



本科毕业设计 (论文)



题 目: 湖南工商大学学位论文

学生姓名: 全松林

学 号: 2123030033

专 业: 大数据与人工智能

班 级: 数智 2101 班

指导老师: 王海东 讲师

人工智能与先进计算学院

2025 年 5 月

湖南工商大学本科毕业设计诚信声明

本人郑重声明：所呈交的本科毕业设计 面向自动驾驶的视觉目标跟踪和意图分析算法 是本人在指导老师的指导下，独立进行研究工作所取得的成果，成果不存在知识产权争议，除文中已经注明引用的内容外，本设计不含任何其他个人或集体已经发表或撰写过的作品成果。对本设计做出重要贡献的个人和集体均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

作者签名: _____
日 期: _____ 年 月 日

摘要

伴随自动驾驶技术持续发展，加强在复杂交通环境下的感知和决策能力成了重要课题，鉴于传统跟踪与行为预测算法在高密度场景时很难兼顾精度和即时性，于是规划出一套依靠 Carla 仿真平台的视觉目标跟踪与意图分析系统。该系统把 DeepSORT 多目标跟踪算法融合进来，利用卡尔曼滤波和深度外观特征获取办法来达成对动态目标的精准跟踪，而且凭借物理模型创建起运动意图识别体系，通过剖析目标速度，相对位置，可以及时判断诸如“靠近”“远离”“危险靠近”之类的行为趋向。实验结果显示，系统在复杂城市交通场景中有较好的即时性和鲁棒性，特别是针对目标被遮挡和高速移动的情况，这项研究成果给自动驾驶视觉感知中的目标追踪和意图预估给予了新方向，具备一定的工程应用意义和扩展空间。

关键词：自动驾驶； 视觉感知； 目标跟踪； 意图识别； Carla 仿真

ABSTRACT

With the continuous development of autonomous driving technology, enhancing perception and decision-making capabilities in complex traffic environments has become an important issue. Given that traditional tracking and behavior prediction algorithms find it difficult to balance accuracy and immediacy in high-density scenes, a visual object tracking and intent analysis system relying on the Carla simulation platform has been planned. The system integrates the DeepSORT multi-target tracking algorithm and uses Kalman filtering and deep appearance feature acquisition methods to achieve accurate tracking of dynamic targets. It also creates a motion intention recognition system based on a physical model. By analyzing the target speed and relative position, it can timely determine behavioral trends such as "approaching", "moving away", and "dangerous approaching". The experimental results show that the system has good immediacy and robustness in complex urban traffic scenarios, especially for situations where targets are obstructed and moving at high speeds. This research achievement provides a new direction for target tracking and intention estimation in autonomous driving visual perception, and has certain engineering application significance and expansion space.

Key words: Autonomous Driving; Visual Perception; Target Tracking;
Intention Recognition; Carla Simulation

目录

第 1 章 绪论	1
1.1 研究背景	1
1.2 国内外研究现状	1
1.2.1 目标检测研究现状	1
1.2.2 目标跟踪研究现状	3
1.2.3 意图分析研究现状	4
1.3 研究目标与内容	5
1.4 本文结构框架	6
第 2 章 研究理论基础	8
2.1 目标检测与视觉跟踪概述	8
2.2 DeepSORT 算法原理与流程	9
2.2.1 算法结构概述	9
2.2.2 状态预测：卡尔曼滤波器	10
2.2.3 外观建模：深度特征提取	10
2.2.4 匹配决策：融合距离与匈牙利算法	11
2.2.5 算法执行流程	12
2.3 行为意图识别的物理建模方法	13
2.3.1 意图识别的物理基础	13
2.3.2 判别规则设计与分类逻辑	14
2.3.3 与视觉跟踪系统的集成实现	14
2.4 小结	14
第 3 章 系统总体设计	16
3.1 系统架构设计	16
3.2 Carla 仿真平台与传感器配置	17
3.2.1 仿真地图选择与场景设定	17
3.2.2 本车模型与控制机制	18
3.2.3 摄像头传感器配置	19
3.2.4 其他配置说明	19
3.3 项目开发环境与工具链	20
3.3.1 开发硬件平台	20
3.3.2 开发软件与工具	20

第 4 章 视觉目标跟踪模块设计	21
4.1 数据集采集机制与结构设计	21
4.1.1 采集流程设计	21
4.1.2 标注信息结构设计	23
4.2 目标检测与跟踪模型设计	23
4.2.1 目标检测策略设计	23
4.2.2 DeepSORT 跟踪模型集成	24
4.3 模型性能评估	26
4.3.1 系统帧处理时序分析	26
4.3.2 模块耗时对比分析	27
第 5 章 意图分析算法设计与实现	29
5.1 意图识别需求分析	29
5.2 基于速度与距离变化的意图判别逻辑	29
第 6 章 系统功能实现与实验验证	31
6.1 系统整体运行流程	31
6.2 界面展示	32
6.3 用户交互与运行控制	33
第 7 章 总结与展望	35
7.1 工作总结	35
7.2 研究不足	36
7.3 后续优化方向	36
致谢	38
附录 A 附录代码	39
A.1 目标跟踪算法	39
A.2 面向自动驾驶的视觉目标跟踪和意图分析算法	39
参考文献	39

第1章 绪论

1.1 研究背景

随着人工智能、计算机视觉、自动控制等技术极速发展，自动驾驶属于智能交通体系的关键部分，正由理论探究迈向商业化应用，自动驾驶系统一般包含感知，决策，控制这三个模块，而视觉感知处于信息获取的最前端，它的精准度和稳定性会直接影响整个系统的安全性与智能水平，在繁杂的交通场景当中，精确地识别车辆，行人之类的动态物体，持续性地跟进它们，并及时判定这些目标的行为意图，这是做到安全行驶和路线规划的重要前提。

当下，依靠深度学习的目标检测和多目标跟踪算法在学术界和工业界被广泛性采纳，像 YOLO 系列，SORT/DeepSORT 这些颇具代表性的视觉跟踪算法已经具备了比较高的即时性和鲁棒性，但是在实际的交通场景当中，仅仅凭借跟踪得来的信息通常无法满足高级驾驶辅助功能（ADAS）的需求，系统不但要知道“是什么”和“在哪里”，而且还要明白“它打算做什么”，也就是针对前方目标的运动趋向和行为目的作出恰当的判定，进而预先展开风险预估和决策干涉，这样一种依托时空轨迹以及速度信息实施意图识别的本领，会变成未来智能驾驭体系的一项关键能力。

另一方面，由于受到真实交通场景数据收集成本高且难以控制等因素限制，虚拟仿真平台对于智能驾驶算法研发而言非常关键，Carla 作为目前最具代表性的自动驾驶开源仿真平台之一，它给予了极为逼真的三维城市交通场景以及全面的传感器模拟功能，给研究人员赋予了一个可以掌控并重现的试验场所，本项研究依靠 Carla 平台，通过形成起能够运行的目标追踪和意图识别系统，达成从视觉感知到智能判定的整个过程的设计与完成，具备很大的研究价值和实际意义。

综上可得，针对面向自动驾驶的视觉目标跟踪与意图分析展开研究，可促使感知系统由“感知当下”朝着“预测未来”方向发展，而且有益于提升自动驾驶系统的安全多余能力并优化交通行为的合理性，本课题所取得的研究成果会给未来的多目标智能预测，复杂场景认知等方向赋予工程操作根基和技术参照，具备较好的应用前景。

1.2 国内外研究现状

1.2.1 目标检测研究现状

目标检测就是把图像或者视频里的目标从其他不关注的地方分离出来，判定有没有目标，如果有还要找出它们所在的位置并识别出目标类别，这属于计算机视觉范畴内的一项任务，按照是不是用了深度学习作为界限，可以把目标检测算法分成传统目标检测算法和依靠深度学习的目标检测算法，传统目标检测算法要靠手工设计特征算子，采取先获取特征再做分类器这样的模式来达成目的。Viola 和 Jones 提出了 Viola-Jones 人脸检测器 **viola2001rapid**，这个检测器运用 Haar - like 特征获取算子，借助 AdaBoost 分类器执

行分类，而且采用由许多个分类器合成的 Cascade 级联分类器去提升识别率。Navneet Dalal 等人 **dalal2005histograms** 提出方向梯度直方图 (Histogram of Oriented Gradients, HOG)，它属于目标检测中的一种特征描述算子，该方法通过计算图像里局部区域梯度的方向来获取图像浅层的外观特征，不过 HOG 过于依赖局部梯度信息，所以其在复杂场景下的特征获取能力比较差。Felzenszwalb 等人 **felzenszwalb2009object** 以 HOG 为基础，提出了可变形组件模组 (DeformablePartModel, DPM)，DPM 采取多组件策略，把目标拆解成许多部分之后再执行建模，从而使得 DPM 针对目标形变具备较强的鲁棒性。传统目标检测算法大多依靠人工设定的特征获取算子去执行图像特征获取任务，当处于多尺度，密集型目标，极端天气等复杂环境的时候，其鲁棒性和准确率都会大幅下降，而且利用滑动窗口逐个计算图像特征的做法，还致使算法计算复杂度偏高，所以传统目标检测法无法符合即时检测的需求，很难在实际工业生产过程中加以运用。

深度学习快速发展之际，目标检测领域的研究收获了冲破性的进程，同传统目标检测算法比起来，依靠深度学习的目标检测模型历经海量图像数据的训练之后，可以“学会”怎样获取目标特征，当遭遇不同大小的目标以及复杂场景的时候，有着更强的鲁棒性和泛化性能，按照是否须要产生候选区域来分，凭借深度学习的目标检测算法可被划归为单阶段 (one - stage) 和两阶段 (two - stage)，两阶段的目标检测算法大致上分成两步，第一步选出图片里也许会存在物体的候选区域，第二步针对这些候选区域执行特征方面的分类和回归操作。Ross Girshick 等人 **girshick2014rich** 提出 R-CNN 目标检测算法，首先对输入图像使用选择搜索 **uijlings2013selective** (selective search) 网络得到大概 2000 个不一样大小的候选区域，然后把所有候选区域统一缩放，变成 227×227 大小。接着用 CNN 网络对全部候选区域执行特征获取，从而得到维度为 2000×4096 的特征向量，之后利用 SVM 分类器实施分类，并依靠回归器调节边界框的位置，R-CNN 目标检测算法需针对大概 2000 个候选框逐个做特征获取，而且候选框之间会有不少重合部分，具备相同特征，这样便产生诸多多余操作，极大地影响到检测速度。为解决此问题，Ross Girshick 等人 **girshick2015fast** 又提出了 Fast R-CNN，Fast R-CNN 先把输入图片执行卷积，以得到相应的特征图。之后把预先处理好的区域候选框投射到特征图上，这样做只需执行一次卷积运算，从而削减诸多不必要的卷积计算量，不同于 R - CNN 的分类任务 + 回归任务模式，FastR - CNN 把这两个任务合并进同一个网络当中，不必再单独训练分类器和回归器，不过 FastR - CNN 算法仍然保留利用选择搜索算法来得到候选区域的做法，这个过程既耗费时间又很可能会忽略掉包含目标的区域。因此 Shaoqing Ren 等人 **ren2015faster** 又提出了 Faster R-CNN，此算法通过区域建议网络 (RegionProposalNetworks, RPN) 去获取区域候选框，达成了产生候选区域步骤同目标检测任务的融合。FasterR - CNN 同时也提出了锚框 (Anchor) 这一关键思想，所谓锚框就是预先在图像上指定许多组大小比例不一的参照框，以此来尽量涵盖物体可能出现的位置。He Kaiming **he2017mask** 等人在 Faster R-CNN 的根基之上又提出了 MaskR - CNN，这个算法用 ROIAlign 替代 FasterR - CNN 里的 ROI Pooling，ROIAlign 可把任意大小感兴趣区域的特征图划分成固定大小的小特征图，而且利用双线性插值取代了 ROI Pooling 里的直接取整做法，从而化解了

ROI Pooling 由于两次量化而产生的区域不契合 (mis-alignment) 现象。在目标检测任务当中，利用 IOU 阈值去区分正负样本，如果 IOU 阈值更大一些，那么正样本的数量就会更少，相反如果减小 IOU 阈值，则会学到许多无关紧要的特征。Cai 等人 **cai2018cascade** 为了解决上述问题，提出了 Cascade R-CNN，该算法采用级联式的结构，通过对多个感知器使用递增的 IOU 阈值进行训练，来解决因提高 IOU 阈值所引起的模型过拟合问题。

1.2.2 目标跟踪研究现状

多目标跟踪算法可分成依靠检测跟踪 (Tracking by Detection, TBD) 和联合检测跟踪 (Joint Detecting Tracking, JDT) 这两种形式，TBD 目标跟踪算法就是先通过目标检测算法得出含有物体的边界框，然后获取目标的运动信息，外观信息等等，最后用数据关联算法去计算目标之间的亲合力并执行目标关联。Bewley 等人 **bewley2016simple** 提出了 SORT 算法，它属于经典的依靠检测的目标跟踪算法，该算法把 FasterR - CNN 当作检测器，利用卡尔曼滤波来预估并更新物体的运动特征信息，通过匈牙利算法实施数据关联，而且用 IOU 当作度量标准以创建联系，这样就能做到对多个目标的跟踪。Wojke 等人 **wojke2017simple** 提出了 DeepSORT 多目标跟踪算法，此算法在 SORT 的基础上添加级联匹配，用卷积神经网络得到目标外观特征，从而削减面对遮挡时出现的身份切换状况。Yu 等人 **yu2016poi** 提出了 POI 算法，采用 Faster R-CNN 作为检测器，并使用 skip pooling 和 multi-region 两种策略来提高检测精度，利用改进的 GoogLeNet **szegedy2015going** 网络进行特征提取。Sun 等人 **sun2019deep** 提出了深度亲和网络 (DAN)，DAN 网络以端到端的方式对目标物体的外观特征进行学习，通过物体和环境的分层特征计算出在不同帧中目标之间的亲和度。Chen 等人 **chen2018real** 将检测和跟踪结果组成一对候选框，提出了一种得分函数用于衡量每一对候选框的匹配程度，使用非极大值抑制算法依据得分情况进行筛选。Zhang 等人 **zhang2021fairmot** 把目标检测和特征获取融合进同一个网络里做，先利用编码器 - 解码器结构得到图像特征，分流之后，用两个分支各自做边界框预测和目标外观特征获取，再通过预测目标中心点处的特征实施边界框的时序结合。Liang 等人 **liang2022rethinking** 提出 CSTrack 算法，采用 CCN (交叉相关网络) 来改进检测与重识别间的协作学习。将目标检测和外观特征提取进行解耦，通过使用自注意力的方式获得自注意力权重图和交叉相关性权重图，并利用 SAAN **zhao2020saan** (尺度感知注意力网络) 对特征提取网络进行优化。Liang 等人 **liang2022fake** 在 CSTrack 的基础上引入时间信息来修正检测器结果，并提出了重检测网络来重新加载被错误分类的目标。Yu 等人 **yu2022relationtrack** 提出了 GCD (Global Context Disentangling) 模块，该模块能将特征图解耦成检测特征和重识别特征两部分，同时采用可变形注意力机制来学习目标和环境之间的关系。

