Environment Modeling, Action Classification, and Control for Urban Automated Driving

by

Rowan Dempster

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2022

Abstract

This thesis discusses the design and implementation of WATonomous' Automated Driving Stack (ADS), which is capable of performing robo-taxi services in specific operational domains when deployed to WATonomous' research vehicle (Bolty). Three ADS modules are discussed in detail: (1) mapping, environment modeling, and behavioral planning, (2) action classification in video streams, and (3) trajectory planning and control. Additionally, the software architecture within which the ADS is developed and deployed, and the ADS data pipeline itself, are outlined.

The thesis begins with preliminaries on WATonomous' Dockerized software architecture (coined *watod*) which runs and orchestrates the communication of the ADS modules. The *watod* ecosystem, due to its Dockerized and cloud-based design, enables rapid prototyping of new software modules, rapid onboarding of new team members, and parallel execution of many ADS development instances on the WATonomous server cluster's Virtual Machine (VM)s. Cloud-based CARLA simulation development of the ADS and deployment to the Bolty research vehicle are also encapsulated in and facilitated by the *watod* ecosystem. The ADS can be developed in simulation and deployed to the physical research vehicle without modifications to the ADS modules due to the replication of the physical platform in the Carla ROS Bridge sensor configuration. The design of the ADS data pipeline is also presented, from raw sensor input to the Controlled Area Network Bus (CAN Bus) interface, as well as the human-computer interface.

The first ADS module discussed is the mapping and environment modeling module. Environment modeling is the backbone of how autonomous agents understand the world, and therefore has significant implications for decision-making and verification. Motivated by the success of relational mapping tools such as Lanelet2, we present the Dynamic Relation Graph (DRG). The DRG is a novel method for extending prior relational maps to include online observations, creating a unified environment model which incorporates both prior and online data sources. Our prototype implementation models a finite set of heterogeneous features including road signage and pedestrian movement. However, the methodology behind the DRG can be expanded to a wider range of features in a fashion that does not increase the complexity of behavioral planning. Simulated stress tests indicate the DRG's effectiveness in decreasing decision-making complexity, and deployment to the WATonomous research vehicle (Bolty) demonstrates its practical utility. The prototype code is available at github.com/WATonomous/DRG.

The second ADS module discussed is the action classification module. When applied in the context of Autonomous Vehicle (AV)s, action classification algorithms can help enrich an AV's environment model and understanding of the world to improve behavioral planning decisions. Towards these improvements in AV decision-making, we propose a novel online action recognition system, coined the Road Action Detection Network (RAD-Net). RAD-Net formulates the problem of active agent detection and adapts ideas about actor-context relations from human activity recognition in a straightforward two-stage pipeline for action detection and classification. We show that our proposed scheme can outperform the baseline on the International Conference of Computer Vision (ICCV) 2021 Road Challenge dataset [1]. Furthermore, by integrating RAD-Net with the ADS' perception stack and the DRG, we demonstrate how a higher-order understanding of agent actions in the environment can improve decisions on a real AV system.

The last ADS module discussed is trajectory planning and control. Trajectory planning and control have historically been separated into two modules in automated driving stacks. Trajectory planning focuses on higher-level tasks like avoiding obstacles and staying on the road surface, whereas the controller tries its best to follow an ever changing reference trajectory. We argue that this separation is (1) flawed due to the mismatch between planned trajectories and what the controller can feasibly execute, and (2) unnecessary due to the flexibility of the Model Predictive Control (MPC) paradigm. Instead, in this thesis, we present a unified MPC-based trajectory planning and control scheme that guarantees feasibility with respect to road boundaries, the static and dynamic environment, and enforces passenger comfort constraints. The scheme is evaluated rigorously in a variety of scenarios focused on proving the effectiveness of the Optimal Control Problem (OCP) design and real-time solution methods. The prototype code is available at github.com/WATonomous/control.

Acknowledgements

I would like to thank my primary advisors *Derek Rayside*, *William Melek*, and *Mohammad Al-Sharman* for their guidance and how they motivated me to structure my research into conference submissions which became the foundation for this thesis. Advising graduate students is one of the great selfless acts, and I commend and thank you for it. I also thank the co-authors of the three conference papers that were the outcome of this thesis: *Jeffery Li* and *Yeshu Jain* (Environment Modeling), *Alex Zhuang*, *Quanquan Li*, *Eddy Zhou*, and *Ali Budhwani* (Action Classification) and *Mohammad Al-Sharman* (Control).

I would also like to thank the hundreds of WATonomous team members who I worked with over the past 5 years. Your contributions to the AV student community at The University of Waterloo (UW) is what made my thesis contributions possible. I would specifically like to thank the students who first submitted UW's application to the SAE AutoDrive Challenge I: *Krisztian Kurucz*, *Briar Smith*, *Ella Rasmussen*, *Richard Lee*, *Chad Bissett* and *Utkarsh Saini*, among others. I thank *Tai (Tom) Sun* as my first manager at WATonomous who fostered my interest in the team and the AV space in general. I thank *Charles Zhang*, *Chu Qing (Anita) Hu*, and *Ruizhe (Ray) Li* for their hard work and dedication to the team to bring it out of a poor showing in 2019 to a strong showing in 2020. I thank *Chuan Tian (Ben) Zhang* for his dedication to providing WATonomous with a powerful server cluster, which was instrumental to the success of WATonomous through the COVID-19 pandemic.

I would also like to thank the wonderful WATonomous co-op cohorts we have had the privilege of mentoring over the years: *Chad Kirton* and *Vinayak Sharma* (F19), *Mitchell Hoyt*, *Zi Wen Lim*, and *Tony Jung* (S19), *Frank Yan* (W20), *Andrew Guo* and *Tae Park* (W21), *Alex Zhuang*, *Eddy Zhou*, *Bao Ha*, and *Julian Yam* (S22).

I would also like to thank the organizing committee of the SAE AutoDrive Challenge I: *Jamie Knauff*, *Allison Hostetler*, and *Kevin Dietrich*, among others. As well as the mentors from the General Motors (GM) side: *Akram Abdel-Rahman* and *Kevin Cochran*.

I would also like to thank *Ross McKenzie* and the entire *WatCAR* organization for their support of the vehicle ecosystem at UW, and the large help they were to WATonomous throughout the years.

Finally I would like to thank those who have carried on and are carrying on the WATonomous legacy after me, specifically *Victor Zheng* for serving as captain and *Eddy Zhou* as technical director.

Dedication

I dedicate my work to my mom Rebecca and my dad Matthew who have supported me in every aspect of my life, for my entire life. Thank you.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

UW	The University of Waterloo
DRG	Dynamic Relation Graph
AD	Automated Driving
ADS	Automated Driving Stack
AV	Autonomous Vehicle
GNSS	Global Navigation Satellite System
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
IDE	Integrated Development Environment
VNC	Virtual Network Computing
CAN Bus	Controlled Area Network Bus
VM	Virtual Machine
ICRA	International Conference of Robotics and Automation
ICCV	International Conference of Computer Vision
RAD-Net	Road Action Detection Network

DDT	Dynamic Driving Task
OCP	Optimal Control Problem
MPC	Model Predictive Control
NMPC	Nonlinear Model Predictive Control
SAE	Society of Automotive Engineers
GM	General Motors
HSV	Hue, Saturation, Value
API	Application Programming Interface
ROAD	ROad event Awareness Dataset
RPN	Region Proposal Network
LFB	Long-Term Feature Bank
NLP	Non-Linear Program
VTN	Video Transformer Network
FFW	Feed Forward Network
ACAR-Net	Actor-Context-Actor Relation Network
ACAR	Actor-Context-Actor Relation
RoI	Region of Interest
mAP	mean Average Precision
AP	Average Precision

DP	Dynamic Programming
CoG	Center of Gravity
FPN	Feature Pyramid Network
GP	Gaussian Process
WRESTRC	Waterloo Region Emergency Services Training and Research Centre
IPOPT	Interior Point Optimizer
RK4	Fourth Order Runge-Kutta

Nomenclature

WATonomous A student design team at the UW full of undergrad and grad students working on automated driving. See http://watonomous.ca/ for details.

