\_speed()函数里面，有详细注释，调用即可。需要注意的是，由于本项目测试视频为行车记录仪视角所拍摄，拍摄设备本身也在移动，此处测得的车速为车辆之间的相对速度。另外，我的项目中将每辆车的中心坐标轨迹和车速分别写入了根目录下的track.txt和speed.txt，实现了每辆车的速度和轨迹信息记录。

4、效果展示

如图所示，每个目标车辆测出来的速度和行驶轨迹的中心坐标分别存储在两个txt里面，id值用于区分不同的车辆，frame值代表视频的第几帧，x、y分别表示横纵坐标值。



**三、跟车距离测量算法及代码解读**

1、主函数各参数含义

```python

foc = 500.0 # 镜头焦距,单位为cm

real\_hight\_bicycle = 26.04 # 自行车高度，注意单位是英寸

real\_hight\_car = 59.08 # 汽车高度

real\_hight\_motorcycle = 47.24 # 摩托车高度

real\_hight\_bus = 125.98 # 公交车高度

real\_hight\_truck = 137.79 # 卡车高度

# 自定义函数，单目测距

def detect\_distance\_car(h):

dis\_inch = (real\_hight\_car \* foc) / (h - 2)

dis\_cm = dis\_inch \* 2.54

dis\_cm = int(dis\_cm)

dis\_m = dis\_cm/100

return dis\_m

```

2、算法实现

车辆距离计算公式：D = (F\*W)/P，其中D是目标到摄像机的距离(即车辆距离）, Ｆ是相机焦距, W是目标的宽度或者高度, P是指目标在图像中所占据的x方向像素的宽或者y方向像素的高（由Yolov8的目标检测结果可获取）。首先需要设置好镜头焦距，这个参数可以通过在网上查询拍摄设备的参数获取，我这里用的测试视频使用行车记录仪拍摄，焦距为500cm，然后分别设置好自行车、汽车、摩托车、公交车和卡车的实际高度（单位为英寸），利用该公式就能计算出前车距离。本质上就是通过车辆现实尺寸和像素尺寸实现了一个距离映射。

3、效果展示

如图所示，1.6km/h代表这辆车相对拍摄设备行驶的相对速度，car代表目标类别为汽车，0.83为目标的置信度，2.42m为测得的跟车距离。



**四、前车碰撞预警（追尾预警）算法及代码解读**

1、算法流程

首先通过detect.py函数里的time\_person变量设置人脑反应后的刹车时间，单位为s，即人开始反应后踩下刹车到车辆刹停的时间，这个时间与车辆本身的速度有关，后续可通过车机系统接口读取该速度，实现更好的预警效果。这里我们的预设值为3s。

```python

time\_person = 3 # 设置人脑反应后的刹车时间，单位为s，即从人反应后踩下刹车到车辆刹停的时间，这个时间与车辆本身的速度有关，后续可通过车机系统接口读取该速度，实现更好的预警效果