联合检测跟踪算法是将检测任务与跟踪任务进行合并，仅使用一个网络来实现多目标跟踪任务。Peng 等人 **peng2020chained** 提出了一种 CTracker 算法，该算法首次将目标检测、特征提取、数据关联三个模块集成到单个网络中，实现了端到端的联合检测跟踪。同时还设计了一种联合注意力模块 (JAM, Joint Attention Module) 来突出检测框中的有效信息区域。Xu 等人 **xu2020deep** 提出了 DHN (深度匈牙利算法)，以可微的方式

对检测框和预测框进行匹配，以及提出了一种新的损失函数用于训练联合检测跟踪范式的多目标跟踪器。Pang 等人 **pang2021quasidense** 提出 QDTrack 算法，通过采用多个正负样本同时计算损失的方式来对特征提取网络进行训练，是一种只利用 ReID 特征而不需要位置和运动信息的多目标跟踪方法。Zhou 等人 **zhou2020tracking** 提出了 CenterTrack 算法，使用 CenterNet **zhou2019objects** 目标检测算法来对目标中心进行定位，CenterTrack 依据上一帧的检测结果得到热量图，峰值代表目标中心点，并采用高斯渲染的方法进行模糊处理。模型利用两个额外的并行分支来预测当前目标相对于上一帧时的水平和竖直方向偏移量。Wu 等人 **wu2021track** 提出在线多目标联合检测追踪模型，该模型先通过 CenterNet 得到图像特征，再凭借 CVA 模块算出两帧图像里目标的偏移量，从而得到目标间的关联情况，之后利用 MFW 模块去流传并加强目标特征，最后依靠头部网络把流传过来的特征和当前特征加以处理，以此做到检测和追踪。Wang 等人 **wang2021multiple** 提出了 CorrTracker 算法，利用局部相关模块来构建目标与周围环境之间的拓扑关系，从而加强模型在密集场景中的识别能力。Sun 等人 **sun2020transtrack** 提出 TransTrack 算法，该算法采用 transformer **vaswani2017attention** 架构，利用 Query-Key 机制来跟踪当前帧中已存在的目标以及对新目标进行检测。通过在一次拍摄中完成目标检测和目标关联，建立了一种新的联合检测跟踪范式。Chu 等人 **chu2023transmot** 提出了 STGT 模型，通过将追踪目标轨迹视作稀疏赋权图来构建目标间的空间关系。STGT 构建了一个空间图 transformer 编码器、时间 transformer 编码器和一个空间 transformer 解码器。利用稀疏图来提升训练和推理时的计算速度。并且由于获取了目标间的结构信息，因此比一般的 transformer 也更加有效。

1.2.3 意图分析研究现状

最早关于识别驾驶意图的讨论可以追溯到 Andrew 等人的一项研究 **liu1997realtime**，他们感到能够通过剖析驾驶员的行为动作去预估其意图，在该项研究当中仅仅利用了车辆的动态数据，譬如横摆角，横摆角速度以及车速等来判定驾驶员有无转弯或者变换车道的意图，若想对周边车辆的变换车道意图实施识别，则须要综合运用感知，数据融合，数据处理等诸多技术 **zhang2023highway**。感知和数据融合技术会把全部从传感设备传送过来的信号加以融合，从而产生出周边所有交通情况以及其它环境因素的相关信息，智能车辆会把这些信息从自己的车辆坐标系转变成目标车辆的坐标系，并进一步处理成可供识别算法采用的特征变量，所以除了识别方法之外，特征变量的选取对于识别效果而言同样十分关键，就单个目标车辆来说，当前被广泛性采纳的信号涵盖纵向/横向位置，速度和加速度 **woo2017lane**。主车的意图识别可以通过车内传感器获得大量有效信息，比如方向盘和制动/油门踏板信号，甚至驾驶员本身的状态等。然而，在车联网技术尚未成熟的当下，目标车辆内的这些信息难以获取。Zhang 等人 **zhang2018lane** 使用目标车辆及其四辆邻近交通车辆的信息，包括它们之间的相对速度和距离，来预测换道行为。Leonhardt 等人 **leonhardt2017feature** 也采用了类似的特征变量来研究换道预测。部分研究使用了目标车辆周围的更多相邻车辆以及与每个相邻车辆相关的更多信息（比如状态量的

历史记录) **patel2018predicting**。Altch 等 **altche2017lstm** 甚至考虑了九辆车来提高预测性能。

2016 年, 日产研制出了前沿科技 ProPILOT, 其安装的实力强劲的摄像机可以较为容易的检测出车相对道路的偏离距离 **zhang2016nissan**。雪铁龙 LDWS 系统装有 6 个电子传感器, 当汽车经过车道标记线时, 由于光线反射不同, 检测单元就会把信息传送到车载控制中心, 车内自带的振动马达便会向驾驶员发出警报, 该系统价格便宜, 适应能力较好, 但在复杂电子环境中的表现不佳, 元件精准度有所下降, 现在只有 C5, C6 车型才配备此系统。Google 也投入了相应研发工作, 凭借高效能的软件及智能探测装置, 并融合雷达与摄像技术之后, 就能在各种复杂状况之下全方位探测周边环境 **wang2020autonomous**, 其结合 GPS 技术的车道偏离技术可以及时矫正汽车的行驶轨迹, 并且误差可以达到厘米级。Enache 等 **enache2009driver** 提出了新的计算方法, 通过预瞄偏差及车辆方向偏差综合信息来估计车路之间的相对距离; Cualain 等 **cualain2012automotive** 利用卡尔曼滤波和霍夫变换建立了车道边界的模型, 计算车辆本身的参数及建立新的预警规范来设计预警系统的阈值; Mammar 等 **mammar2006time** 通过计算道路曲率、方向偏差等估计车辆跨过车道线的时间; Ulsoy 等 **chiu1996time** 利用准确度较高的算法估计 TLC 的不确定, 考虑到了不同方面的误差; Gaikwad 等 **gaikwad2015lane** 通过对感兴趣区域的划分, 建立实际车道偏离的度量标准。

1.3 研究目标与内容

本课题意在设计并完成一套依靠视觉感知的自动驾驶目标跟随及意图剖析体系, 利用 Carla 仿真平台创建起可控制的交通环境, 通过整合深度学习目标跟随算法和凭借物理信息的行为推断机制, 达成对诸如车辆, 行人之类的动态交通目标持续不断地辨别, 路线跟随以及运动意图判别, 而且可以在图形界面上即时显现跟随进程和意图剖析成果, 进而给智能驾驭体系给予基本的感知支持。

为实现上述目标, 本课题主要围绕以下几个方面开展具体研究与开发工作:

数据集构建与处理。在 Carla 提供的仿真场景中, 安排好本车并自动生成车流, 利用安装的 RGB 摄像头针对周边交通目标展开图像收集和注解工作, 获取含有图像帧, 目标边框, 目标速度等信息的初始数据, 从而创建起供视觉分析用的多模态数据集, 还要规划出合适于后续分析和训练的标注形式以及存储架构。

视觉目标跟踪模型设计与集成。本系统把依靠深度外观特征加强 DeepSORT 算法当作目标跟踪模型, 通过对视频帧里的检测结果执行轨迹层面的数据关联, 达成对车辆和行人的单独身份标识以及跨帧追踪, 系统允许在单目标跟踪模式时自动挑选“最为靠近本车”的目标实施连续跟踪, 而且会把跟踪成果即时绘制到屏幕图像上。

基于速度信息的意图分析方法设计。在目标跟踪的基础上, 提取目标在世界坐标系下的运动方向与速度大小, 并结合其与本车的相对位置关系, 设计运动意图识别规则体系。采用物理特征量(如点积关系、欧氏距离变化、速度阈值)判断目标是否存在“靠近”“远离”“危险靠近”等行为状态, 并输出中文提示文本以实现可视化展示与预警反馈。

系统集成与仿真测试。将上述模块集成为一个完整的自动驾驶视觉分析系统，结合 Carla 提供的控制接口与传感器 API 实现全过程联动，在 Town10 与 Town01 两个典型城市街景场景中分别进行系统测试与功能验证，评估模型在不同场景下的运行效率与判断准确率，并结合帧率等性能指标进行可行性分析。

本研究着眼于自动驾驶核心任务需求，通过仿真数据收集，视觉追踪，行为分析以及即时显示这四个方面展开工作，塑造起由感知至判断的完备流程，具有较大的应用价值和操作意义。

1.4 本文结构框架

本课题以自动驾驶场景中的视觉感知和行为预测作为核心内容，凭借 Carla 仿真平台所营造出的高真实度交通环境以及 Python 接口功能，利用模块化设计理念塑造起“数据采集—目标跟踪—意图分析—可视化表现”这样一条连贯的工作流，并创建出一个能够实际运作且具备拓展性的目标感知与行为识别系统，整个研究手段包含数据采集与前期处理，目标跟踪算法设计，意图分析部分开发，可视化体现及系统整合这四个重要步骤，其大致流程可由下图表示出来：

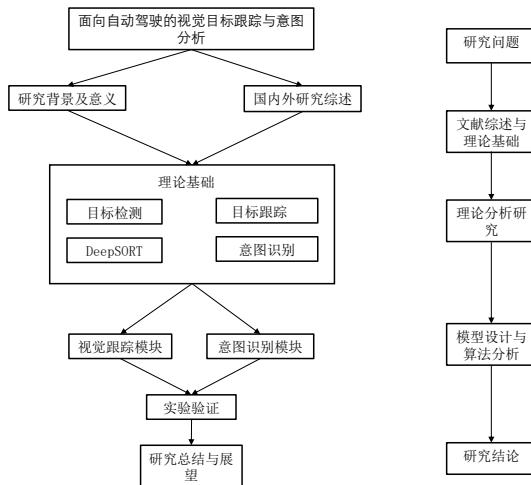


图 1-1 论文结构框架图

在数据采集方面，系统借助 Carla 平台产生典型交通场景，优先选用 Town10 和 Town01 当作试验场所，在车上装配前向 RGB 摄像头来收集图像数据，每一幅图像经过处理之后，可以得到车辆的二维边框，目标 ID，速度大小这些结构化的数据。为了便于后面的训练和分析，系统在运行的时候会自行保留图像帧（.jpg 格式）以及对应的标注文件（.json 格式），其中涵盖边框坐标，目标速度，是不是当前正在追踪的对象等重要信息，从而创建起带有时间顺序特点的感知数据集，保证其具有重现性和可拓展性。

目标跟随时系统的关键所在，本设计融合了 DeepSORT 跟踪算法，该算法把外观特征获取同卡尔曼滤波融合起来，既维持住即时性又保有不错的数据关联水准，通过调用

CarlaAPI，可以马上得到当前场景里全部车辆的三维坐标位置，再加上来自前端图片当中的二维边框信息，就可以做到让目标在图像空间持续被标记出来，为使算法一直关注当下对本车最具潜在交互危险的那个对象，采取“先近后远”的挑选原则来决定要跟踪哪个目标，而且每次只盯住单独一个即可，这样做既能削减运算量又能精简意图剖析的复杂程度。

在行为识别上，系统依靠目标的速度向量，方向角度及其相对于本车的位置，形成起轻量级的意图分析模块。这个模块通过计算目标车辆速度方向与本车的点积关系，融合速度大小和空间距离的动态改变状况，来判定目标的当下行为趋向，其中涵盖“靠近中”“远离中”“危险靠近”“目标稳定”等状态，而且利用阈值合成判断实施分级判断，在实际运作过程中，系统能够及时预测目标行为，并把分析成果用中文文本形式显示在跟踪框上面，从而提升系统的人机交互性和直观程度。

整个系统的主控逻辑存在于 Carla 的同步刷新机制当中，通过 game_loop() 主循环来达成数据获取，图像渲染，目标识别，意图计算以及结果显示等全过程的联动，系统利用 Pygame 实施图像渲染和键盘交互，允许用户对本车行驶执行手动操控，而且会自动保存图像和标签，有益于后续的模型训练和性能考量，总体架构很好地表现出模块化，可检测和可拓展的设计理念，可以有效地支撑不同场景之下的视觉感知和行为分析研究，具备较好的工程化实现根基。

第2章 研究理论基础

本章旨在对本研究所依赖的关键算法与基础理论进行系统阐述，为后续视觉目标跟踪与意图识别系统的实现提供理论支撑。内容包括目标检测与视觉跟踪技术的基本概念，DeepSORT 跟踪算法的原理与工作流程，以及基于速度与空间信息的意图识别模型。同时，为便于理解，还将简要介绍在本研究中承担仿真任务的 Carla 平台的相关原理。

2.1 目标检测与视觉跟踪概述

自动驾驶系统当中，环境感知乃是达成安全行驶与智能决策的根基所在，通过计算机视觉技术来识别路上的车辆、行人、交通标志等重要物体，并持续追踪它们的动态状况，这属于完成环境创建以及行为预估的主要方法，目标探测和视觉跟随属于这里面非常关键的形成单元，会径直左右到自动驾驶系统对于周边环境的认识水平及其决策的精准度。

目标检测（Object Detection）即在输入图像当中找出全部存在的感兴趣目标，并精淮地对这些目标在图像里的所在位置（一般用边界框来体现）以及所属类别实施回归，传统的目标检测算法大多依靠人工指定的特征加上滑动窗口这种机制，代表性的有 Haar 特征以及 HOG + SVM 检测器，此类方法虽然容易做到，但是在面对复杂背景的时候鲁棒性比较差，近些年来，深度学习不断发展起来，依靠卷积神经网络（CNN）的目标检测方法慢慢变成了主流，使得检测的准确率和速度均得到很大改善。当下主流的检测模型大致可归为两类，其一为以 FasterR - CNN 为代表的两阶段检测器，该检测器先产生候选区域而后展开类别判断及边框回归操作，此类检测器精度较高，但速度偏慢。另一类是以 YOLO (YouOnlyLookOnce)，SSD (Single Shot MultiBoxDetector) 等为典型的单阶段检测器，它们直接于特征图上实施回归预测，具备更快的即时性，适宜于像自动驾驶这般对时延较为敏感的场合当中。

与目标检测不同，目标跟踪（Object Tracking）是指在已知初始检测结果的前提下，持续对目标在视频序列中的位置进行估计。根据跟踪目标的数量，通常可分为单目标跟踪（Single Object Tracking, SOT）与多目标跟踪（Multiple Object Tracking, MOT）。单目标跟踪关注于对一个特定目标进行持续跟踪，其主要挑战在于遮挡、快速运动、目标消失与再出现等；而多目标跟踪则需要同时对多个目标进行识别与数据关联，面临着更高的关联复杂性与遮挡问题。在实际的自动驾驶场景中，由于交通参与者种类多、状态变化快、相互干扰强，因此往往需在较短时间内完成高准确率的多目标跟踪任务。

视觉目标跟踪往往把目标检测结果当作输入，依靠匹配机制达成目标的跨帧关联，按照是不是利用外部检测结果，跟踪算法可被划列为两类，其中一类是依托检测的跟踪（Tracking - by - Detection），此类方法先通过检测器得到每帧图像里的目标位置，再凭借轨迹预测和数据关联部件来做到目标编号的一致，这属于当下主流的工程化做法；另一

类则是端到端的跟踪方法，它直接通过时序特征对目标的运动轨迹执行建模，适宜于复杂行为的建模任务，前面那种因为便于部署而且能和已有检测模型相适应，成了不少自动驾驶感知系统的优先选项。