Bolty Name of the AV research platform used to validate software at WATonomous

SAE AutoDrive Challenge I A collegiate competition which ran from Sep 2017 - Jun 2021 in which UW was one of the competitors. Competitors were challenged to build an AV capable of completing Automated Driving (AD) scenarios designed by a panel of judges. See https://www.sae.org/attend/student-events/autodrive-challenge for details.

Dynamic Relation Graph A novel environment modeling and decision modeling technique developed by WATonomous and presented at International Conference of Robotics and Automation (ICRA) 2022

GUI Tools A container introduced to the *watod* ecosystem for the purpose of running desktop based visualization programs and exposing a VNC server to allow for client side interaction with said GUI programs.

Chapter 1

Introduction

1.1 Motivation

This thesis discusses the implementation of an AV robo-taxi that operates in urban environments. The research presented in this thesis targets a deployment to Bolty, WATonomous' research vehicle. The requirements for Bolty's autonomous capabilities were set by the SAE AutoDrive Challenge I which ran from September 2017 - June 2021. Since June 2021, WATonomous has been self-motivated with the goal of maintaining automated driving involvement of undergrad and graduate students at UW, and advancing the field of AV research and development.

Some examples of the robo-taxi requirements are:

1. Lane topology must be navigated while obeying traffic rules (staying on road surface, obeying turning lanes, etc...).
2. Lane topology altering road signage (right turn only, do not enter) must be obeyed.
3. Traffic signals must be obeyed.
4. Dynamic agents including non-ego vehicles and pedestrians must be avoided.

See Appendix A for a full definition of the requirements.

1.2 Scope

To limit the scope of this thesis, object detection, tracking, and ego localization are not considered. In practice on Bolty, object detection is done via off-the-shelf YOLOv5 [4], tracking is done via a simple linear Kalman Filter, and ego localization is done via off-the-shelf Novatel SPAN Global Navigation Satellite System (GNSS) and Inertial Measurement Unit (IMU) devices to give us a highly accurate Inertial Navigation System (INS). Further background information on the entire ADS is given in Chapter 2.

What is in scope for this thesis includes: (1) The software architecture design, including the Dockerized ROS network design and the ADS data pipeline, (2) Lanelet2 mapping, environment modeling, and behavioral planning, (3) action classification in video steams, and (4) trajectory planning and control.

1.3 Contributions

1.3.1 Software Architecture and Data Pipeline

1. The software that makes up an ADS is highly heterogeneous. Different modules depend on different software packages, and even different operating systems. To isolate the dependencies of the different ADS modules, a novel WATonomous Docker (*watod* for short) ecosystem was created based on Docker Compose.

2. *watod* has Dockerized CARLA simulation built-in, allowing any ADS module contained in the *watod* ecosystem to be quickly tested using simulated data.

3. Multiple instances of the *watod* ecosystems can be running simultaneously on a single VM on WATonomous' server cluster using Docker's network isolation protocol. This enables dozens of WATonomous' software developers to work in parallel without conflicting data streams.

4. Inside of *watod*, an entire ADS has been implemented that can run in real-time to navigate simulated CARLA environments, or be deployed to Bolty for usage in the real world.

These software architecture contributions were demonstrated at WATonomous' Year 4 Demo (`https://youtu.be/DNZgheT4Y2s`), which earned WATonomous 2nd place in the SAE AutoDrive Challenge I (Year 4).

1.3.2 Mapping, Environment Modeling, and Decision Making

Standardized methods and toolboxes for creating and embedding relationships between prior maps and online observations are lacking in the literature, making the logic of AV decision systems opaque. Many different decisions could be made using the same prior and online information, depending on how the relationships are modeled. Some systems must be safer than others, but without standardization there is little hope of being able to compare implementations. The ability to compare implementations goes hand-in-hand with the ability to verify a model, since verification can be done by comparing to some benchmark. Verifying an AV's model of the world is essential to explaining why a decision was made, an area where the AV industry has struggled to meet the public's expectations[1].

In light of these holes in the literature, we propose our work on the DRG which offers the following contributions:

1. The DRG serves as a standard tool for extending prior maps with online observations to create a unified environment model (see Fig. 3.1), allowing for greater transparency and collaboration on decision-making algorithms.
2. A novel method for analyzing the combinatorial complexity of decision making in urban environments, and an analysis of how the DRG reduces these complexities.
3. Design details on various DRG augmentation routines for common environment features, including pseudo-code and a C++ prototype implementation which are released at github.com/WATonomous/DRG.

These contributions were presented at ICRA 2022 in Philadelphia PA under the title *DRG: A Dynamic Relation Graph for Unified Prior-Online Environment Modeling in Urban Autonomous Driving* [6].

1.3.3 Action Classification

The perception task in automated driving is to understand the scene around the ego vehicle using camera data. What constitutes "understanding" is an on-going conversation, motivated by what types of information are necessary for safe and effective driving decisions

[1]The infamous Uber ATG fatality in 2018 notably lacked an explanation of the autonomous system's model of the world before the crash, and led to Uber ceasing testing [5].

to be made autonomously. Historically, the literature has focused on agent detection and classification tasks which operate on single image frames [7]. However, as the field of automated driving matures, the perception research community is shifting their attention towards tasks that represent a deeper understanding of the road scene as a whole [1]. A useful task in this regard is action classification. Separating actions requires the classifier to relate spatial and temporal information across frames. In the literature, human action recognition is a well studied topic, and this thesis transfers and pushes those ideas further in the context of a road action detection network (coined RAD-Net), with contributions including:

1. Demonstration that action classifiers original built for human action recognition are also effective in the road scene domain, and in fact benefit from pretraining on large human action datasets due to shared neural representations of actions between the two domains.

2. Formulation and solutions to the active agent detection problem. Using optical flow to better classify active and inactive agents.

3. A novel dynamic RoI-Alignment procedure which uses tracks of agents across RAD-Net's input view to better capture spatial features. This processes reduces spatial noise introduced by the high-motion scenes present in the ROAD dataset, something that previous action classification algorithms (which were designed for low-motion human datasets) did not address.

4. Superior performance on the ICCV 2021 ROAD Challenge, compared to the baseline RetinaNet-3D in [1].

5. Deployment of the proposed RAD-Net to the Bolty research platform, and demonstration of how it enhances the fidelity of the environment model and enables more accurate behavioral planning.

These contributions were submitted to ICRA 2023 in London UK under the title *RADACS: Towards Higher-Order Reasoning using Action Recognition in Autonomous Vehicles* [8].

1.3.4 Trajectory Planning and Control

Generally, the task of an AV robo-taxi can be seen as progressing towards a goal state in a feasible and efficient manner. In the context of urban driving, feasible means without collisions and while obeying traffic rules, and efficient means operating near the speed limit. There are several papers in the literature that address the real-time obstacle avoidance problem using optimal control techniques. However, all have drawbacks that make them ill-suited for the problem presented above, see Section 5.1.1 for details. This thesis addresses these issues by introducing a novel OCP formulation of the robo-taxi task and an accompanying MPC solution. The main contributions in this regard are:

1. The system model aligns closely with physical platforms and allows for the computation and thus the constraint of passenger comfort metrics mandated by Society of Automotive Engineers (SAE).