```

再调用plot\_one\_box()函数，将前述变量 time\_person、所测得的车辆目标速度、类别名称等值传入。plot\_one\_box()函数在plots.py中的定义如下，首先根据不同的标签名称调用不同的函数计算跟车距离，再利用测出来的速度和距离计算时间t，与预先设定的人脑反应后的刹车时间time\_person在draw\_speed()函数中进行比较，并返回一个标记值flag。若时间t小于time\_person的1/2，则判定为高风险，并将车辆目标绘制为红色框进行预警；若时间t介于time\_person和time\_person的1/2之间，则判定为低风险，并将车辆目标绘制为黄色框进行预警；若时间t大于time\_person，则并将车辆目标绘制为绿色框，判定为无风险。

```python

def plot\_one\_box(x, img, speed, outputs, time\_person, color=None, label=None, line\_thickness=3, name=None):

# Plots one bounding box on image img

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]))

# w = int(x[2]) - int(x[0]) # 框的宽

h = int(x[3]) - int(x[1]) # 框的高

dis\_m = 1.00

if name == 'bicycle': # 根据标签名称调用不同函数计算距离

dis\_m = detect\_distance\_bicycle(h)

elif name == 'car':

dis\_m = detect\_distance\_car(h)

elif name == 'motorcycle':

dis\_m = detect\_distance\_motorcycle(h)

elif name == 'bus':

dis\_m = detect\_distance\_bus(h)

elif name == 'truck':

dis\_m = detect\_distance\_truck(h)

label += f' {dis\_m}m' # 在标签后追加距离

# 利用测出来的速度和距离计算时间，与预先设定的人脑反应后的刹车时间进行比较，

flag=''

if len(outputs) > 0:

bbox\_xyxy = outputs[:, :4]

identities = outputs[:, -2]

img, flag = draw\_speed(img, speed, bbox\_xyxy, identities, time\_person, dis\_m)

if flag == "High risk": # 根据判定的不同风险等级，绘制不同颜色的目标框，起到预警的作用

cv2.rectangle(img, c1, c2, [0, 0, 255], 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, [0, 0, 255], -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)

elif flag == "Low risk":

cv2.rectangle(img, c1, c2, [0, 215, 255], 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, [0, 215, 255], -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)

else:

cv2.rectangle(img, c1, c2, [48, 128, 20], 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

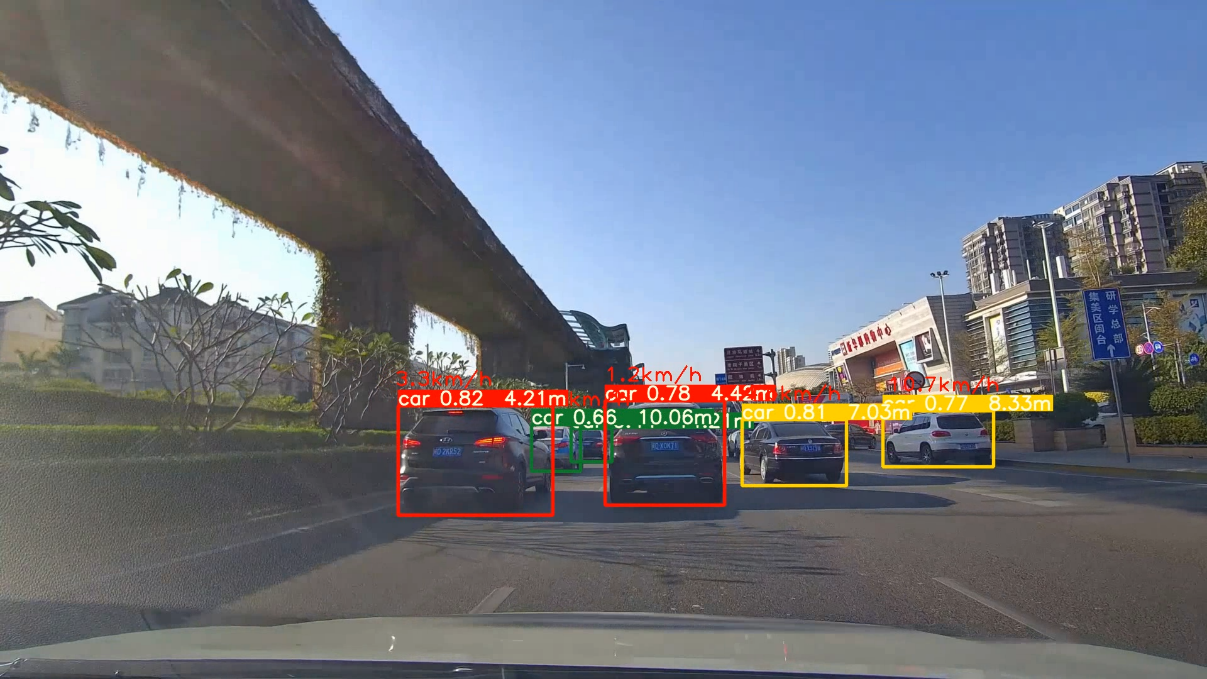
cv2.rectangle(img, c1, c2, [48, 128, 20], -1, cv2.LINE\_AA) # filled

cv2.putText(img, label, (c1[0], c1[1] - 2), 0, tl / 3, [225, 255, 255], thickness=tf, lineType=cv2.LINE\_AA)

```

2、效果展示

如图所示，正前方有四辆车由于跟车距离过近和相对速度过快，触发了系统的预警功能，目标框分别显示为红色和黄色，起到对驾驶员或自动驾驶系统进行提醒的作用。还有目标由于距离过远，对车辆的行车安全不构成威胁，所以显示为绿色框。



**五、总结及源码获取**

1、总结

本项目基于深度目标检测和跟踪技术，结合了一些图像逻辑后处理算法，实现了车辆检测、跟踪、测速、车间距离的测量和前车碰撞预警的功能，检测准确率较高，算法实时性较好，对于自动驾驶车辆的交通安全和环境感知具有一定参考意义和实用价值。

2、项目相关资源

相关参考博客（可用于学习和写论文、课设报告）：

<http://t.csdn.cn/QsSYU>

<http://t.csdn.cn/ySIa4>

<http://t.csdn.cn/ilkPp>

<http://t.csdn.cn/XM4nl>

<http://t.csdn.cn/abdNA>

<http://t.csdn.cn/bGhTV>

本项目效果展示视频：

<https://www.bilibili.com/video/BV14d4y177vE/?spm_id_from=333.999.0.0&vd_source=8c532ded7c7c9041f04e35940d11fdae>

项目数据集下载地址（公开数据集）：https://cocodataset.org/#download