本研究中，为加强系统的即时性与稳定性，采用“检测-跟踪”分离式结构，也就是先通过图像解析获取候选目标的边界框及状态信息，再借助外部跟踪器（DeepSORT）执行跨帧目标关联，这种结构可兼顾深度检测模型的精准度优势以及目标的运动学特点，达成即时，持续，稳定的目标轨迹追踪，而且还加入了单目标跟踪策略，专门关注当下跟自己车辆存在可能发生交互危险的目标，从而优化后面的意图剖析环节的准确性与实用价值。

目标检测与视觉跟踪技术属于自动驾驶系统感知层的关键支撑部分，会给整个系统的安全，即时以及可靠程度带来直接影响，在这个课题当中，这两项技术处于算法链的起始位置，会给后面的意图识别和行为预测给予基本的信息支持。

2.2 DeepSORT 算法原理与流程

在视觉目标跟踪任务当中，传统的多目标跟踪（MOT）算法诸如 SORT（Simple Online and Realtime Tracking），由于其结构较为简单而且处理速度较快，所以被全面地应用到即时系统里面。但是在复杂的交通场景之下，目标相互之间频繁出现遮挡情况，外观相似之处较多，轨迹交叉十分密集等因素，极易引发目标 ID 发生切换或者导致跟踪过程中断，进而影响到系统的稳定性和实用性，为了解决这些问题，Wojke 等人 [45] 在 2017 年的时候提出了 DeepSORT（Deep Simple Online and Realtime Tracking）算法，该算法在保留 SORT 轻量级这一特性的前提之下，采用了外观信息来做进一步的数据关联操作，从各个角度加强了多目标跟踪的鲁棒性及其精准度。

2.2.1 算法结构概述

DeepSORT 属于依靠检测推动的在线目标追踪算法，它的总体架构包含三个主要部分：目标运动建模单元（卡尔曼滤波器），数据关联单元（匈牙利算法加上结合距离度量）以及外观特征获取单元，它是以 SORT（SimpleOnlineandRealtimeTracking）算法为根基加以拓展得来的，SORT 算法利用卡尔曼滤波器和匈牙利算法来达成对目标状态的预估以及数据的关联任务，不过仅仅凭借目标的运动信息实施匹配。DeepSORT 通过采用深度学习模型，依靠目标的外观特征进一步提升了算法的鲁棒性和准确性，它的核心思路在于：每一幅图像从检测器那里得到边界框之后，借助预测和匹配把这些边界框同已有的轨迹联系起来，从而做到目标编号的延续以及轨迹的连贯，这样的结构既关注精度又重视即时性，很适宜应用到像自动驾驶这种对时延较为敏感的系统当中。

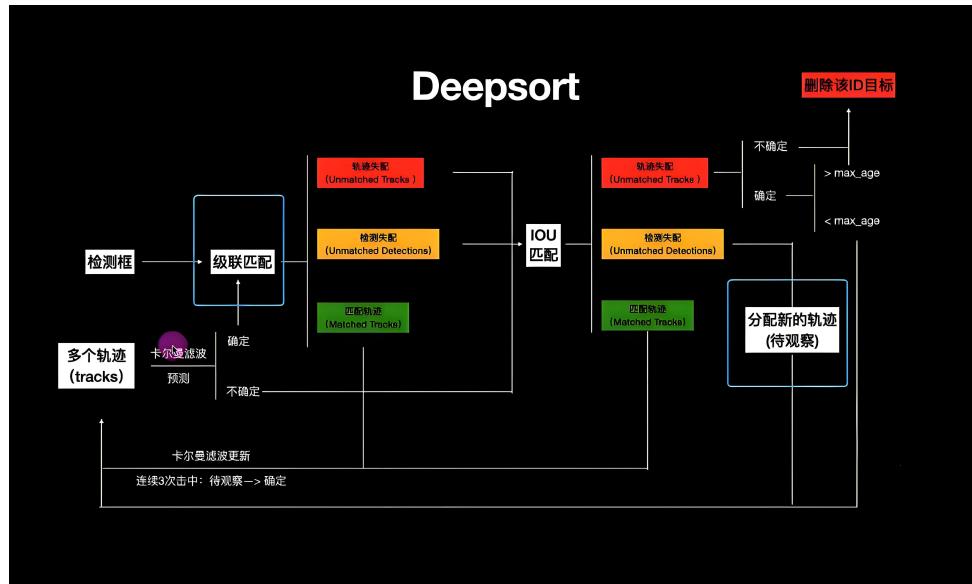


图 2-1 DeepSORT 处理示意图

2.2.2 状态预测：卡尔曼滤波器

DeepSORT 使用卡尔曼滤波器对每个目标进行状态建模与短时位置预测。每个目标的状态向量通常定义为：

$$x = [u, v, \gamma, h, \dot{u}, \dot{v}, \dot{\gamma}, \dot{h}]^T$$

DeepSORT 的第一步是对目标状态执行预测，这要依靠卡尔曼滤波器来完成，卡尔曼滤波器属于一种凭借动态系统实施状态预估的手段，它可以针对目标的运动状况展开平滑处理并予以预估，这里面牵涉到目标所处的位置，移动速度等等方面，在开展目标追踪的时候，卡尔曼滤波器会遵照前一帧所估算出的目标状态，去推测目标在当前帧大概会处于什么地方，这样就能减小因为目标消失或者被遮挡而产生的误差。

卡尔曼滤波器的核心在于利用目标的运动方程执行线性预估，融合测量值来更新目标的状态，在每一个新的时刻点（即每一帧），它会依照前一帧所得到的目标位置及速度去预测本帧当中目标所处的状态，再通过目标被检测到的实际情况对这一预测加以校正，以此维持目标的连续性，填补检测时可能出现的缺失与错误之处。

2.2.3 外观建模：深度特征提取

在多目标跟踪当中，目标的外观特征获取属于保证算法鲁棒性与精准度的关键技术之列，以往的目标跟踪算法大多依靠目标的运动信息（比如位置，速度之类的）来执行跟踪任务，当遇到目标被遮挡，相互重叠或者高速运动这些情况的时候，通常就难以取得良好的成效。DeepSORT 利用深度学习模型，特别就是深度卷积神经网络（CNN），不但可以得到目标的运动信息，而且还能融合其外观特征，从而极大地提升了算法应对复杂场景时的跟踪准确性。

DeepSORT 的外观建模部分依靠卷积神经网络（CNN）来获取目标的视觉特征，这

些特征往往涵盖目标的颜色，形状，纹理等信息，有益于区分不同目标，特别当多个目标相互重叠或者出现短时被遮挡情况的时候，外观特征可有效地应对身份混淆现象。CNN 会针对目标边界框所圈定的区域实施裁剪操作，再把裁剪之后的图像输入到网络当中，从而得到具有固定维度的特征向量，此向量便是目标的外观特征，其体现着目标独有的视觉信息。

通过外观特征获取，DeepSORT 可以在跟随时，对比目标于不同帧里的特征向量，以此辨别并区分各个目标，这个办法大幅优化了算法处于复杂动态环境时的性能，特别是当目标相互交错以及被长时间挡住的时候，外观特征给予了一种连续不断的确信度。

不过，外观特征获取不是毫无难题的，在多目标追踪过程中，目标的外观特征也许会被光照，视角改变或者其他环境要素左右，使得目标的视觉特性产生变动，怎样维持特征的一致性，并且在这种变动情况下执行有效的契合，这是 DeepSORT 碰上的一大难点，对于这个问题，日后的研究能够凭借改良深度网络架构或者融合别的特征获取手段来进一步加强外观建模的稳定性。

总的来说，DeepSORT 利用深度学习的外表特征获取机制，让算法不再仅仅依靠目标的运动信息，而且可以凭借视觉特征有效地辨别目标，从而提升了跟踪的准确性和鲁棒性，这项技术对于动态而繁杂场景下的应用有着重大的现实价值，特别适合于自动驾驶，智能监测之类须要精准多目标跟踪的场合。

2.2.4 匹配决策：融合距离与匈牙利算法

在多目标跟踪任务当中，目标的适配决策属于保障跟踪精准度与一致性的关键环节，DeepSORT 算法把目标的运动信息和表观特点融合起来实施目标的适配，这样就防止了单纯依靠运动信息而产生的身份混乱和丢失现象，适配决策重点在于通过对目标间相似程度加以计算，并利用改良算法譬如匈牙利算法来执行理想适配。

DeepSORT 依靠两种信息去决定目标的契合情况，其一为依托目标位置的欧式距离，其二则是凭借目标外观特征的相似程度，通过这两类信息加以融合之后，DeepSORT 就能更为精准地判定当前帧里的检测框同前一帧中的目标是否存在对应关系，确切来讲，欧式距离可用来度量目标在空间中的位置改变状况，至于外观特征方面的相似程度，则体现出目标在视觉空间里所具有的一致性特点。

欧式距离：多目标跟踪时，目标的位置改变常常体现为运动，欧式距离是衡量目标在图像空间里位置变化的常见尺度，其计算的是目标在两幅连续图像中的位置差别，目标之间的运动距离越小，表示它们的联系就越密切，于是位置上的距离差异便成了目标契合的关键依照。

外观特征的相似度：为了进一步提升匹配的精准度，DeepSORT 会用目标的外观特征做匹配，外观特征通过深度卷积神经网络得到，涵盖目标的颜色，纹理，形状等信息，DeepSORT 凭借算出当前帧中目标的外观特征同前一帧里目标特征的相似程度，可以判定两个目标是不是同一个，哪怕它们所处的位置有所改变。

DeepSORT 当中，目标的契合决策通过算出每一对目标的契合代价来执行，代价矩

阵包含目标间的欧氏距离与外观特征相似度，这两种信息融合成一个全面的契合代价，在此代价矩阵里，每一对目标之间的代价体现出它们相互契合的困难程度，代价越小，表示契合越精准。

DeepSORT 要想从代价矩阵当中选出理想的配对，就采用了匈牙利算法，匈牙利算法属于解决二分图匹配难题的典型算法之列，可以在代价矩阵里找出最小的搭配代价，做到目的的理想结合。通过匈牙利算法，DeepSORT 就能在当下这一帧和前面那一帧之间寻到最为合适的关联，保障目标身份的连贯性，而且把搭配差错缩减到最少程度。

在实际操作当中，匈牙利算法凭借最优化匹配代价的手段，应对数据关联里目标匹配的状况，把目标检测框同已知目标加以对比，找出最为合适的搭配对子，在复杂的动态环境下，该算法可在目标之间创建起精确的对应联系，缩减目标遗失以及身份错乱现象的发生。

2.2.5 算法执行流程

DeepSORT 处理流程包含多个步骤，以下是每帧图像的执行过程：

接收当前帧图像和检测器输出（边界框）：在每一帧里面，DeepSORT 最先会收到当前帧图像和通过目标探测算法 (Yolo, Faster-RCNN 等等) 所输出的边界框，这些边界框里含有各个目标所处的位置，大小和探测到的可信度。

对当前所有轨迹通过卡尔曼滤波器进行状态预测：DeepSORT 针对全部已知目标轨迹，利用卡尔曼滤波器去预估它们的状态，也就是位置与速度，卡尔曼滤波器依照目标在前一帧的位置及速度相关信息，把目标状态加以平滑处理，并对其在当前帧所处的位置予以预测，此步骤有益于填补因为目标丢失或者被遮挡而产生的检测信息空白之处。

对当前检测框提取外观特征向量：DeepSORT 利用预先经过训练的深度卷积神经网络 (CNN) 来获取每一个目标的外观特征，这些外观特征可用来描绘目标的颜色，纹理，形状等视觉方面的信息，依靠这些特征，DeepSORT 就能有效地辨别出不一样的目标，即便当目标彼此之间存在遮挡或者其位置发生很大改变的时候，也仍然可以精准地识别并追踪目标。

构建融合距离矩阵并输入匈牙利算法匹配轨迹与检测：DeepSORT 依靠目标的运动信息以及外观特征来创建融合距离矩阵，这个矩阵会算出每一个检测框同当前跟踪轨迹之间的匹配程度，其中涵盖了依据位置得出的欧式距离，而且包含了依照外观特征算出的相似系数，之后便利用匈牙利算法针对目标实施最优匹配，以保证每个目标在上一帧和当前帧当中可以精准地关联起来。

对匹配成功的轨迹执行更新，未匹配项处理为新轨迹或丢失轨迹：DeepSORT 会更新那些被匹配上的目标的位置，速度之类的信息，而对于没有被匹配上的目标，DeepSORT 按照既定的规则来判定到底是把它当成新轨迹也就是新检测到的目标，还是把它当作丢失了的轨迹，丢失的轨迹会依照卡尔曼滤波器所做的预测一直保留着其自身的状态，直至该目标再次出现或者超过了预先指定好的丢失界限值。

输出每个有效轨迹的编号与位置：DeepSORT 最后会输出每个有效的目标轨迹，包

含目标的编号，位置等信息，各个目标在全部跟踪进程中有个别的 ID，以保证目标身份在所有帧里维持一致。

这个流程于每一幅图像当中各自运行，而且可以即时更新目标的状态，所以它有着较好的在线处理能力，非常适宜用在对时延要求比较高的自动驾驶系统，智能监测之类的即时应用场合之中，通过这样的一连串的步骤之后，DeepSORT 就可以达成对大量目标的精确追踪，即使处于复杂环境之下也能守住较高的即时性能。

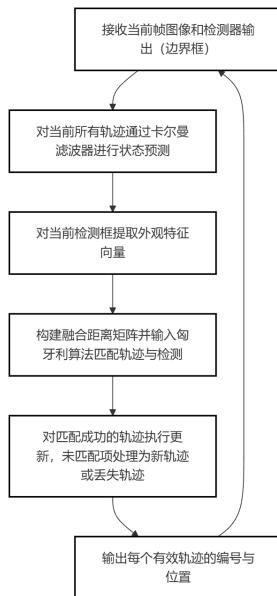


图 2-2 DeepSORT 算法执行流程

2.3 行为意图识别的物理建模方法

自动驾驶系统当中，仅仅具有对环境里物体的“检测”与“跟踪”能力是不够的，想要达成更高层级的决策控制以及路线规划，系统就要去识别周边交通参与者的行意图，也就是要判定这些参与者日后大概会有怎样的运动趋向，以及他们相对于本车存在何种风险关联，这种行为意图方面的分析既关乎驾驶安全，又同车辆的行为产生策略休戚相关，所以被视作联系感知和决策的重要纽带，在本课题里，对于城市交通环境之下常见的车辆交互情形，规划出一套依靠物理状态信息的行为意图判别手段，并形成起一个能够融入到视觉跟踪体系里的轻型化意图剖析单元。

2.3.1 意图识别的物理基础

本研究所采用的行为意图识别方法，基于目标的相对位置变化与瞬时速度信息，构建了一种近似物理建模的分析机制。设定某一时刻跟踪目标的二维图像平面投影中心 (x_t, y_t) ，上一时刻为 (x_{t-1}, y_{t-1}) ，自车屏幕参考点位置为 (x_c, y_c) 。则目标相对位置变化量可定义为：

$$\Delta d = \|(x_t, y_t) - (x_c, y_c)\| - \|(x_{t-1}, y_{t-1}) - (x_c, y_c)\|$$

其中, $| \cdot |$ 表示欧几里得距离。该值用于衡量目标与自车的相对距离变化趋势。结合目标的瞬时速度 v_t , 可初步判断其运动意图。

此外, 为增强分析的动态敏感性, 引入了前后帧距离差分值 Δd 与速度 v_t 的联合判断阈值规则。通过组合判断目标是否正在靠近本车、远离、停留或加速穿越等, 从而实现意图的粗分类。这种方法不依赖于轨迹预测或时序模型, 具有实现简单、计算开销小、适用于实时系统等优点。