2. Constraints are added to the OCP formulation that guarantee feasibility of control actions with respect to road boundaries, static obstacles, and dynamic vehicles. In this way, trajectory planning can be executed simultaneously within the controller, and the final control actions are guaranteed to be dynamically feasible.

3. The controller operates in real-time by employing a novel parallel-solver method which uses a warm-started "online" solver for real-time performance, while employing a parallel "exploration" solver which serves to break out of warm-starting local minima.

These contributions were submitted to ICRA 2023 in London UK under the title *Real-Time Unified Trajectory Planning and Optimal Control for Urban Autonomous Driving Under Static and Dynamic Obstacle Constraints* [9].

1.4 Organization

This thesis is organized as follows:

- Chapter 2 presents the software architecture used to develop, run, and deploy the ADS to Bolty, as well as the ADS data pipeline.

- Chapter 3 covers the mapping framework used to create the backbone of Bolty's environment model, and how the environment model is augmented at runtime to make online decisions.

- The decision making capabilities of Bolty are extended in Chapter 4 by introducing a computational vision approach to action classification, which allows for more accurate augmentations to the DRG over the geometric based methods presented in Chapter 3.

- Finally, no ADS is complete without a control scheme, which is presented in Chapter 5. An MPC was developed that obeys vehicle body constraints and also takes into account a free space definition that changes dynamically over time (based on information that the environment model provides).

Chapter 2

Software Architecture and Data Pipeline

This chapter covers how the WATonomous ADS is architected in simulation and on Bolty.

Acknowledgement: The design and implementation of the technology covered here was done primarily by Rowan Dempster, with secondary contributions by Charles Zhang and Chuan Tian (Ben) Zhang.

2.1 Requirements

The requirements for the software architecture design were:

1. Support rapid prototyping of new software modules, and the rapid onboarding of new WATonomous team members. By rapid prototyping it is meant that when a new software library dependency or and entire new software module is introduced, that process should be painless. Developers should not have to worry about incompatible versioning. By rapid onboarding it is meant that new team members should only have to install a small set of additional software (limited to Integrated Development Environment (IDE) software and visualization software like VNC), and any modern laptop should be able to run such required software.

2. Support the concurrent use of a single VM in the WATonomous server cluster by multiple developers. This means multiple WATonomous team members can do development work simultaneously using the CARLA simulator on the same VM. Thus,

data coming from CARLA and data being exchanged between ADS modules must be isolated for every VM user.

3. Have the ability to deploy the ADS developed in simulation to the production Bolty vehicle with no modifications to the software modules themselves.

2.2 Solutions Implemented

In light of these requirements, the *watod* ecosytem was designed. In *watod*, each software module (e.g. the detector, tracker, environment model, occupancy grid, etc...) is containerized into its own Docker container which only has that module's software dependencies installed (see right side of Fig. 2.1). Thus, when a new dependency is required for a specific ADS module, only the that module's Docker image needs to be updated, without having to worry about affecting the whole system.

watod also supports inter-module communication via ROS. A new module can be added to *watod* as a new Docker container and hook-in to the existing data pipeline using the ROS communication protocols.

An instance of the *watod* ecosystem can be run remotely on any of WATonomous' server cluster VMs and connected to via the Visual Studio Code IDE. Thus, there is no hardware requirements for new members' laptops to interact with *watod*. New team members can get up and running just by connecting to a VM and executing the *watod* commands necessary to start the Docker containers they need to do development (see top left side of Fig. 2.1).

The CARLA simulator is also Dockerized and available via *watod*. CARLA provides sensor inputs to the software modules via the Carla ROS Bridge package, allowing for simulation based development of the ADS. *watod*, via Docker network isolation, ensures that data coming from CARLA stays within each VM user's separate ADS instance, allowing many simulators and ADS instances to be running concurrently on a single VM. Thus, there is no software-enforced limit to how many concurrent users the VMs in the WATonomous server cluster can support. There are limits imposed by the hardware resources needed to run the simulations. At the time of writing, the WATonomous server cluster can support 50+ concurrent team members remotely developing using CARLA.

Furthermore, the Carla ROS Bridge package is configured to publish information in the same format as the actual sensors on Bolty. Therefore, the ADS modules need not be aware if they are running in the context of the CARLA simulator, or the real world. This makes deployment to the research vehicle simple. The only thing that changes is where