2.3.2 判别规则设计与分类逻辑

在本研究系统中, 结合实验场景和工程实现, 设计了以下几种典型的行为状态分类逻辑:

目标靠近中: 当 $\Delta d < -\delta_d$ 且 $v_t > v_{min}$ 时, 表明目标正在快速靠近自车;

目标远离中: 当 $\Delta d > \delta_d$ 且 $v_t > v_{min}$ 时, 目标正在逐渐离开;

危险靠近: 若当前距离改变量 Δd 低于设定阈值 $d_{critical}$ 且速度超过 v_{danger} 时, 则标记为潜在危险行为;

目标稳定: 若目标距离变化缓慢或速度较低, 判断为意图不明或保持状态;

目标初始化中: 用于系统首次观测目标, 未形成完整判断所处状态。

上述分类逻辑采用了硬阈值判定策略, 其参数 δ_d 、 v_{min} 、 v_{danger} 、 $d_{critical}$ 均可根据场景密度或车辆速度进行调节。在实验中, 采用经验法则设定 $\delta_d=5$ 、 $v_{min}=1.5m/s$ 、 $v_{danger}=3.0m/s$ 、 $d_{critical}=-20$, 实现了较高的判别正确率。

2.3.3 与视觉跟踪系统的集成实现

本意图分析模块在执行时嵌于单目标跟踪模块之后, 它会针对被选中目标的时序状态展开分析, 输出当前帧的意图状态表述文本, 而且会在图像上即时显现出来, 该模块还规划了状态缓存机制, 以保障判断结果具有一定的时间连贯性, 免除由于短时检测偏差造成意图颤抖现象发生。

在 Carla 仿真平台的 Town10 与 Town01 场景当中, 本模块对于前方车辆标注“目标远离中”、“目标靠近中”或者是“危险靠近中”的视觉提示, 从而加强了整个系统的交互解释性以及安全回应能力, 给后续的路线规划和驾驶行为产生赋予了语义层面的输入支撑。

2.4 小结

这一章着眼于“视觉目标追踪和行为意图识别”在自动驾驶系统里的主要功能, 详细论述了此课题研究依靠的有关理论和重要技术, 第一, 考虑到当下智能驾驶车辆对于处在动态环境时感知能力的优化需求, 简单总结了目标检测和视觉追踪的发展过程, 剖析了从依靠运动模型的传统追踪手段, 到结合检测器成果和深度特性的当代方法所经历的技术改进, 通过整理视觉感知的整个流程, 清楚表明了视觉追踪在动态交通场景下对目标连贯性, 身份维持以及行为判定起到的无法被取代的意义。

其次，论文着重剖析了本研究里用到的 DeepSORT 算法的主要原理及其完整处理流程，这个算法既保留了传统 SORT 算法速度方面的长处，又加入了外观特征建模机制，从而较好地解决了目标被遮挡，遮蔽之后 re - identification（再识别）失败之类的问题。论文针对 DeepSORT 当中的卡尔曼滤波器状态建模，ReID 特征获取网络，把马氏距离和余弦距离相融合的度量机制，还有轨迹适配及更新策略做了逐个解读，而且阐述了此算法在本系统中的实际操作情况，也就是怎样联系检测成果来达成在线目标轨迹的更新，进而输出稳定且可控制的目标标识编号，给后面的高层语义分析形成根基。

第三部分当中，这一章也详细探究了行为意图识别的物理建模办法，相比依靠深度学习的预测模型或者时序模型，此项研究采取更为轻巧，高效并且适合于即时场景的依靠位置与速度信息的规则建模策略，这个方法把相对距离的改变和目标的运动速度联系起来形成一种复合条件的判别机制，这样就能识别出诸如“靠近”“远离”“危险逼近”这些具有代表性的语义意图。该模块在执行方面被合并到目标追踪逻辑后面，通过接连帧的数据开展物理量运算并配合阈值策略来达成对于动态交互行为的解读与警示，而且，为了改善系统的互动性和直接表现效果，本研究创建了相应的视觉化机制，会把判别结果立刻添加到目标标识框之上，进一步优化了系统针对繁杂交通行为的回应水平及解读水平。

综合而言，这一章站在理论层面上，全面整理了本课题里有关视觉感知，目标追踪以及行为认知这些重要部分，给后面整个系统功能的达成形成稳固的技术根基，下一章会依靠这一章的理论铺垫，细致地阐述系统总体架构的设计想法，包含 Carla 仿真平台怎样设置，自动驾驶控制主要模块如何创建，还有整体工程开发环境怎样营造等方面，从工程方面来促使本研究方案得以落实。

第3章 系统总体设计

3.1 系统架构设计

本课题期望依靠 Carla 仿真平台创建起一套融合了自动驾驶控制，视觉目标检测与跟踪，行为意图识别以及可视化警报功能的智能驾驶实验系统，该系统要能够针对诸如车辆，行人之类的前方交通参与者实施持续性的感知，并对其交互行为加以分析，整个系统会采取模块化的结构形式来保证各个功能部件彼此间具备较好的拓展性与独立性，从而顺应将来可能出现的算法改良和模型更替情况。

本研究所设计的系统架构如图 ?? 所示，主要包含以下六个核心模块：

仿真环境模块（Carla Simulator）：给出高精度的虚拟城市道路，动态交通参与者，本车运动模型等仿真要素，依靠 Carla 给予的 PythonAPI 接口，该系统能够操控自动驾驶汽车在指定地图（Town10 和 Town01）上产生并即时行驶，给后面的感知和控制模块赋予仿真数据输入。

本车驾驶控制模块：通过主控脚本 client_bounding_boxes.py 中 control(self,car) 函数达成，其可接收仿真里关于自车的控制指令（包含加速，制动，转向等），而且具备同传感器同步，处理车辆状态之类的功能，此模块给整个系统赋予了运行的主循环以及控制接口，各个子模块都是挂在这个框架上面的。

图像采集与传感器模块：采用 Carla 中的 RGB 摄像头传感器并挂接到本车前部，用来收集每帧图像数据，其采集到的信息通过回调机制传递到主进程当中，接着对图像实施渲染，保存以及交给下游算法去做进一步处理，而且摄像头的设置参数（包括分辨率，视场角等）能够灵活变动，从而保证符合各类算法所规定的输入需求。

目标检测与跟踪模块：系统里整合了 DeepSORT 即时多目标追踪算法，每一张图片通过外部目标检测器处理之后，其检测框和类别就会被送进 DeepSORT 模型当中，依靠卡尔曼滤波和 ReID 特征关联机制来达成对目标的持续追踪，当下是以“选取最近目标”当作单目标策略，从而保证重点放在自车正前方有可能产生威胁的物体上。

行为意图分析模块：融合追踪目标的相对位置，速度及帧间距变动状况，采用依托物理规则的方法来执行目标行为的类别判定，此模块能够判别目标是否处于“靠近”“远离”或者“危险接近”状态，并给出语义化标签，这个功能具备轻量化特性，而且无需依靠深度模型，可以做到即时反馈与互动控制。

结果可视化与数据存储模块：系统凭借 Pygame 接口把每帧图像，边界框，跟踪 ID 以及行为意图标签及时渲染到显示窗口上，全部数据（图像 + 标签）每隔若干帧便会自动保存到本地目录当中，这对后续的模型训练或者分析重放很有帮助，有着不错的数据采集和重复使用能力。

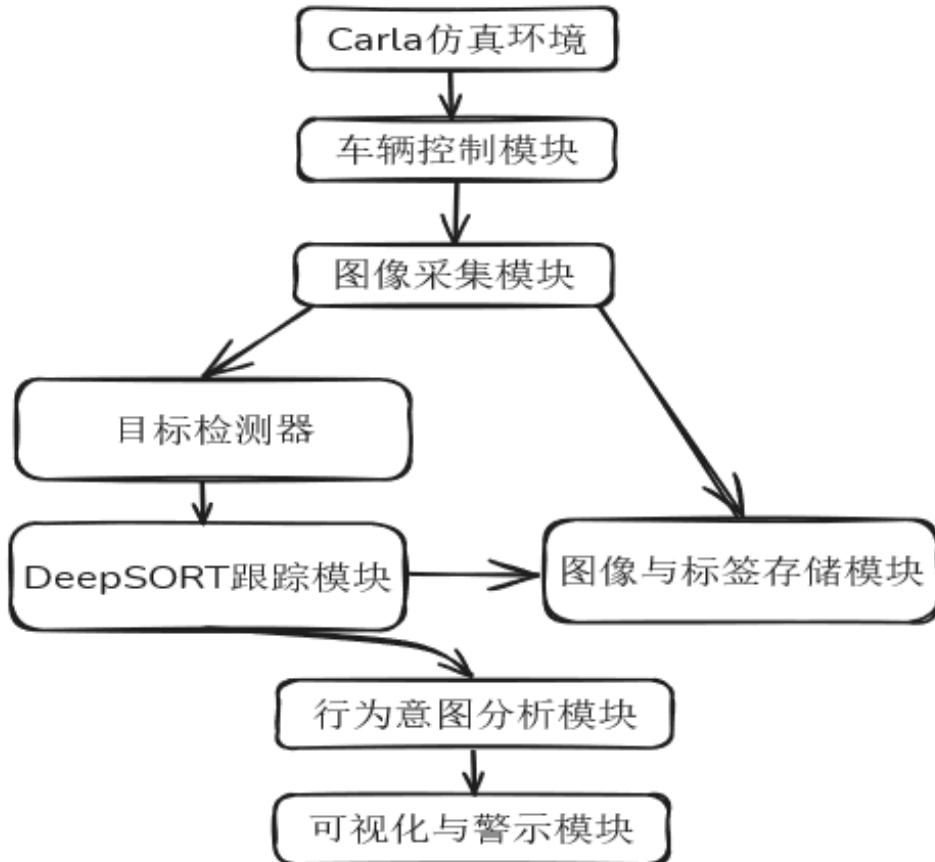


图 3-1 系统架构图

3.2 Carla 仿真平台与传感器配置

要达成本课题里关于自动驾驶车辆的控制，视觉感知以及目标行为意图分析这些功能，系统把 Carla（CarLearningtoAct）当作仿真平台来创建极为逼真的城市交通环境，Carla 是 IntelLabs 和 ComputerVisionCenter 一同研发出来的一款开源自动驾驶仿真平台，它具有高保真度的城市地图，各类传感器模拟器，车辆物理引擎以及交通流经营手段，被全面用在自动驾驶研究方面。

3.2.1 仿真地图选择与场景设定

本系统选择 Carla 自带的两张典型城市地图：Town10HD 和 Town01 作为主要测试场景。

Town10HD 场景包含多车道城市道路、交通信号灯、交叉路口、障碍物遮挡、静态与动态交通参与者等复杂因素，适用于测试系统在高密度交通环境下的感知鲁棒性与行为分析准确性。



图 3-2 Town10HD 地图展示

Town01 场景则结构更为简单，适合用于算法功能验证与对比实验。



图 3-3 Town01 地图展示

仿真平台通过 Python API 方式加载地图并设置交通参与者生成密度与行为逻辑，确保每次启动后均可生成具有真实感的动态交通场景。

3.2.2 本车模型与控制机制

在系统初始化阶段，通过如下语句从 Carla 蓝图库中筛选车辆模型并生成本车：

```
python

car_bp = self.world.get_blueprint_library().filter('vehicle.*')[0]
location = random.choice(self.world.get_map().get_spawn_points())
self.car = self.world.spawn_actor(car_bp, location)
```

图 3-4 控制函数部分代码展示

车辆生成后系统对其施加控制命令，支持手动驾驶（通过键盘方向键操控）或接入后续自动控制模块。控制命令通过 Carla 的 car.apply_control() 接口执行，包含油门、刹车、转向、手刹等基础控制量。

3.2.3 摄像头传感器配置

为获取车辆前方图像信息，系统在本车前部安装一个 RGB 摄像头传感器，其参数配置如下：

表 3-1 摄像头参数配置表

参数	数值
分辨率	960×540 (与系统窗口相适配)
视场角 (FOV)	90°
安装位置	自车后方 5.5 米，高度 2.8 米
俯仰角	-15°，向下略俯视

摄像头配置由 camera_bp.set_attribute() 函数完成，采集的图像数据通过监听函数 self.camera.listen() 注册至主控循环，每帧图像均可进行目标检测、跟踪与意图分析，并实时渲染至用户界面或保存为数据集。

3.2.4 其他配置说明

要想让系统各个模块针对相同时间点的图像帧实施同步处理，本系统便开启 Carla 的同步仿真模式 (`synchronous_mode=True`)，使得每一阶段环境状态的更新都能同传感器输出相对应，进而加强处理过程的可掌控性以及图像帧的稳定性，处于同步模式的时候，系统会按照固定频率 (20FPS) 去调用 `world.tick()` 以推动环境向前发展，这样就能保障传感器输出和控制行为之间存在确切的一一对应关系，有益于维持状态追踪的准确性和延续分析逻辑的连贯性。

而且，系统依靠 pygame 图形界面来做图像渲染工作，用 numpy, cv2, json 这些工具库执行图像处理，数据标注以及数据集形成方面的任务，整个系统要靠 CarlaServer 正常运行（直接在 Windows 下双击 `CarlaUE4.exe` 就能启动），从而保证模拟世界稳定又开放。

3.3 项目开发环境与工具链

为高效实现自动驾驶场景下的视觉目标跟踪与意图识别算法，并完成系统级仿真测试与可视化功能开发，本文构建了一个基于 Carla 仿真平台的完整算法开发与测试环境。本节将对本项目使用的软硬件配置、开发语言、核心依赖库与工具链进行说明。

3.3.1 开发硬件平台

实验采用配置如下表的工作站，满足实时仿真需求：

表 3-2 硬件配置表

组件	规格
CPU	Intel Core i7-10750H @ 2.60GHz
GPU	NVIDIA RTX 2060 (6GB GDDR6)
内存	16GB DDR4
存储	512GB NVMe SSD

3.3.2 开发软件与工具

项目开发主要在 Windows 10 平台下进行，核心依赖环境和工具包括：

表 3-3 软件工具链配置

软件	版本	用途
Python	3.7.9	主控程序开发
Carla	0.9.15	自动驾驶仿真
Pygame	2.5.2	可视化界面渲染与用户交互界面
OpenCV	4.8.0	图像读取与保存，数据集采集处理
deep_sort_realtime	1.3.2	多目标跟踪

其中 CarlaSimulator 的安装与启动采用图形化模式，即双击启动 CarlaUE4.exe 来打开仿真服务端，而客户端代码运行时则是以 TCP 方式通信（默认端口 2000），为保证数据同步性，系统均统一采用 Carla 的同步模式运行，使得传感器输出，车辆控制以及仿真时间能够严格对齐。

Python 环境用 Anaconda 来做管理，创建单独的虚拟环境之后再通过 pip install 去装那些依赖库，这样就能保证版本稳定而且可重现，像 deep_sort_realtime 这种库是从 PyPI 上获取的，或者按照官方 GitHub 上的源码执行安装。

第 4 章 视觉目标跟踪模块设计

4.1 数据集采集机制与结构设计

在视觉目标跟踪与意图分析系统研发时，数据集的形成与管理十分关键，特别对于自动驾驶而言，要想达成精确的模型训练，考量及算法验证，就务必依靠带有真实感的动态交通数据和具备条理化的标签体系。由于受到实际环境数据收集的诸多限制，于是文中凭借 Carla 仿真平台创建起一套自动化的数据搜集与标识机制，通过传感器模拟输出并配合同步控制策略，产生包含图片，速度，位置，身份等信息的优质标注样本库。

4.1.1 采集流程设计

系统每一次执行仿真时，会自动搜集由模拟车载摄像头捕捉到的图像帧，并且立即从当前帧当中识别出其它行驶中的车辆目标，利用 Carla 给出的车辆状态接口，可以得到这些目标的三维坐标位置以及速度大小等数据，再通过转换矩阵把它们映射到摄像头图像所在的平面坐标系里面，从而生成适用于目标探测任务的二维边框形状。

为增强数据的多样性和实用性，系统设计了如下采集流程：

- (1) 图像获取：定期（如每隔 5 帧）从主摄像头传感器中截取 RGB 图像；
- (2) 目标提取：从当前帧中提取所有可见车辆，获取其三维边界框与速度；
- (3) 投影计算：将三维框转换为图像平面坐标，形成二维检测框；
- (4) 追踪标记：判断是否为当前正在追踪的目标，赋予唯一 ID；
- (5) 结果保存：将图像帧以 JPG 格式保存，同时输出 JSON 格式的结构化标签文件。

此流程保证所采集的数据具备较高的时效性且带有完整的标签结构，从而能够应对后续诸如目标检测，跟踪以及行为识别之类的诸多任务需求。

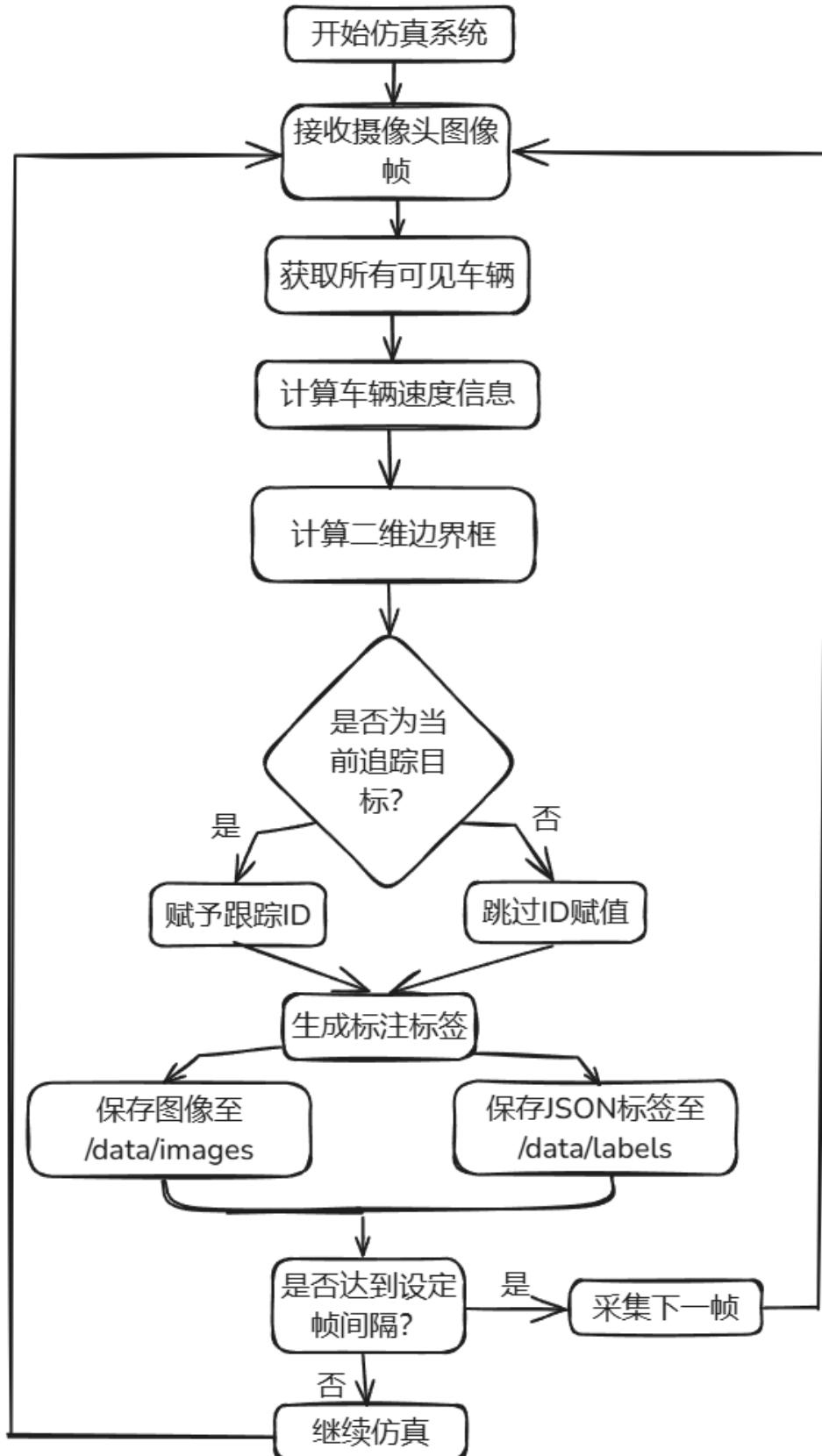


图 4-1 数据采集流程图

4.1.2 标注信息结构设计

每个 JSON 标签文件对应一帧图像，包含该图像中所有检测到的车辆目标。标签信息以列表形式记录，每一项包含如下字段：

bbox: 二维边界框左上角起 4 个顶点的图像坐标；

speed_m_s: 目标瞬时速度（单位为米每秒）；

tracked_id: 若目标被当前追踪模型识别并持续跟踪，则记录其分配 ID，否则为 null。

此结构可容纳常用目标检测框架（诸如 YOLO, FasterR - CNN）以及跟踪框架（诸如 DeepSORT, ByteTrack）所必要的数据格式，而且能够被拓展应用到行为分析任务里的时序建模当中。

```
{  
    "bbox": [  
        [385,175],[393,172],[414,172],[408,175]],  
    "speed_m_s": 5.24,"tracked_id": null  
},  
,  
{  
    "bbox": [  
        [1569,157],[1624,158],[1627,158],[1572,157]  
    ],  
    "speed_m_s": 0.01,"tracked_id": null  
},
```

图 4-2 数据文件结构截图

4.2 目标检测与跟踪模型设计

4.2.1 目标检测策略设计

目标检测处于视觉感知系统的第一环，也是后面目标跟踪和意图分析的根基所在，要想加强整个系统的运行效率和鲁棒性，本文就在仿真环境里采用了一种依靠物理建模的投影式目标检测法，凭借 Carla 平台所供应的三维边界框以及车辆状态信息，避开了常规深度学习模型的推断流程，达成了一种既精准又快速的目标检测方案。

在 Carla 当中，每个交通参与者（涵盖车辆，行人，自行车等）被生成的时候都会带有自身的三维边界框信息（BoundingBox），参照图 ?? 可知，这个边界框由对象的局部中心点，长宽高（extent）以及朝向参数一起确定，准确地体现出目标物体所占据的空间范围。这些边界框是以局部坐标的形式存在的，必要通过变换才可以映射到图像平面上，详细来讲，系统先要得到目标的边界框顶点坐标，接着把它从局部坐标系转换成世界坐标，再依靠摄像头的变换矩阵投射到相机坐标系当中，最后用相机的内参实施二维图像平面的投影，从而形成目标在图像里的二维边界框。



图 4-3 Bounding Box 目标检测边界框示图

此检测策略有不少优点，其一，依靠仿真物理参数而产生高精度检测，不会出现检测误差与误判情况；其二，其运算效率很高，检测时无需执行卷积操作或者获取繁杂的特征，从而大大削减了系统的负担；其三，这种检测结果蕴含着大量语义信息，诸如目标的速度，加速度，位置以及朝向等物理属性，给之后的行为分析给予了数据根基。而且，这种办法具有很强的可拓展性，可以针对不同类别的目标（比如车辆，行人）实施专门化处理并创建标签，还可以通过调整参数来自由把控采样频率和检测精度。

同依靠神经网络的端对端检测法（像 YOLO，FasterR - CNN 之类）比起来，本文所用的检测策略不用预先训练模型，也不要海量样本集，这样就明显减小了模型创建和布置的难度，而且，因为仿真环境具有确定性和可控性，所以这种方法有着较好的可重现性与一致性，可以用在自动驾驶系统前期研发期间的感知部件形成，算法证实以及功能考查等情形下。

总的来说，本文所设计的目标检测模块较好地发挥出 Carla 平台在高保真仿真上的长处，通过对三维边界框实施几何建模并向图像投影，形成起稳定，高效又具扩展性的二维目标检测机制，给后面的目标跟踪和意图识别模块赋予了有力的感知依托。

4.2.2 DeepSORT 跟踪模型集成

目标跟踪重点在于维持同一目标在连续帧里的身份，自动驾驶感知系统所处环境复杂且存在诸多干扰因素，目标也许会由于遮挡，加速，转弯等行为而出现大幅度的位置改变，所以创建起一种稳定又高效的视觉跟踪机制十分关键，依托此，本文借助 DeepSORT 算法塑造起单目标即时跟踪模块，并把它融合进 Carla 仿真平台当中的主控系统代码里面，从而达成了从检测输入一直到轨迹输出这样完整的功能流程。

相比于仅仅依靠位置信息的传统手段，DeepSORT 即便处于遮挡环境之下仍然保留了一定的目标关联能力，本文所采用的是 Python 社区里相对更为成熟的 deep_sort_realtime 版本，其优势在于便于部署，接口清晰且适配性较好。

在整合环节当中，系统会把每帧通过 Carla 投影方式产生的二维检测框当作 Deep-SORT 的输入，先对检测结果执行筛选（去掉那些面积太小，边界不正常的框），然后把它们全都统一格式化成 $[x,y,w,h]$ 形式的矩形框输入列表，连同默认置信度，类别标签一道传给跟踪器。之后，DeepSORT 内部利用卡尔曼滤波器去预估上一帧里全部跟踪目标此刻所处的位置，再融合新帧的检测框以及以往的运动轨迹，通过匈牙利算法来做目标之间的适配工作。在匹配环节当中，DeepSORT 并非仅仅依靠 IOU (Intersection over Union) 来做几何上的重叠度计算，其另外采用了通过卷积神经网络所得到的目标表观特征 (ReID embedding)，从而加强了对那些相似度极高的目标之间的辨别能力。

目标一旦匹配成功，便会被赋予独有的 TrackID，其状态将会不断得到更新，而当检测器识别出未匹配上既有轨迹的新目标时，就会自动生成新的 ID，进而产生一条新的跟踪轨迹。在本文所涉及的系统当中，由于是针对单目标展开跟踪设计的，所以只会把当前帧里距自车最近的那个目标选出来送进跟踪模块，如此一来便可以保障跟踪的唯一性与稳定性，此策略另外精简了系统的复杂程度，还能让后面意图分析环节更为方便地专门应对单个危险对象。

要做到视觉直观反馈，本文针对每帧图像利用 pygame 做即时绘制，把当下正在追踪的目标边界框用显眼的黄色矩形凸显出来，如下图 ?? 所示，并在框上面用文字表现这个目标的 TrackID，系统也凭借颜色编码来区分不同状态，比如初始化，丢失，确认等等，从而优化人机交互的友好程度。

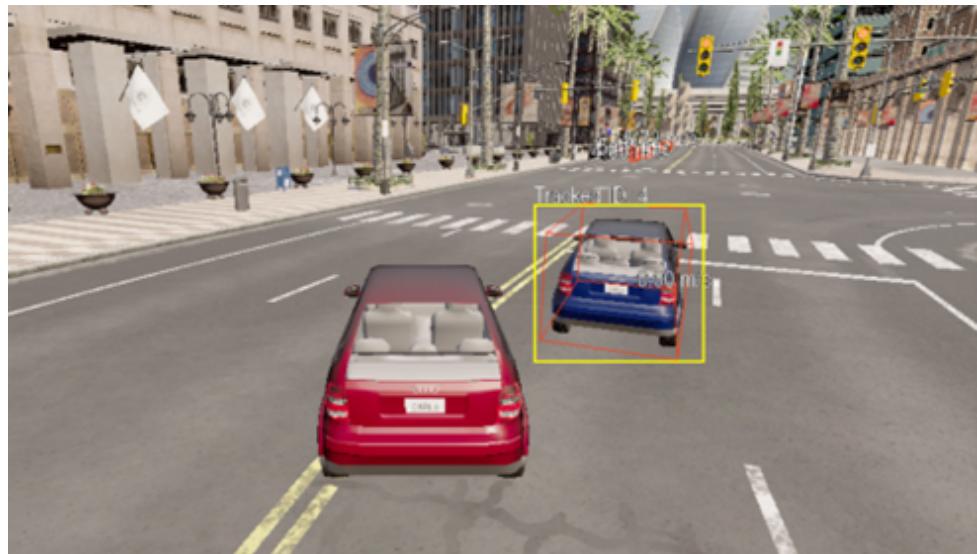


图 4-4 DeepSORT 目标跟踪边界框示图

从整体上看，DeepSORT 模型在本系统中有较好的即时性与跟踪精准度，特别是在交通场景里发生遮挡，快速移动等状况的时候，依然可以守住目标轨迹的持续性和身份的一致性，这个跟踪单元既给后面的行为意图分析供应了稳固的输入根基，又为系统日后向多目标跟踪或者群体行为分析拓展构筑了技术架构根基。

4.3 模型性能评估

为了系统性地评估本文所构建的视觉目标检测与跟踪模块的性能，实验从处理效率与运行稳定性两个维度展开。性能评估基于 Carla 仿真平台，在 Town10 和 Town01 等典型城市市场景中进行，所有实验均在本地 Windows 平台（CPU：Intel i7-10750F，GPU：NVIDIA RTX 2060，内存 16GB）执行，未启用 GPU 加速推理，测试结果具备代表性。

4.3.1 系统帧处理时序分析

图 ?? 展现了系统处理一帧图像数据时完整的流程及其各个阶段所耗费的时间，其中包含从仿真推进开始，通过图像获取，目标检测，目标跟踪，意图推断直到用户界面渲染结束这一整套过程，为精准度量性能，每个模块在主循环里设置了高精度计时器，用以记载时间戳，并算出相邻阶段之间所耗的时间，从而形成起帧处理的时序图。

问题	输出	调试控制台	终端	端口
[Frame Time Summary]				
		world.tick	:	18.95 ms
		render image	:	3.99 ms
		get bbox	:	16.95 ms
		track + intention	:	69.32 ms
		UI display	:	1.00 ms
		total	:	110.21 ms

图 4-5 帧处理时序输出图

将时序输出图转化成对应统计表格分析可知，单帧总处理时间为 110.21 毫秒，系统整体运行帧率约为 8.83 FPS，接近准实时运行性能要求。其中，各子模块平均耗时如下：