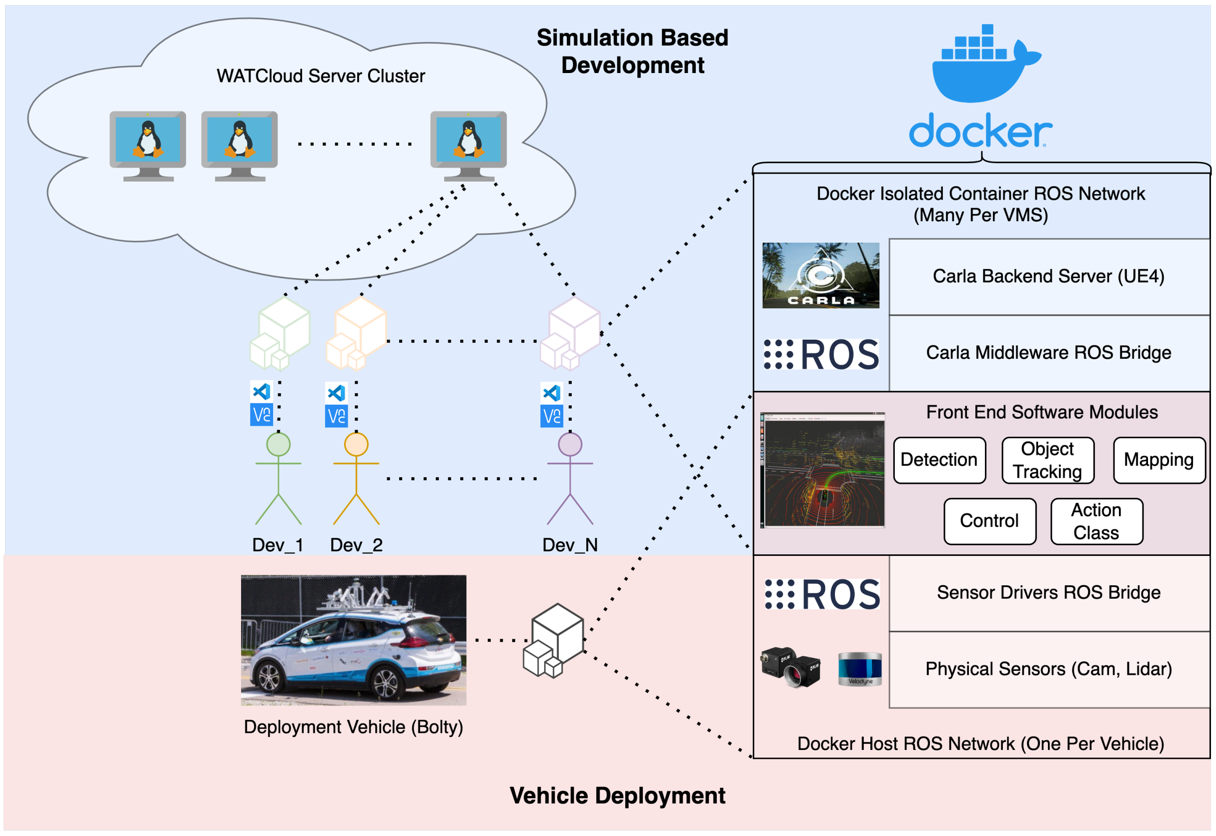
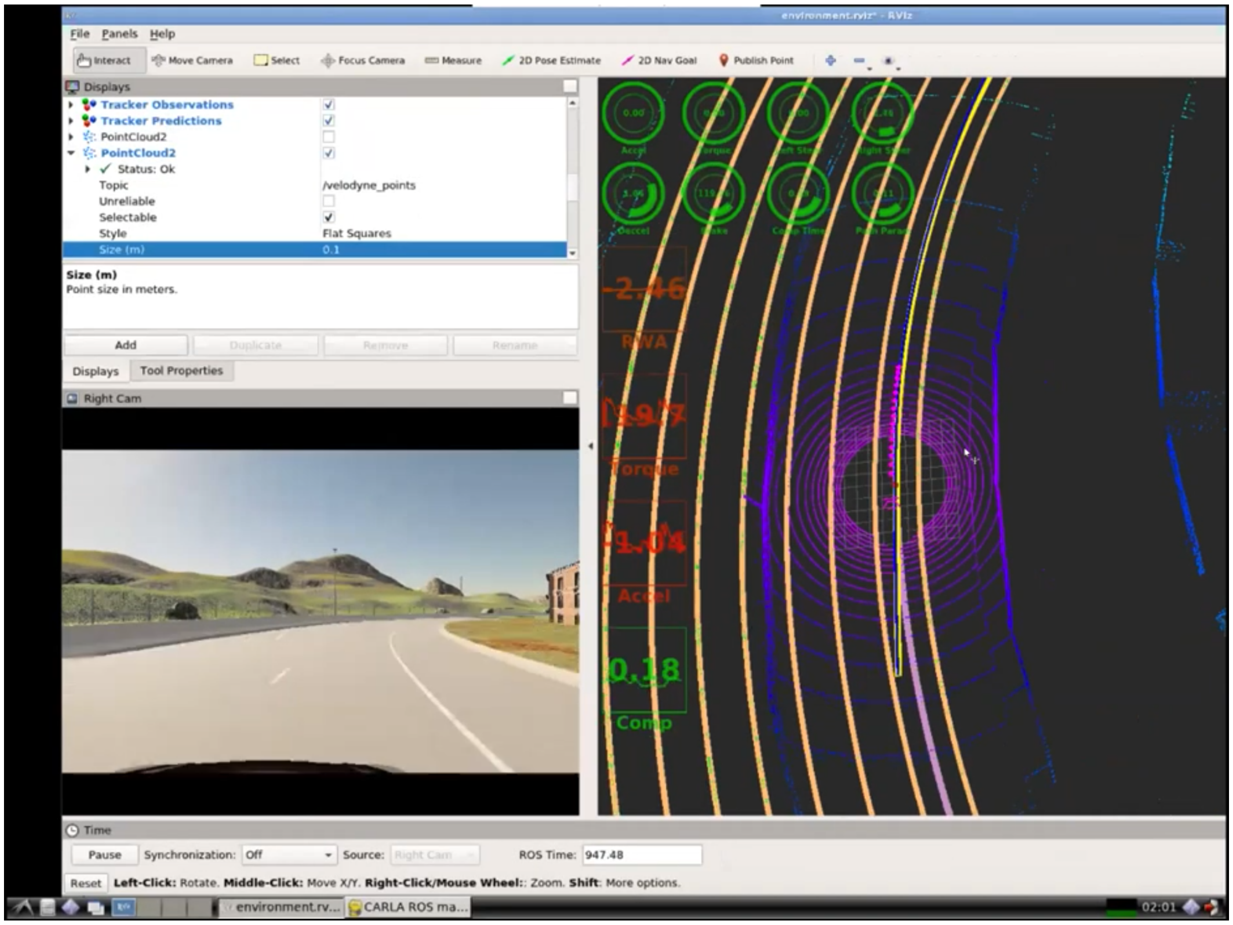

Simulation Based Development
WATCloud Server Cluster
docker
Docker Isolated Container ROS Network (Many Per VMS)
Carla Backend Server (UE4)
ROS
Carla Middleware ROS Bridge
Front End Software Modules
Detection
Object Tracking
Mapping
Control
Action Class
Dev_1
Dev_2
Dev_N
ROS
Sensor Drivers ROS Bridge
Physical Sensors (Cam, Lidar)
Deployment Vehicle (Bolty)
Docker Host ROS Network (One Per Vehicle)
Vehicle Deployment

Figure 2.1: Diagram of WATonomous' approach to software development and deployment. On the simulation side (top), the penguin monitors represent abstracted Ubuntu VMs running on some physical hardware in the WATonomous server cluster. Multiple developers can run isolated ADS instances on a single VM, and can interact with their instance using Visual Studio Code and a VNC client. Each isolated ADS instance is spun up using *watod* utilities, and includes an instance of CARLA, the Carla ROS Bridge, and however many "front-end" software modules the developer needs for their work. On the deployment side, there is only ever a single instance of the ADS running on Bolty, and the CARLA simulator inputs are replaced by the physical sensors. However, from the perspective of the "front-end" software modules, nothing has changed.

the sensor input is coming from and thus no changes to the software modules are needed for deployment of the ADS.

One downside of a fully Dockerized simulator and ADS is that any desktop based



Status: Ok
Topic /velodyne_points
Unreliable
Selectable ✓
Style Flat Squares
Size (m) 0.1

Size (m)
Point size in meters.

Synchronization: Off Source: Right Cam ROS Time: 947.48

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift**: More options.

Figure 2.2: Screenshot RViz running in the GUI Tools container, accessible to remote developers via a VNC client software.

visualization (like the popular RViz software for robotics) is not immediately supported. To move past this limitation, the GUI Tools container was added to the *watod* ecosystem. The purpose of this container is solely to run desktop based visualization software such as RViz and expose a VNC server. This way, remote developers are able to interact with any desktop software via a VNC client application on the developer's local laptop. Fig. 2.2 shows the RViz instance that developers see when they connect to the VNC server exposed by the GUI Tools container.

An in-depth tutorial of how to use the *watod* ecosystem in conjunction with ROS and the WATonomous server cluster can be found at:

https://drive.google.com/file/d/17N2Zrw1cjUfZgALg3ImMrmcRMzwkOGVz/view. Additionally, an in-depth tutorial of how to use the CARLA simulation functionality built into *watod* can be found at:
https://drive.google.com/file/d/1F32Ui6QaegQ8wSl-ga5ifP1fNKGzba09/view.

2.3 ROS Software Module Data Pipeline Design

The *Front End Software Module* block on the right side of Fig. 2.1 is where the majority of the design work of the ADS was done. A more detailed view of that block is shown in Fig. 2.3. See Appendix B for the custom ROS message definitions mentioned in this section.