表 4-1 模块时序统计表

模块名称	平均耗时	比例估算
场景同步 (tick)	18.95ms	17.2%
图像渲染 (render)	3.99ms	3.6%
边界框生成 (bbox)	16.95ms	15.4%
跟踪与意图分析	69.32ms	62.9%
界面刷新 (UI)	1.00ms	0.9%
总计 (Total)	110.21ms	100%

场景同步 (tick)：平均时耗达到了 5.98ms，其用途在于同 Carla 仿真服务器保持同步，并推动仿真世界向前迈进一帧，这一部分属于系统的基础性操作范畴，时耗较为稳

定，基本上不会受到仿真地图复杂程度以及传感器数量之类因素的左右。

图像渲染 (render): 平均时耗达 3.99ms，其职能在于把图像传感器所传回来的原始 BGRA 数据加以剖析，转换成 RGB 形式，再转变成 pygame 能够渲染的格式，从而交给后面的处理环节去做后续工作，这个阶段的性能比较稳定，属于图像类任务的一种常见开销。

边界框生成 (bbox): 平均时耗达 11.97ms，利用 Carla 自身具备的 3DBoundingBox 投影功能，从真实车辆模型创建顶点坐标，并把这些坐标投影到摄像机图像平面上，这种做法取代了依靠图像的传统目标检测网络，明显减小了计算量，还改良了标注的准确性，给后续的跟踪供应了稳定的输入。

跟踪与意图分析 (track + intention): 这个模块所耗费的时间很多，达到了 90.33ms，大约占据总帧处理时间的 80%，其过程包含 DeepSORT 的轨迹维持和状态更新（卡尔曼滤波，Hungarian 符合，轨迹结合），还有依靠目标运动状态的意图判断逻辑（距离变化率，速度阈值，靠近风险判断等等），这个阶段耗时太多是影响帧率的关键因素，必要着重加以改善。

用户界面刷新 (UI): 平均耗费时间只有 1.00ms，其功能在于把文字信息以及边界框显示到屏幕之上，这部分开销十分微小。

4.3.2 模块耗时对比分析

要想进一步明晰各个模块在资源占用上的相对贡献度，于是把连续 100 帧的平均耗时统计结果制成了柱状图（??），从图上可以看出，DeepSORT 跟踪模块以及意图推断阶段所占比例最大，这就表示，在以后的改良过程当中，应该最先考量针对这个模块展开算法提速或者模型压缩。

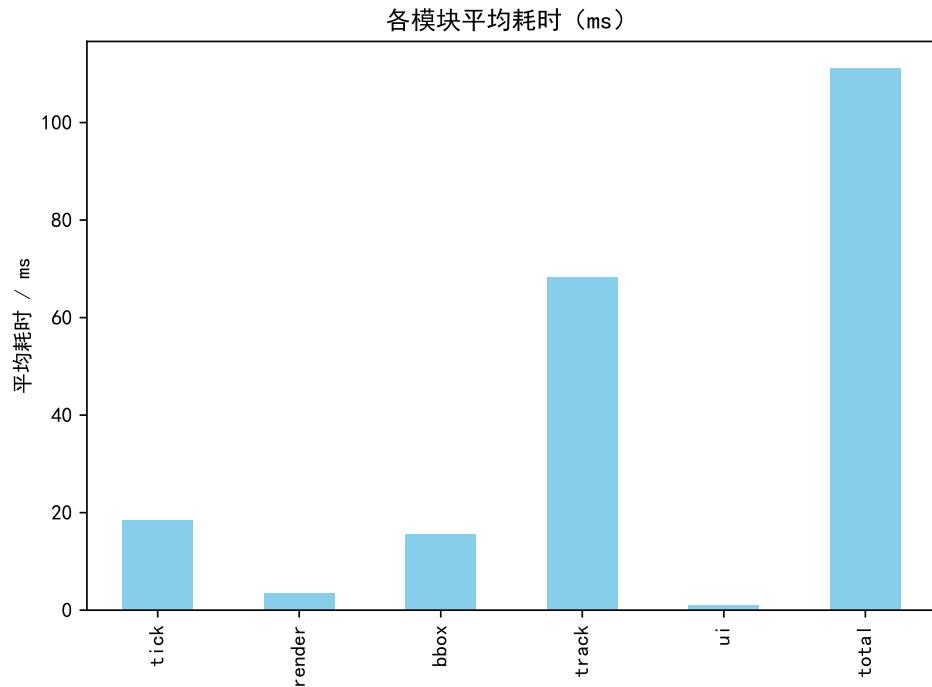


图 4-6 耗时统计柱状图

可以看出，图像渲染、边界框生成这类模块所耗费的时间比较少，这表明用 Carla 平台原本的三维 boundingbox 机制来取代图像检测模型是个有效的办法，既守住了高精度检测输入，又明显缩减了计算资源的占用量，界面渲染这个部分耗费的时间极少，可以在之后执行部署的时候关掉 UI 从而留出更多的处理能力。

第5章 意图分析算法设计与实现

5.1 意图识别需求分析

伴随自动驾驶技术持续向前迈进，车辆针对周围环境的感知需求变得更高，传统的目标检测与跟踪算法即便可以给系统赋予交通参与者的空间位置信息，但如果缺少对目标行为趋势更进一步的认识，那么当面临潜在危险时，系统就无法立即作出反应，在复杂的城市交通场景当中，车辆或者行人不会一直沿着规律的路线移动，其可能会出现诸如加快速度接近，骤然改变车道，径直穿越马路之类具有较高风险的举动，此时仅仅依靠那些静止不动的边框信息远远达不到高级别自动驾驶系统所提出的“先知先觉”的要求，所以说，行为意图的剖析与判定成了塑造智能感知体系必不可少的一部分。

在本系统当中，意图识别模块重点针对被系统持续追踪的目标，凭借它在连续帧之间的速度改变以及同本车的相对距离变动状况，来判定这个目标当下的运动趋向及其潜藏风险等级，通过在图像坐标系下算出目标中心点到视野中心的欧几里得距离，并融合目标本身的线速度，就可以做到对诸如“靠近”“远离”“危险靠近”之类动作的判别，这种依靠物理建模而不是深度学习方式创建起来的行为识别模型，其达成过程较为简易，所需运算量也比较小，可以满足自动驾驶系统对于即时性的要求，而且它并不依靠额外的训练数据，所以具备较好的通用性与拓展能力。

在仿真平台 Carla 所设的 Town10 与 Town01 这两个场景当中，系统借助调用车辆和传感器的同步 API 接口，既能保留图像渲染的即时性，又能得到其他车辆的空间位置及其动态信息。意图识别模块就是把这些基本数据当作输入量，创建起简单而有效的推断规则，针对目标的运动趋向实施及时判断，然后将分析结果用中文文本形式覆盖显示在跟踪框上面，告知“目标正在靠近”“目标渐渐远去”或者“有危险靠近”之类的状况，这样就形成起完整的感知 - 识别 - 反馈循环，进而突出优化系统对于突然发生情况的警报水平和安全保障水平。

意图识别模块一方面补充了系统感知环节中的语义层输出，另一方面也给后面的路径规划及控制逻辑的决策给予了重要参照，它是联系感知和智能决策的关键纽带，对于优化系统整体的智能水平有着重要意义，下一节将会详细论述这个模块的物理建模逻辑以及判别策略。

5.2 基于速度与距离变化的意图判别逻辑

自动驾驶环境下，车辆要随时察觉并认识周边目标（诸如其他车辆，行人）的行为趋向，这样才能及时做出决策及控制反应，若想超越视觉目标跟踪层面进而优化环境感知水准，文章规划并完成了一套依靠速度和距离改变的行为意图判别逻辑部件，该部件可即时判定被跟踪目标针对本车的动向情况，由此给予具备前瞻性的警示作用。该模块的核心思想是：通过连续帧之间的目标相对位置变化（欧氏距离）和当前帧的目标速

度，联合判断其是否存在靠近、远离或危险状态。在具体实现上，系统首先对当前帧目标的边界框进行中心点计算，结合本车视角中心作为参考点，求取目标与本车之间的距离值；随后与上一帧距离进行对比，计算两帧之间的距离变化量（ Δd ），并结合目标当前的瞬时速度（ v ）进行意图分类判断。

为提高判别的精度与稳定性，系统设置了多重判别条件，并赋予合理的速度与距离阈值，判别逻辑详见下表：

表 5-1 意图识别逻辑表

意图类别	判定条件
目标初始化中	当前为首帧，无历史距离
危险靠近	当前帧与本车距离 $d < 150\text{m}$ 且目标速度 $v > 3.0 \text{ m/s}$
目标靠近中	距离变化 $\Delta d < -5\text{m}$ 且 $v > 1.5\text{m/s}$
目标远离中	距离变化 $\Delta d > 5\text{m}$
目标稳定	不满足上述任一条件

上面这些规则依靠单纯的几何物理指标来执行建模，利于开展植入式部署并实施即时计算，而且规避了深度模型对于大量训练样本的需求，规则判别逻辑具备较好的可解读性，有益于后续的守护和改良。

系统运行时，判别结果通过图形化界面及时叠加显现于目标跟踪框之上，而且用中文文本告知用户当下的意图分析结论，譬如“危险靠近”或者“目标正在远离”，此模块同 DeepSORT 跟踪模块紧密结合，保证在维持单目标状态下持续跟踪的情况下，达成对动态意图的判别与输出。

后续工作中，可以把现在这种依靠规则的模块拓展成结合规则和学习的混合模型，通过对历史轨迹执行建模来优化行为预测的效果。

第6章 系统功能实现与实验验证

6.1 系统整体运行流程

本系统依靠 Carla 仿真平台及其 Python 接口实施开发，重点围绕视觉目标检测，跟踪及行为意图识别展开融合工作，从而创建起一套具有即时性与可视化特征的自动驾驶感知子系统，此系统能够在 Carla 环境里达成目标感知，意图推断以及图形界面交互这样一种完整的循环过程，进而给后面的风险决策或者辅助控制供应先验方面的输入信息。

系统以 `client_bounding_boxes.py` 作为主控脚本，它的运作流程包含如下一些重要步骤：

仿真环境初始化：程序先通过 `carla.Client` 和 Carla 仿真服务创建起联系，再把预先指定好的地图场景（诸如 Town10）给加载进来，接着生成代表我方车辆（`egovehicle`）以及周围环境中的交通流，设置好摄像头之类的传感器之后开启同步模式。

实时图像获取与处理：车辆前置 RGB 摄像头持续采集画面帧图，系统凭借 Carla 原生的投影功能从场景当中获取每辆车的 3D 边界框，并把它投影到图像平面上，以此当作检测输入。

目标选择与跟踪处理：系统通过图像中心最近原则选定一个主目标当作跟踪对象，再用 DeepSORT 算法实施跨帧关联，守住稳定的目标 ID。**行为意图识别：**依靠跟踪到的信息，再加上目标速度，运动方向以及中心点位移的改变情况，用物理建模的办法来做行为趋向判断，给出“靠近”“远离”“危险靠近”之类的表述。

可视化与数据记录：系统把全部边界框，目标编号以及意图分析文字及时画到主界面上，用户可以用键盘来操控车辆行驶，而且系统每隔一定帧数就会把图像和标注信息自动存成数据集，以供后面训练和评价时使用。

图像渲染环节利用 Pygame 完成，该部分把图像帧，边界框，速度信息，追踪 ID 以及意图结果等相关元素加以整合，然后一并绘制到显示窗口上，从而生成出清晰易懂的运行界面，为了给后续的模型训练及效果回溯给予支撑，系统内部设置有自动保存图像帧和标注信息的功能，它会按照规定的间隔将图像以及包含边界框和速度信息的 JSON 格式文件存入本地数据集文件夹之中。而且，系统还加入了针对各个模块执行时间的统计功能，每当一帧处理结束之后就会输出各个阶段所花费的时间，当系统运行过一定数量的帧数之后，就会把所有时间段的详细信息整理成一张 CSV 表格并保存下来，方便使用者展开性能方面的分析与改良工作。

通过以上流程的达成，本系统既在仿真环境下做到了从感知到认识直至警报的全过程闭合，又有着较好的可视化表现能力和运行稳定程度，从而给后面章节里的功能演示以及性能评价形成牢靠根基。

6.2 界面展示

系统基于 pygame 实现实时渲染窗口，在 Windows 系统上运行稳定，图像帧率维持在约 8~10 FPS。在主界面中，用户可清晰看到如下信息：

- **前视图图像帧：**显示本车前方道路环境、其他车辆等元素；
- **目标边界框：**每个检测到的车辆被标注为 3D 投影框；
- **跟踪目标编号：**被追踪目标在框上方显示 Tracked ID；
- **意图分析结果：**在跟踪目标框的上方显示当前系统判断的意图（如“危险靠近”、“目标远离中”等）；
- **速度信息显示：**在边界框上标记目标当前速度（单位：m/s）；
- **系统状态输出：**在终端控制台实时输出当前帧耗时、目标状态等信息，辅助调试与评估。

用户能够用键盘来操控本车的行为（比如油门，刹车，转向等），以此考察系统处于各种驾驶状态时的感知鲁棒性，要想验证本系统在 Carla 仿真平台上的运行情况，本节通过一组功能界面截图表现系统的重要功能模块以及动态交互过程，在系统运行期间，各个模块相互协作去执行车辆环境感知，目标筛选，轨迹追踪和行为意图判断，从而创建起稳定的视觉处理及风险提示体系。



图 6-1 意图识别演示图（目标靠近）



图 6-2 意图识别演示图（目标远离）



图 6-3 意图识别演示图（目标稳定）

6.3 用户交互与运行控制

本系统设计时，充分考量到用户的交互体验及控制便利性，通过键盘操作结合即时图像反馈塑造起相对完备的人机交互机制，此系统主要依靠 Pygame 库来达成交互接口，从而让用户得以在仿真运行期间随时操控车辆运动，变更观察视角并查看感知成果。

车辆控制上，用户凭借键盘按键就能达成油门，制动，转向以及手刹这些基本驾驶行为，按一下 W 键就可以让车向前行驶，而 S 键则负责倒车，A/D 键分别用来向左转和向右转，Space 键被设定为手刹功能，ESC 键能随时停止仿真运行，这样做是为了保证系统在用户结束的时候可以平稳地释放资源，关掉所有的传感器和窗口，系统会在每一次的循环里去读入键盘输入，并把它转化成 Carla 的控制指令，以此来做到对车辆持续不断的控制。

在感知交互上，用户通过主界面可直接看到每帧图像里感知与识别的成果，系统会自动画出车辆边界框，而且在被跟踪目标的框体上面显现其 ID 号以及行为意图判断结果，所有信息都是即时刷新的，这有益于用户知晓系统当下的感知状况和意图识别判断，当发生高风险情形（诸如危险靠近）的时候，系统就用红色高亮文字作出警告，以此来提升用户的关注度，进而达成辅助驾驶提醒的目的。

而且，系统具备自动数据采集与保存功能，可以随时把图像帧和标注信息记录到后台，不用人为参与，这个机制有益于之后的离线模型训练和性能考量，加强了系统的拓展性和工程应用价值，用户可以通过设置参数来指定数据采集的频次和储存形式，从而符合各种实验的需求。

本系统在用户交互设计上做到简洁明晰且操作灵活，再加上 Carla 仿真平台稳固的底层支撑，就形成起一个具有真实驾驶控制体验以及智能感知反馈能力的测试平台，可以有效地支撑相关算法的开发验证工作。