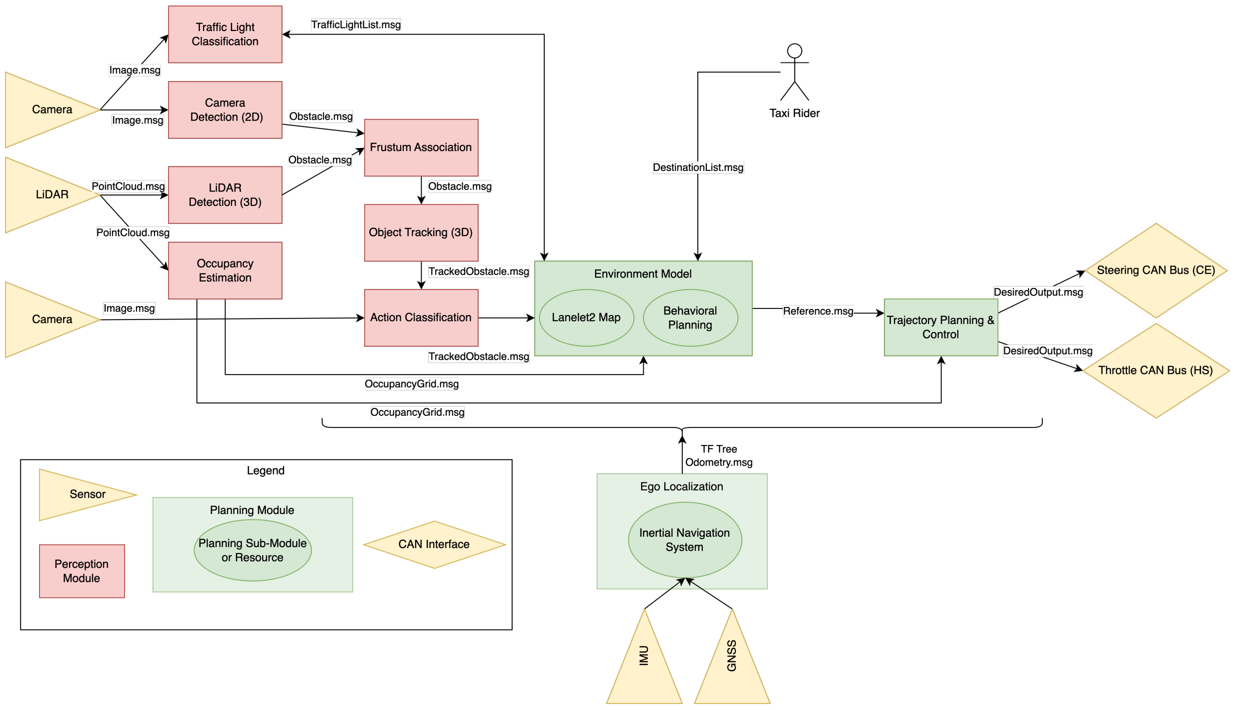


Figure 2.3: The WATonomous ADS data pipeline. Uni-directional arrows represent “topics” in the publisher-subscriber ROS communication paradigm. Bi-directional arrows represent topics in the client-server ROS communication paradigm. Arrows are labeled with the data type (ROS message) being transmitted over that topic.

Sensor Interfacing

WATonomous uses cameras (Blackfly S Color 3.2 MP GigE Vision) and LiDARs (Velodyne Ultra Puck) to perceive the world, and a GNSS + IMU Span Device (Novatel SPAN PwrPak7-E1) to provide odometry. First, raw sensor readings are received from the sensors and processed into ROS messages by the sensor drivers (Pointgrey Camera Driver, Velodyne LiDAR Driver, and Novatel SPAN Driver), and then are published into the ROS network so that the other software modules can further process the information.

Localization

The Novatel SPAN Driver implements a complete inertial navigation system, and publishes an *Odometry.msg* messages at 20Hz into the ROS network, which many software modules subscribe to. The localization stack also sets up the ROS transform tree.

Perception

Next, the perception modules perform detection, classification, and tracking to transform the high dimensional sensor modalities (images and point clouds) into low dimensional representations that the environment model can understand. YOLOv5 [4] is used for 2D object and sign detection in camera images, and publishes *Obstacle.msg* messages. The euclidean clustering package from Autoware [10] is used for 3D obstacle detection on the LiDAR point clouds, and also publishes *Obstacle.msg* messages. A custom frustum-based heuristic is used for 2D-3D association. The occupancy grid estimation module used was developed by Autonomoose [11], and is based on the well-known log odds formulation from Probabilistic Robotics by Thrun *et al.* [12]. The environment model sends the 3D positions of upcoming traffic lights to the traffic light state classification module, where they are projected into the camera 2D frame and then classified in 2D using Hue, Saturation, Value (HSV) thresholds. The traffic light state classification module publishes *TrafficLight.msg* messages.

Object tracking is done in 3D after the frustum-based association. A modified version of the AB3DMOT [13] algorithm (a benchmark 3D multi-object tracker which uses a linear Kalman Filter and Hungarian matching for association) is used, and publishes *TrackedObstacle.msg* messages. These tracks as well as the camera stream are then sent to the action classification module which appends a history of actions to each track using the neural design described in Chapter 4.

Mapping and Environment Modeling

Offline high definition Lanelet2 maps [14], which describe lane geometry and topology, are used for navigation and motion planning. These maps are crafted by hand using the JOSM map editing software [15].

The environment modeling module unifies the prior Lanelet2 map with the online perception outputs, creating a combined relational graph that is used for behavioral planning. This process is described in detail in Chapter 3. The behavioral plan is expressed as a *Reference.msg* message, which is continuously sent to the controller.

Motion Planning and Control

The controller consumes the *Reference.msg* message as well as a description of free space (*OccupancyGrid.msg*) from the occupancy estimation module. These messages are then used as parameters for a non-linear program which is continuously solved as part of a MPC design that optimizes trajectory and control actions (longitudinal acceleration and road wheel angle). These control actions are then sent to the vehicle's CAN Bus. Details of the controller implementation are given in Chapter 5.

CAN Bus Interfacing

The control actions are sent to the CAN Bus modules, or the CARLA simulator in the context of remote simulation development, via the *DesiredOutput.msg* message. The control actions are then actuated and thus the robotic feedback loop is closed

Human-Computer Interface

When a user of the system enters the vehicle, they select a point on the Lanelet2 map which is displayed via RViz. This point gets translated into a destination lane, which then gets sent to the environment model as a *DestinationList.msg* message. The environment model next runs a search over its internal graphical representation of a world to find a route from the AV's current location to the user's destination, and the autonomous drive begins.

Chapter 3

Mapping, Environment Modeling, and Decision Making

Acknowledgement: The work in this chapter was done primarily by Rowan Dempster, with help from those acknowledged in Section 3.9.5

3.1 Introduction

Decision making in AVs relies completely on the ability of the agent to aggregate disparate information sources into a useful model of the agent's environment. Examples of disparate information sources include: hand-crafted lane geometries (see top left of Fig. 3.1), tracks produced by a perception scheme (see middle right of Fig. 3.1), or even a prior environment model from a previous drive. In this work, we tackle modeling these disparate sources using relationships, and study how to embed relationships in a unified graphical model we refer to as the Dynamic Relation Graph (DRG).

Motivating our DRG approach anthropologically, humans do not make decisions based on isolated objects floating around in a disconnected world, but rather using relationships between objects to elicit higher order properties. For example, we infer that the keyboard in front of us has a relationship with the desk it is sitting on. We understand the properties of this relationship: the keyboard is on top, the desk is solid. Using this understanding, we decide that we can carry on typing without the keyboard falling away from our fingers.