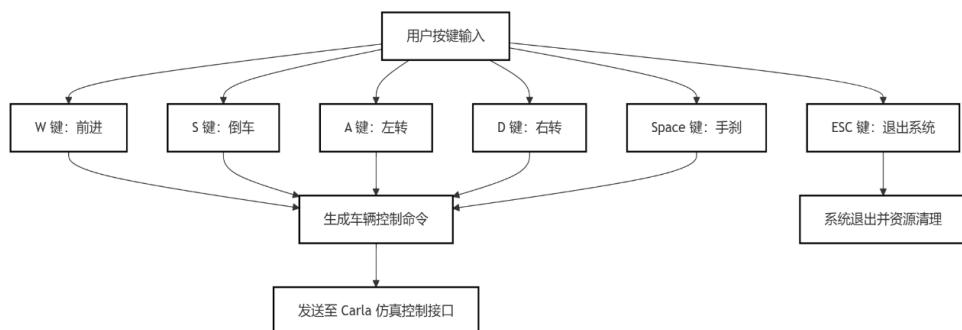


图 6-4 用户键盘控制映射关系示意图

第7章 总结与展望

7.1 工作总结

伴随自动驾驶技术不断发展，怎样加强系统在复杂交通环境下的感知能力和决策智能性成了研究重点，本文通过设计并完成面向自动驾驶的视觉目标跟踪与意图分析系统，并结合 Carla 仿真平台，探究怎样借助目标检测，视觉跟踪以及意图分析等技术来助力自动驾驶系统变得更为安全、智能。

本研究针对“动态交通环境下的目标跟踪”这一问题，采取了 DeepSORT 算法，该算法在传统 SORT 算法之上增添了外观信息，从而加强了系统应对遮挡，快速移动等复杂情况时的鲁棒性，DeepSORT 依靠卡尔曼滤波来预估目标位置，而且融合目标的外观特征实施数据关联，如此一来，即便目标被遮挡或者相互重叠，依然可以完成稳定跟踪，为便于后续展开意图分析与决策处理，本文在跟踪策略方面选取了单目标模式，着重关注当下正与本车交互最为密切的目标，缩减了计算量。

在意图分析上，本文给出了一种依靠物理模型的轻量化方法，通过剖析目标的速度，相对位置以及轨迹改变状况，规划出“靠近”，“远离”和“危险靠近”等行为预估准则，这样一种凭借物理量计算的做法，可以给予即时的行为预先判断，而且规避了深度学习模型也许会产生的计算压力，具有比较高的即时性，符合自动驾驶系统对于时效性的需求。

要保证系统稳定且具备可拓展性，研究依靠 Carla 仿真平台营造出很逼真的城市交通场景，用仿真数据集来做检测，Carla 平台给予了包含道路，交通标志，行人等在内的完备交通环境，从而保障了实验数据的丰富性与真实性，在此基础之上，文章规划出一套数据收集和标识体系，通过 RGB 摄像头及时获取图像数据，而且得到目标物的二维边框，速度之类的信息，给后面的目标探测，追踪以及意图剖析赋予牢靠的数据支撑。

实验结果显示，DeepSORT 算法在复杂交通环境下可维持目标稳定跟踪，免除了目标身份的混淆情况，依靠物理模型的意图分析方法，可以及时判定目标运动趋向，给后续的路线规划和决策给予精确的行为警报，而且，系统在仿真环境中的运行效率得到了证实，可以达成即时跟踪和警报，满足自动驾驶系统对于时效性的需求。

本文的贡献在于给出一种融合视觉感知和意图分析的全新自动驾驶方案，深度学习与物理模型相融合之后，既加强了系统的感知精准度，又改进了在复杂交通情况下的安全预测能力，研究所用到的 Carla 仿真平台给后续的算法改良和系统检测给予了关键的参照，将来，伴随自动驾驶技术不断发展，本文的研究成果有可能拓展至多目标追踪，群体行为分析等更为繁杂的任务当中，从而给智能驾驶系统的高效决策及其安全性赋予更强有力的技术支持。

7.2 研究不足

虽然本文就自动驾驶视觉目标跟踪及意图分析展开的研究收获了一些成果，可还是存在不少不够完善之处，其一，即便 DeepSORT 算法具备应对动态环境里目标被遮挡，高速运动之类状况的能力，但当遭遇极其繁杂的交通场景时，该算法仍旧有可能陷入目标身份错乱或者跟丢目标的困境之中，特别在高密度交通或者目标相互交织的情形之下，既有的算法也许很难在长时间的跟踪进程当中守住足够的鲁棒性，那么，怎样进一步优化 DeepSORT 在复杂场景中的性能，这依旧是个迫切必要解决的课题。

本文所提依靠物理模型的意图识别法可做到快速而精准的行为预估，但这种方法依靠预先指定好的规则及阈值，也许难以处理更为繁杂的驾驶行为，特别是存在大量目标物以及情况错综复杂的时候，伴随驾驶行为变得多种各类，情形也愈发繁杂，仅仅凭借简单的物理规则去做判断大概不能掌握全部有可能出现的意图，日后须要考量采用更多种高效的学习型模型，从而加强系统的智能决策水平。

而且，本文的实验只是局限于 Carla 仿真平台里的数据集，这个平台有着很强的可控性与仿真性，但是不能把真实交通环境里的各种复杂情况完全模仿出来，比如天气改变，光照情况，道路突然发生的状况等等，这也许会引发系统在实际环境当中碰上更多的难题和不确定因素，那么，怎样把仿真环境下得到的研究成果更好地应用到现实环境当中去，还是个必要深入探究的问题。

7.3 后续优化方向

即便本文给出的自动驾驶视觉目标跟踪与意图分析系统在仿真环境中有一些收获，可还是存有不少改良之处，其一，当下所用的 DeepSORT 算法在大部分情况下可做到有效的目标跟踪，但当处于高密度交通以及复杂目标交互情形的时候，却会发生目标身份转换和跟踪中断之类的状况，日后的研究不妨探寻更为优良的多目标跟踪算法，可以融合各类特征或者利用深度学习手段去提升目标跟踪的精准度及其鲁棒性，特别针对那些动态且繁杂的交通状况而言。

在意图分析模块当中，依靠物理规则的分析方法即便可以达成比较精确的行为预估，但它存在一定局限，也就是不能应对更繁杂的驾驶行为，特别当目标之间存在诸多交互的时候，以后的改良途径之一就是利用深度学习模型来做意图预测，格外是凭借深度加强学习之类技术的动态决策模型，可以通过学习改善自身的决策流程，进而更好地解决复杂场景下的不确定因素。

而且，本文的实验和验证大多依靠 Carla 仿真平台来做，这个平台虽然能给予高保真度的仿真环境，可是对比实际道路环境还是会有一些差别，真实环境里的光照改变，天气状况以及交通流的复杂程度，大概会给系统的稳定性和鲁棒性带来更高的需求，所以之后的工作应该把研究成果转为成实际应用，用真实世界的数据展开检测和验证，从而更好地考量系统在实际交通场景中的性能。

通过这些改良之后，将来的系统能够更好地应对动态且复杂的交通状况，进而加强自动驾驶系统的智能感知及决策能力，从而为达成更安全、更智能的自动驾驶技术打下

根基。

致谢

不觉间光阴似箭，我的本科生涯就要落下帷幕，此篇毕业论文也步入尾声，在这段拼搏与成长相伴的旅途中，我一直没有停歇，现在总算有机会在“致谢”这一页，为自己所经之事，所感之情，所想之人写下一些话。

首先我要特别感谢我的论文指导老师王海东老师，非常荣幸能在您的悉心指导下，顺利完成面向自动驾驶的视觉目标跟踪与意图分析这一课题，从开始的选题、制定方案、代码调试、写论文等方面都是在您的细心指导和帮助下才得以顺利完成，在反复的尝试和修改过程中，自己慢慢的沉淀下来，真正体会到做科研的严谨性以及其中的乐趣。

也要借这个机会，向自己过往四年的本科生涯做个回望与道别，刚迈进人工智能与先进计算学院的时候，就怀揣着一种要用技术看清并改变这个世界的憧憬，为了能看得更远一些，除此之外我还辅修了工商管理作为第二专业，那阵子课多事紧，可每次难关也都愈发巩固了继续走下去的决心，偶尔会质疑、会不安、甚至感觉快要撑不住，但只要一想到日后那个自己正站在理想的彼岸朝这边挥手，好像眼前这些付出也就变得值得起来。

我要专门感谢我的家人，在我这四年做出的每个选择以及付出的每份坚持背后，他们都是我最为坚实的依靠，也要感谢那些教过我的老师们，是你们种下了我对知识喜爱的种子，并且让我可以在探寻过程里持续向前迈进，还要感谢我的朋友们与伙伴们，正是因为有你们一直陪在身边，给我带来安慰，聆听我的心声并予以包容，才令我懂得除了学习以外，生活也是需要我们去用心对待的。

以后的路还长着呢，我会带着这四年的积淀与坚守，一直去追寻更远的地方，希望每个认真生活的人都能得偿所愿，祝愿大家平安健康，诸事顺遂。

附录 A 附录代码

A.1 目标跟踪算法

算法 A.1 DeepSORT 目标跟踪算法

```
1: 初始化跟踪器集合  $\mathcal{T}$ 
2: for 每一帧图像 do
3:   检测所有目标，生成检测集合  $\mathcal{D}$ 
4:   提取每个检测框的外观特征向量
5:   根据卡尔曼滤波器预测每个跟踪器的位置
6:   使用匈牙利算法（Hungarian Algorithm）匹配  $\mathcal{D}$  与  $\mathcal{T}$ ，代价函数结合马氏距离和
      外观特征距离
7:   for 每个成功匹配的检测与跟踪器对 do
8:     更新跟踪器状态（位置、速度、外观特征）
9:   end for
10:  for 每个未匹配到检测的跟踪器 do
11:    标记为失配，增加失配计数
12:  end for
13:  for 每个未匹配到跟踪器的检测 do
14:    初始化新的跟踪器
15:  end for
16:  移除失效跟踪器（如失配次数超过最大阈值）
17: end for
```

A.2 面向自动驾驶的视觉目标跟踪和意图分析算法

```
#!/usr/bin/env python

# Copyright (c) 2019 Aptiv
#
# This work is licensed under the terms of the MIT
# license.
# For a copy, see <https://opensource.org/licenses/MIT>.
```

"""

An example of client-side bounding boxes with basic car controls.

Controls :

W : throttle

S : brake

AD : steer

Space : hand-brake

ESC : quit

"""

#

=====

-- find carla module

#

=====

```
import json
import cv2
from datetime import datetime
from deep_sort_realtime.deepsort_tracker import
    DeepSort
```

```
import pathlib
pathlib.Path("data/images").mkdir(parents=True,
    exist_ok=True)
pathlib.Path("data/labels").mkdir(parents=True,
    exist_ok=True)
```

```
import glob
```

```
import os
import sys

try:
    sys.path.append(glob.glob('.. / carla / dist / carla-*%d-%d-%os.egg' % (
        sys.version_info.major,
        sys.version_info.minor,
        'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
except IndexError:
    pass

# =====

# -- imports
# =====

import carla

import weakref
import random

try:
    import pygame
    from pygame.locals import K_ESCAPE
    from pygame.locals import K_SPACE
    from pygame.locals import K_a
    from pygame.locals import K_d
    from pygame.locals import K_s
    from pygame.locals import K_w
```

```
except ImportError:  
    raise RuntimeError('cannot import pygame, make  
    sure pygame package is installed')  
  
try:  
    import numpy as np  
except ImportError:  
    raise RuntimeError('cannot import numpy, make sure  
    numpy package is installed')  
  
VIEW_WIDTH = 1920//2  
VIEW_HEIGHT = 1080//2  
VIEW_FOV = 90  
  
BB_COLOR = (248, 64, 24)  
  
#  
===== # -- ClientSideBoundingBoxes  
#  
===== # 在 ClientSideBoundingBoxes 类中添加一个方法来获取车  
辆的速度并渲染速度文本  
  
class ClientSideBoundingBoxes(object):  
    @staticmethod  
    def get_bounding_boxes(vehicles, camera):  
        """  
        Creates 3D bounding boxes based on carla  
        vehicle list and camera.  
        """  
        bounding_boxes = []
```

```
for vehicle in vehicles:
    bbox = ClientSideBoundingBoxes.
        get_bounding_box(vehicle, camera)
    speed = vehicle.get_velocity()
    speed_magnitude = np.sqrt(speed.x**2 +
        speed.y**2 + speed.z**2) # 计算车辆的速度大小
    bounding_boxes.append((bbox,
        speed_magnitude)) # 返回边界框和速度
# filter objects behind camera
bounding_boxes = [bb for bb in bounding_boxes
    if all(bb[0][:, 2] > 0)]
return bounding_boxes

@staticmethod
def draw_bounding_boxes(display, bounding_boxes):
    """
    Draws bounding boxes on pygame display.
    """
    bb_surface = pygame.Surface((VIEW_WIDTH,
        VIEW_HEIGHT))
    bb_surface.set_colorkey((0, 0, 0))
    chinese_font = pygame.font.Font("C:/Windows/
        Fonts/simhei.ttf", 20)

    for bbox, speed in bounding_boxes:
        points = [(int(bbox[i, 0]), int(bbox[i,
            1])) for i in range(8)] # 绘制边界框
        pygame.draw.line(bb_surface, BB_COLOR,
            points[0], points[1])
        pygame.draw.line(bb_surface, BB_COLOR,
            points[0], points[1])
        pygame.draw.line(bb_surface, BB_COLOR,
            points[1], points[2])
        pygame.draw.line(bb_surface, BB_COLOR,
            points[2], points[3])
```

```
pygame.draw.line(bb_surface, BB_COLOR,
    points[3], points[0])
# top
pygame.draw.line(bb_surface, BB_COLOR,
    points[4], points[5])
pygame.draw.line(bb_surface, BB_COLOR,
    points[5], points[6])
pygame.draw.line(bb_surface, BB_COLOR,
    points[6], points[7])
pygame.draw.line(bb_surface, BB_COLOR,
    points[7], points[4])
# base-top
pygame.draw.line(bb_surface, BB_COLOR,
    points[0], points[4])
pygame.draw.line(bb_surface, BB_COLOR,
    points[1], points[5])
pygame.draw.line(bb_surface, BB_COLOR,
    points[2], points[6])
pygame.draw.line(bb_surface, BB_COLOR,
    points[3], points[7])

# 绘制速度文本
speed_text = f'{speed:.2f} m/s' # 显示速度，保留两位小数
text_surface = chinese_font.render(
    speed_text, True, (255, 255, 255)) # 白色文字
text_rect = text_surface.get_rect(center=(
    int(bbox[0, 0]), int(bbox[0, 1]) - 10))
# 在边界框上方显示
bb_surface.blit(text_surface, text_rect)

display.blit(bb_surface, (0, 0))

@staticmethod
def get_bounding_box(vehicle, camera):
```

```
"""
Returns 3D bounding box for a vehicle based on
camera view.
"""

bb_cords = ClientSideBoundingBoxes.
    _create_bb_points(vehicle)
cords_x_y_z = ClientSideBoundingBoxes.
    _vehicle_to_sensor(bb_cords, vehicle, camera
)[3:, :]
cords_y_minus_z_x = np.concatenate([
    cords_x_y_z[1, :], -cords_x_y_z[2, :],
    cords_x_y_z[0, :]])
bbox = np.transpose(np.dot(camera.calibration,
    cords_y_minus_z_x))
camera_bbox = np.concatenate([bbox[:, 0] /
    bbox[:, 2], bbox[:, 1] / bbox[:, 2], bbox[:, 2],
    bbox[:, 2]], axis=1)
return camera_bbox

@staticmethod
def _create_bb_points(vehicle):
    """
    Returns 3D bounding box for a vehicle.
    """

    cords = np.zeros((8, 4))
    extent = vehicle.bounding_box.extent
    cords[0, :] = np.array([extent.x, extent.y, -
        extent.z, 1])
    cords[1, :] = np.array([-extent.x, extent.y, -
        extent.z, 1])
    cords[2, :] = np.array([-extent.x, -extent.y, -
        extent.z, 1])
    cords[3, :] = np.array([extent.x, -extent.y, -
        extent.z, 1])
    cords[4, :] = np.array([extent.x, extent.y,
```

```
        extent.z, 1])
cords[5, :] = np.array([-extent.x, extent.y,
                        extent.z, 1])
cords[6, :] = np.array([-extent.x, -extent.y,
                        extent.z, 1])
cords[7, :] = np.array([extent.x, -extent.y,
                        extent.z, 1])
return cords

@staticmethod
def _vehicle_to_sensor(cords, vehicle, sensor):
    """
    Transforms coordinates of a vehicle bounding
    box to sensor.
    """

    world_cord = ClientSideBoundingBoxes.
        _vehicle_to_world(cords, vehicle)
    sensor_cord = ClientSideBoundingBoxes.
        _world_to_sensor(world_cord, sensor)
    return sensor_cord

@staticmethod
def _vehicle_to_world(cords, vehicle):
    """
    Transforms coordinates of a vehicle bounding
    box to world.
    """

    bb_transform = carla.Transform(vehicle.
                                  bounding_box.location)
    bb_vehicle_matrix = ClientSideBoundingBoxes.
        get_matrix(bb_transform)
    vehicle_world_matrix = ClientSideBoundingBoxes
        .get_matrix(vehicle.get_transform())
    bb_world_matrix = np.dot(vehicle_world_matrix,
                            bb_vehicle_matrix)
```

```
world_cords = np.dot(bb_world_matrix, np.
    transpose(cords))
return world_cords

@staticmethod
def _world_to_sensor(cords, sensor):
    """
    Transforms world coordinates to sensor.
    """

    sensor_world_matrix = ClientSideBoundingBoxes.
        get_matrix(sensor.get_transform())
    world_sensor_matrix = np.linalg.inv(
        sensor_world_matrix)
    sensor_cords = np.dot(world_sensor_matrix,
        cords)
    return sensor_cords

@staticmethod
def get_matrix(transform):
    """
    Creates matrix from carla transform.
    """

    rotation = transform.rotation
    location = transform.location
    c_y = np.cos(np.radians(rotation.yaw))
    s_y = np.sin(np.radians(rotation.yaw))
    c_r = np.cos(np.radians(rotation.roll))
    s_r = np.sin(np.radians(rotation.roll))
    c_p = np.cos(np.radians(rotation.pitch))
    s_p = np.sin(np.radians(rotation.pitch))
    matrix = np.matrix(np.identity(4))
    matrix[0, 3] = location.x
    matrix[1, 3] = location.y
    matrix[2, 3] = location.z
    matrix[0, 0] = c_p * c_y
```

```

        matrix[0, 1] = c_y * s_p * s_r - s_y * c_r
        matrix[0, 2] = -c_y * s_p * c_r - s_y * s_r
        matrix[1, 0] = s_y * c_p
        matrix[1, 1] = s_y * s_p * s_r + c_y * c_r
        matrix[1, 2] = -s_y * s_p * c_r + c_y * s_r
        matrix[2, 0] = s_p
        matrix[2, 1] = -c_p * s_r
        matrix[2, 2] = c_p * c_r
    return matrix

# =====#
# -- BasicSynchronousClient
# =====#
class BasicSynchronousClient(object):
    """
    Basic implementation of a synchronous client.
    """

    def save_frame_and_labels(self, array,
                             bounding_boxes, frame_idx):
        """
        保存当前帧图像和目标边界框 + 速度信息
        """

        timestamp = datetime.now().strftime('%Y%m%d%H%M%S')
        img_filename = f"frame_{frame_idx}_{timestamp}.jpg"
        json_filename = img_filename.replace('.jpg', '.json')