Widely accepted prior maps of static environment features are relationship-rich [16, 14, 17]. Such models encode lane neighborhood/successor relations, relationships between

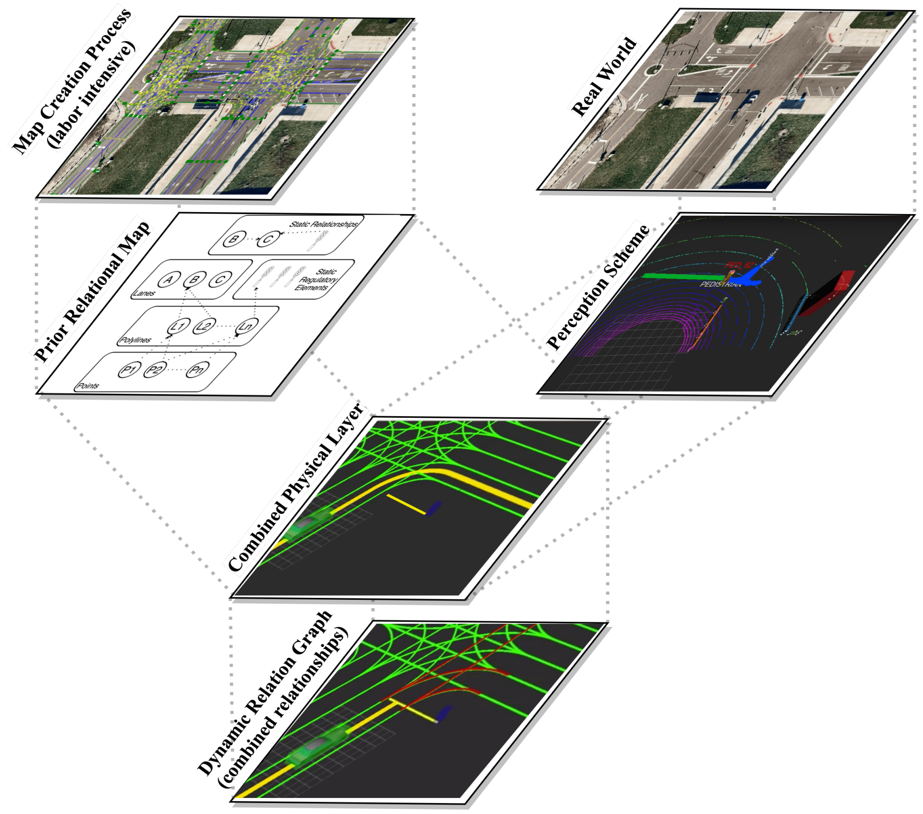

Map Creation Process (labor intensive)
Real World
Prior Relational Map
Static Relationships
Lanes
Static Regulatory Elements
Polylines
Points
Perception Scheme
Combined Physical Layer
Dynamic Relation Graph (combined relationships)

Figure 3.1: The information layers that are fused to create the DRG. Top right is a section of the MCity Course (https://mcity.umich.edu/), top left is the prior map of the same section. The middle three layers show how physical information from the map and tracker are combined, and the bottom layer shows the conflict relationship created.

traffic lights and the lane(s) they control, and have the potential for much more. These relationships are handcrafted offline by humans during the laborious map creation process. Therefore, the relationships can be highly complex and expressive if we are willing to incur high creation costs (time and effort). However, prior maps are static representations. In order to make dynamic decisions, relationships between online detections and the prior map must be built at system runtime.

Standardized methods and toolboxes for creating and embedding prior-online feature relationships are lacking in the literature, making the logic of AV decision systems opaque. Many different decisions could be made using the same prior and online information, depending on how the relationships are modeled. Some systems must be safer than others, but without standardization there is little hope of being able to compare implementations. The ability to compare implementations goes hand-in-hand with the ability to verify a model, since verification can be done by comparing to some benchmark. Verifying an AV's model of the world is essential to explaining why a decision was made, an area where the AV industry has struggled to meet the public's expectations[1].

In light of these holes in the literature, we propose our work on the DRG which offers the following contributions: (1) The DRG serves as a standard tool for extending prior maps with online observations to create a unified environment model (see Fig. 3.1), allowing for greater transparency and collaboration on decision-making algorithms. (2) A novel method for analyzing the combinatorial complexity of decision making in urban environments, and an analysis of how the DRG reduces these complexities. (3) Design details on various DRG augmentation routines for common environment features, including pseudo-code and a C++ prototype implementation which are released at github.com/WATonomous/DRG.

The rest of this chapter is organized as follows: Section 3.2 presents related work on offline mapping toolsets and attempts at prior-online modeling, followed by our novel DRG methodology of extending a prior relationship graph with online entities in Section 3.3. Section 3.4 then briefly covers our implementation of specific feature designs to illustrate the practical use of the DRG, and Section 3.5 presents the novel method of analyzing the reduction in decision-making complexity afforded by the DRG. Simulation and on-road results are discussed in Section 3.7, conclusions are drawn in Section 3.8, and Section 3.9 serves as an appendix for implementation details. To limit the scope of our proposed work, we are not concerned with ego state estimation errors nor measurement uncertainties, which are assumed to be handled by upstream modules [18, 19].

3.2 Related Work

The mapping toolset and graph implementation which DRG uses is Lanelet2 [14] [20], an offline tool for handcrafted static maps. A Lanelet2 map consists of three layers: a physical layer, a relational layer, and a topological/routing layer (see Fig. 3.2). The physical layer

[1]The infamous Uber ATG fatality in 2018 notably lacked an explanation of the autonomous system's model of the world before the crash, and led to Uber ceasing testing [5].

contains the observable elements of the world represented using points and linestrings. Points are the basic elements of the map described by their three-dimensional position in metric coordinates and linestrings are an ordered array of points used to represent the polygonal elements in the map.

The relational layer forms edges between elements of the physical layer, thus introducing relational entities such as lanelets, areas, and regulatory elements (short: regElems). Lanelets are used to identify sections of the map with directed motion, like vehicle lanes, pedestrian crossings, rails, etc. A lanelet owns one linestring as the left border, one as the right, and optionally owns regElems describing the traffic rules applicable to the lanelet. RegElems are used to define traffic rules such as speed limits or traffic signals[2]. The neighborhood relationships between lanelets generate a topological layer, also known as the routing graph. The routing graph arises from a network of passable regions, where the exact topology depends on the road user at hand (emergency vehicles are allowed to take different routes than passenger vehicles).

Lanelet2 is purely a tool for offline map creation, whereas the DRG is capable of augmenting prior maps autonomously during system runtime, which is far outside of the Lanelet2 design scope. The RoadGraph model [21] [22] and other graph based environment modeling techniques [23] are closer in functionality to the DRG. As with our work, these papers deal with aggregating and fusing information obtained at runtime of the system with a prior map. However, they fall short of our work in key places: (1) They do not give design details or examples of how object tracks from on-board sensors are incorporated into the model (see our design details in Section 3.4). (2) They focus on describing relationships in the prior map, which are now part of the Lanelet2 library. Our work uses the established Lanelet2 library as our prior map and focuses on pushing the design paradigms forwards, towards a unified online model. (3) They do not cover how their RoadGraph model is mutated in cases where certain lanes are no longer traversable due to signage or blockage. Our work examines in detail the problem of modeling features that affect routing decisions.

In [24], Koschi and Althoff describe a reachability set approach to analyzing the interactions between online tracked features and a prior map. The scope of their design is large, considering phantom tracks and abstractions of vehicle dynamic models in their reachability analysis. However, the paper does not consider extracting the information of surrounding traffic participants from sensor measurements, and the uncertainty of these measurements. In contrast, our approach is integrated into a full scale AV system; we explicitly state and deal with perception, tracking, and occupancy estimation schemes. Additionally, our system was validated in a closed-loop fashion whereas Koschi's and Althoff's approach was

[2]In our work, the DRG primarily exploits regElems as our graph entity for modeling online observations.

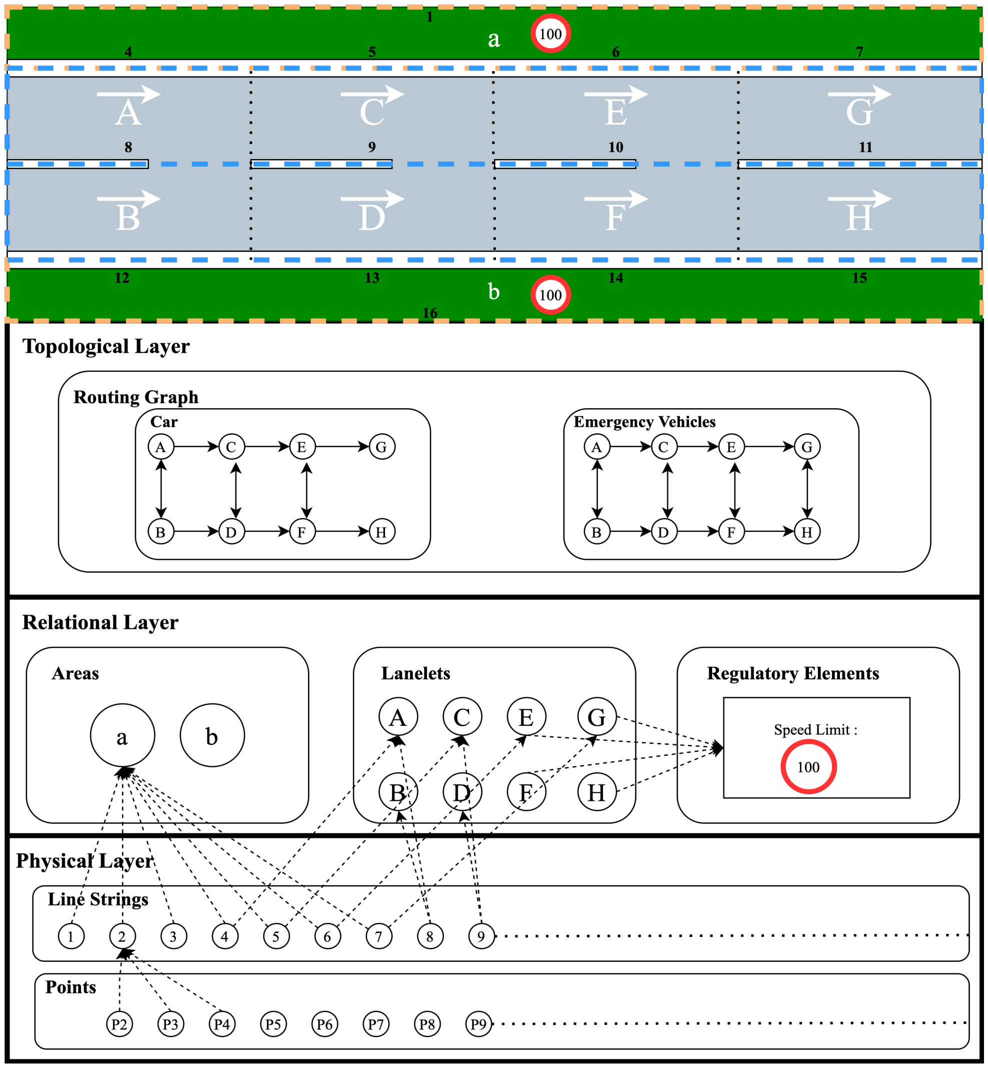


Figure 3.2: A one way street illustrating the different layers of a Lanelet2 map. The lanelets are represented using uppercase letters, the areas using lowercase letters, and the linestrings using numbers.

only evaluated in an open-loop fashion.

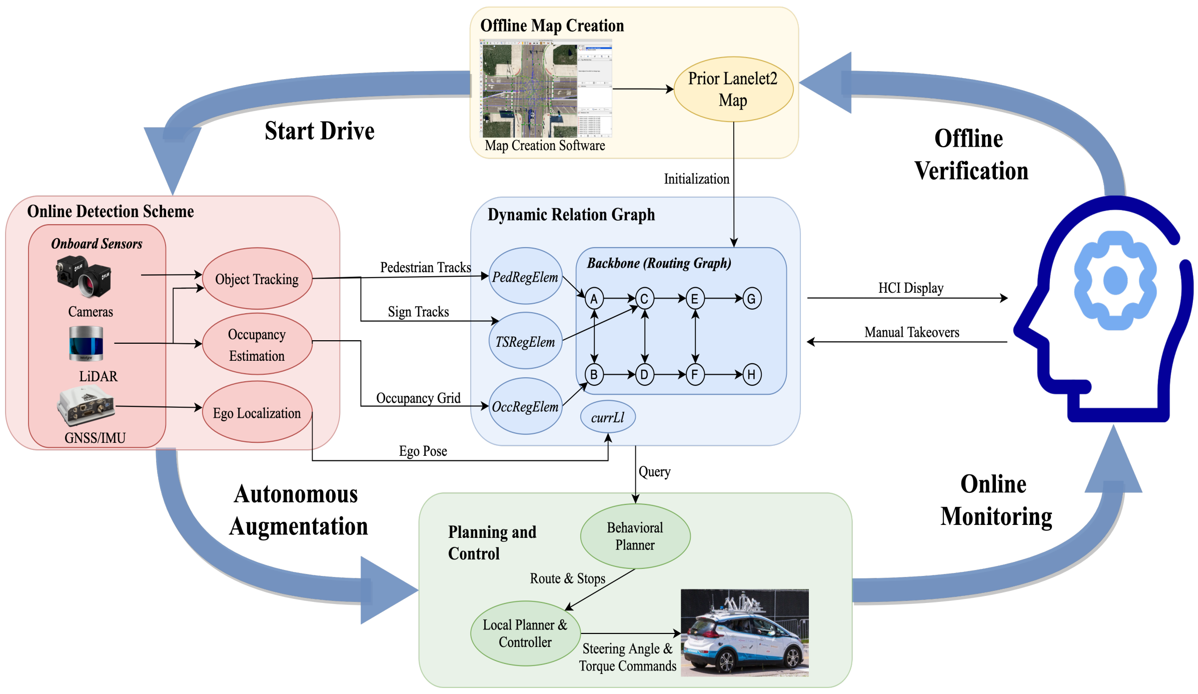


Figure 3.3: Data flow diagram illustrating the DRG's lifecycle during a drive, as described in Section 3.3

.

3.3 Methodology

The DRG sits between the measurement (perception) and decision (planning) layers of the AV stack (see Fig. 3.3). It represents a map-centric view of the environment, as opposed to the common ego-centric approach [25] [26]. The DRG is implemented as a graph of relationships which is being continuously augmented by the fusion of new online measurements.

Initialization: On AV system start-up, the DRG is initialized to the prior map. The features and relationships in the prior map may have been handcrafted, or they may have been autonomously created and embedded during a previous drive. This flexibility allows the DRG to continuously and autonomously refine itself, with the possibility of humans in the loop for verification and correction.

Augmentation: During a drive, the graph structure of the DRG is augmented over time as new measurements become available. Details of how various specific environment features mutate or create new entities and relationships in the DRG is addressed in Section 3.4. However, the DRG is flexible to any sort of augmentation that can be cast as an update to an existing entity, or as a new entity which has a relationship to an existing entity. All physical information used to create or update an entity or relationship must be stored in the DRG for use in verification as described below.