```

```
# 保存图像
img_path = os.path.join("data/images",
                        img_filename)
cv2.imwrite(img_path, array)

# 保存标注
label_data = []
for bbox, speed in bounding_boxes:
    points = [(int(bbox[i, 0]), int(bbox[i,
                                             1])) for i in range(4)]
    is_tracked = False
    if self.tracking_mode and self.target_id
        is not None:
        x_min = min([p[0] for p in points])
        y_min = min([p[1] for p in points])
        x_max = max([p[0] for p in points])
        y_max = max([p[1] for p in points])
        box_area = (x_max - x_min) * (y_max -
                                       y_min)
        # 简易判定是否与当前追踪目标相符（可改进为 IoU）
        is_tracked = True if box_area > 1000
                           else False

    label_data.append({
        "bbox": points,
        "speed_m_s": round(speed, 2),
        "tracked_id": self.target_id if
                       is_tracked else None
    })

with open(os.path.join("data/labels",
                      json_filename), 'w') as f:
    json.dump(label_data, f, indent=2)

def __init__(self):
```

```

        self.client = None
        self.world = None
        self.camera = None
        self.car = None
        self.display = None
        self.image = None
        self.capture = True
        self.tracker = DeepSort(max_age=15)
        self.target_id = None          # 当前跟踪的目标
                                       ID
        self.tracking_mode = True      # 是否启用追踪
                                       (便于后期控制开关)
        self.prev_distance = None      # 上一帧距离
        self.intent_text = ""         # 当前分析结果

def select_closest_target(self, bounding_boxes):
    if not bounding_boxes:
        print("[TRACK] 没有检测到目标")
        return None
    """
    在所有检测目标中选择最近一个 (速度最快+框最近)
    """
    # 简单使用左上角点距离中心来估算“接近程度”
    center_x = VIEW_WIDTH / 2
    center_y = VIEW_HEIGHT / 2

    min_dist = float('inf')
    closest_bbox = None
    for bbox, speed in bounding_boxes:
        x_coords = bbox[:, 0]
        y_coords = bbox[:, 1]
        x_min, y_min = np.min(x_coords), np.min(
            y_coords)
        dist = np.sqrt((x_min - center_x) ** 2 + (

```

```
        y_min - center_y) ** 2)
    if dist < min_dist:
        min_dist = dist
        closest_bbox = (bbox, speed)
return closest_bbox

def analyze_intention(self, bbox, speed):
    """
    基于边界框位置和速度分析当前意图
    """
    # 当前中心点
    x_coords = bbox[:, 0]
    y_coords = bbox[:, 1]
    x_center = np.mean(x_coords)
    y_center = np.mean(y_coords)
    current_center = (x_center, y_center)

    # 自车视图中心（屏幕正下方）
    car_center = (VIEW_WIDTH / 2, VIEW_HEIGHT)

    # 计算欧氏距离
    distance = np.linalg.norm(np.array(
        current_center) - np.array(car_center))

    if self.prev_distance is None:
        self.prev_distance = distance
        self.intent_text = "目标初始化中"
        return

    delta_d = distance - self.prev_distance
    self.prev_distance = distance

    if delta_d < -5 and speed > 1.5:
        self.intent_text = "目标靠近中"
    elif delta_d > 5:
        self.intent_text = "目标远离中"
```

```
        elif distance < 150 and speed > 3:  
            self.intent_text = "危险靠近"  
        else:  
            self.intent_text = "目标稳定"  
  
  
def camera_blueprint(self):  
    """  
  
    Returns camera blueprint.  
    """  
  
  
    camera_bp = self.world.get_blueprint_library()  
        .find('sensor.camera.rgb')  
    camera_bp.set_attribute('image_size_x', str(  
        VIEW_WIDTH))  
    camera_bp.set_attribute('image_size_y', str(  
        VIEW_HEIGHT))  
    camera_bp.set_attribute('fov', str(VIEW_FOV))  
    return camera_bp  
  
  
def set_synchronous_mode(self, synchronous_mode):  
    """  
  
    Sets synchronous mode.  
    """  
  
  
    settings = self.world.get_settings()  
    settings.synchronous_mode = synchronous_mode  
    self.world.apply_settings(settings)  
  
  
def setup_car(self):  
    """  
  
    Spawns actor-vehicle to be controled.  
    """  
  
  
    car_bp = self.world.get_blueprint_library().  
        filter('vehicle.*')[0]  
    location = random.choice(self.world.get_map()).
```

```
        get_spawn_points())
        self.car = self.world.spawn_actor(car_bp,
                                         location)

def setup_camera(self):
    """
Spawns actor-camera to be used to render view.
Sets calibration for client-side boxes
rendering.
    """

    camera_transform = carla.Transform(carla.Location(x=-5.5, z=2.8), carla.Rotation(
        pitch=-15))
    self.camera = self.world.spawn_actor(self.
                                         camera_blueprint(), camera_transform,
                                         attach_to=self.car)
    weak_self = weakref.ref(self)
    self.camera.listen(lambda image: weak_self().set_image(weak_self, image))

    calibration = np.identity(3)
    calibration[0, 2] = VIEW_WIDTH / 2.0
    calibration[1, 2] = VIEW_HEIGHT / 2.0
    calibration[0, 0] = calibration[1, 1] =
        VIEW_WIDTH / (2.0 * np.tan(VIEW_FOV * np.pi
                                    / 360.0))
    self.camera.calibration = calibration

def control(self, car):
    """
Applies control to main car based on pygame
pressed keys.
Will return True If ESCAPE is hit, otherwise
False to end main loop.
    """

```

```
keys = pygame.key.get_pressed()
if keys[K_ESCAPE]:
    return True

control = car.get_control()
control.throttle = 0
if keys[K_w]:
    control.throttle = 1
    control.reverse = False
elif keys[K_s]:
    control.throttle = 1
    control.reverse = True
if keys[K_a]:
    control.steer = max(-1., min(control.steer
        - 0.05, 0)))
elif keys[K_d]:
    control.steer = min(1., max(control.steer
        + 0.05, 0)))
else:
    control.steer = 0
control.hand_brake = keys[K_SPACE]

car.apply_control(control)
return False

@staticmethod
def set_image(weak_self, img):
    """
    Sets image coming from camera sensor.
    The self.capture flag is a mean of
    synchronization - once the flag is
    set, next coming image will be stored.
    """
    self = weak_self()
    if self.capture:
        self.image = img
```

```
self.capture = False

def render(self, display):
    """
    渲染图像并返回 RGB 数组（用于保存）
    """
    if self.image is not None:
        array = np.frombuffer(self.image.raw_data,
                              dtype=np.dtype("uint8"))
        array = np.reshape(array, (self.image.
                                  height, self.image.width, 4))
        rgb_array = array[:, :, :3] # RGB, 不做反转
        bgr_array = rgb_array[:, :, ::-1] # BGR
        for pygame

            surface = pygame.surfarray.make_surface(
                bgr_array.swapaxes(0, 1))
            display.blit(surface, (0, 0))
        return rgb_array # 返回原始 RGB 图像用于保存
    return None

def game_loop(self):
    """
    Main program loop.
    """

    try:
        pygame.init()

        self.client = carla.Client('127.0.0.1',
                                   2000)
        self.client.set_timeout(2.0)
        self.world = self.client.get_world()
```

```
self.setup_car()
self.setup_camera()

self.display = pygame.display.set_mode((
    VIEW_WIDTH, VIEW_HEIGHT), pygame.
    HWSURFACE | pygame.DOUBLEBUF)
pygame_clock = pygame.time.Clock()

self.set_synchronous_mode(True)
vehicles = [v for v in self.world.
    get_actors().filter('vehicle.*') if v.id
    != self.car.id]

frame_count = 0 # 添加在循环开始前的计数
器
while True:
    self.world.tick()
    self.capture = True
    pygame_clock.tick_busy_loop(20)

    frame_count += 1
    rgb_array = self.render(self.display)

    bounding_boxes =
        ClientSideBoundingBoxes.
        get_bounding_boxes(vehicles, self.
            camera)
    ClientSideBoundingBoxes.
        draw_bounding_boxes(self.display,
            bounding_boxes)
    # ----- 单目标追踪逻辑 -----
    target_input = None
    if self.tracking_mode:
        closest = self.
            select_closest_target()
```

```
        bounding_boxes)
    if closest:
        bbox, speed = closest
        x_coords = bbox[:, 0]
        y_coords = bbox[:, 1]
        x_min, y_min = np.min(x_coords),
                        np.min(y_coords)
        x_max, y_max = np.max(x_coords),
                        np.max(y_coords)
        width = x_max - x_min
        height = y_max - y_min

        # 避免空框
        if width >= 10 and height >=
            10:
            target_input = [[x_min,
                            y_min, width, height],
                            0.9, "vehicle")]

# 用 DeepSort 追踪目标

track_id = None
if target_input:
    tracks = self.tracker.
        update_tracks(target_input,
                      frame=rgb_array)

bb_surface = pygame.Surface((
    VIEW_WIDTH, VIEW_HEIGHT))
bb_surface.set_colorkey((0, 0, 0))
font = pygame.font.SysFont("Arial",
                           20)

for track in tracks:
    if not track.is_confirmed():
        continue
    track_id = track.track_id
```

```
l, t, r, b = track.to_ltrb()
pygame.draw.rect(bb_surface,
(255, 255, 0), pygame.Rect(l,
, t, r - 1, b - t), 2)
text_surface = font.render(f"""
    Tracked ID: {track_id}""",
True, (255, 255, 255))
bb_surface.blit(text_surface,
(int(l), int(t) - 20))
# 分析意图（基于 closest）
if closest:
    bbox, speed = closest
    self.analyze_intention(
        bbox, speed)

chinese_font = pygame.font.
Font("C:/Windows/Fonts/
simhei.ttf", 20)
intent_color = {
    "危险靠近": (255, 0, 0),
    "目标靠近中": (255, 128,
0),
    "目标远离中": (0, 255, 0),
    "目标稳定": (200, 200,
200),
    "目标初始化中": (150, 150,
150)
}.get(self.intent_text, (255,
255, 255))
intent_surface = chinese_font.
render(self.intent_text,
True, intent_color)
bb_surface.blit(intent_surface
, (int(l), int(t) - 40))
```

```
        self.display.blit(bb_surface, (0,
                                         0))

        self.target_id = track_id
# ----- end -----

# 每 5 帧保存一次
if frame_count % 5 == 0 and rgb_array
    is not None:
    self.save_frame_and_labels(
        rgb_array, bounding_boxes,
        frame_count)

pygame.display.flip()
pygame.event.pump()
if self.control(self.car):
    return

finally:
    self.set_synchronous_mode(False)
    self.camera.destroy()
    self.car.destroy()
    pygame.quit()

#
=====

# -- main()
-----



#
=====
```

```
def main():
    """
    Initializes the client-side bounding box demo.
    """

    try:
        client = BasicSynchronousClient()
        client.game_loop()
    finally:
        print('EXIT')

if __name__ == '__main__':
    main()
```