Querying: The primary function of the DRG is to be queried by the decision making module, providing all necessary physical and relational information. The DRG serves as a single source of truth for the decision making module, and therefore explains every behavioral decision the AV system makes. Section 3.5 details how the decision making complexity of a generic behavioral planner is reduced using the DRG.

Verification: In the event of an accident, the state of the DRG at any point in time can be inspected offline and run through the behavioral planner to reproduce the sequence of decisions that lead to the accident. Other designs that do not have a centralized environment model must store (in the worst case) all sensor data to reproduce behavioral decisions. Furthermore, the state of the relationship graph can be monitored and interpreted online by a safety driver. If at any point in time the relationship graph does not match what the safety driver observes, they can stop the vehicle before a flawed behavioral decision is made. Because of the DRG's map-centric design, online verification and data sharing about the environment between multiple autonomous agents is possible.

3.4 Implementation

The methodology behind the DRG is implementation agnostic, any mapping toolset and graph library implementation can be used. For our prototype we used the Lanelet2 C++ library[3] due to existing implementations of convenient entities like points, linestrings, lanelets, and regElems, as well as methods to describe properties (via the *attributes* API) and relationships (via the *addRegElem* API). Entities that did not already exist, e.g. pedestrians, were implemented as new regElems. The handcrafted lane geometry and topology (i.e. the routing graph) is the "backbone" of the DRG implementation, which can be traversed using the *besides*, *next*, and similar Lanelet2 graph APIs. Thus, graph search routines that allow for expressive queries can be implemented. During runtime the backbone is augmented by mutating entity properties or instantiating new regElem instances and forming relationships between those regElems and the backbone.

[3]Descriptions and algorithms rely on Lanelet2 Application Programming Interface (API) concepts covered in Section 3.2 and in the Implementation Details Section 3.9, which also contains more detailed descriptions of the augmentations routines mentioned in this section.

The remainder of this section covers four case studies of DRG augmentation routines which run as the AV perceives the environment. It is important to note that the DRG methodology can be applied on a wide range of features using different augmentation routines. The routines presented here are intentionally simple as the focus is on the DRG methodology; more complex routines are discussed in Section 3.8.

A. Stop Signage: In our implementation, we assume that signage is not part of the prior map, and that the DRG ingests tracks of stop signs at runtime with their 3D position and a unique ID. A regElem is instantiated using the *TSRegElem* API to hold the sign's physical information and the *addRegElem* API is used to assign the stop sign to the lanelet(s) it regulates. The regulated lanelet(s) are chosen using a search routine over the DRG backbone. The behavioral planner then queries each lanelet the ego traverses for ownership of the regElem, stopping when necessary.

B. Intersection Signage: Signage such as No Right/Left Turn augments the DRG in a similar manner to stop signs. For each sign, regElems are initialized using the *TSRegElem* API and regulated lanelets are assigned ownership via the *addRegElem* API. The regulated lanelet search routine over the DRG backbone is presented in Algorithm 1. After the regElem is owned by the regulated lanelets, the behavioral planner queries for a new route that obeys the augmented routing graph because it is possible the previous route is now disallowed by the new regulation.

C. Closed Roads: The DRG also ingests occupancy grid information about the environment, from which untraversable lanelets can be identified. A regElem is initialized using the *OccRegElem* API to store the blocking occupancy grid, which is then added to each untraversable lanelet. The behavioral planner again uses the *findRoute* API to search for a new traversable route.

D. Pedestrian Movement: In our implementation, the DRG ingests tracks of pedestrians with their predicted and historical states. A regElem is instantiated using the *PedRegElem* API which holds the predicted and historical states. A lanelet conflict set is generated for the pedestrian based on its predicted and historical states as described in Algorithm 2. The regElem is then augmented with the stopping point of each conflict and added to the conflicting lanelets. Similar to stop signs, the behavioral planner queries each lanelet the ego traverses for ownership of *PedRegElems* and stops at the designated stopping point.

Algorithm 1 Intersection signage augmentation

```
Input: sign, DRG
Output: Augmented DRG
1: Initialize regElem ← TSRegElem(sign.type, sign.pos)
2: Initialize interLls ← {} {Empty dictionary of relevant intersection lanelets}
3: adjacentLls ← currLl.besides() {Set of all lanelets adjacent to the ego vehicle}
4: for ll ∈ adjacentLls do
5:     interLls.insert(DRG.findInter(ll)) {DRG.findInter API searches for a successor intersection lanelet}
6: end for
7: for ll ∈ interLls do
8:     if DRG.signControls(regElem, ll) {If the sign controls the turning direction of the lanelet} then
9:         ll.addRegElem(regElem)
10:        DRG.findRoute() {DRG.findRoute() API searches for a new route given the new regulation}
11:    end if
12: end for
```

Algorithm 2 Pedestrian movement augmentation

```
Input: ped, DRG, waitDist {The conflict radius around a waiting pedestrian}
Output: Augmented DRG
1: regElem ← PedRegElem(track)
2: confLls ← DRG.within(regElem.predictedStates)
3: if regElem.isWaiting() then
4:     confLls.insert(DRG.within(regElem.currState, waitDist))
5: end if
6: for confLl ∈ confLls do
7:     stopPoint ← intersect(confLl, track.predictedStates)
8:     regElem.setStop(stopPoint)
9:     confLl.addRegElem(regElem)
10: end for
```

3.5 Decision Complexity Analysis

In this section, we take a general view of behavioral planners, analyzing how any given planner deals with the combinatorial complexity of the perceived environment features.

In the literature, there is a notable lack of techniques for expressing the combinatorial complexity of making a decision, and thus we present our own novel approach: Let $F = \{C_1, \ldots, C_N\}$ be the feature class sets for the N distinct classes of features, (e.g. C_{ped} represents the set of pedestrians and $C_{ped}^{(1)}$ is a specific pedestrian instance). Let $E = \{e_1, \ldots, e_P\}$ be the set of atomic ego behaviors (a simple binary planner, with $P = 2$, may have $e_1 = driving, e_2 = stopping$). The task of any given behavioral planner is to implement a mapping between the *feature space* F, of size $N * \sum_{C_i \in F} |C_i|$, to the planner's range E.

There are two sources of combinatorial complexity in this mapping, intra- and inter-class interactions. For example, in the pedestrian feature set C_{ped}, intra-class interactions arise from considering pairs of instances $(C_{ped}^{(1)}, C_{ped}^{(2)})$ and extracting aggregated information such as minimum distance to the ego vehicle. Inter-class interactions must also be addressed. Consider the case where C_{cw} is the set of pedestrian crosswalk lanelets specified in the prior map and how the planner reacts depends on the pedestrian's proximity to the C_{cw} features. Here, the interaction between the pair of classes (C_{ped}, C_{cw}) is pertinent.

Let $I_{inter}(\cdot, \cdot)$ and $I_{intra}(\cdot, \cdot)$ be the interaction functions, which express the inter- and intra- sources of combinatorial complexity (see Fig. 3.4 for an illustration of these functions and their composition). Note that the output spaces of I_{inter} and I_{intra} scale quadratically over the number of feature classes, and the number of instances in each feature class, respectively:

$$\Theta(I_{inter}(F, F)) = |F \times F| = \Theta(N^2) \tag{3.1}$$

$$\Theta(I_{intra}(C_i, C_i))) = |C_i \times C_i| = \Theta(|C_i|^2) \tag{3.2}$$

Applying the $I_{intra} \circ I_{inter}(F)$ (or equivalently, $I_{inter} \circ I_{intra}(F)$) composition to the original feature space F produces a quadratic *decision space* D of size $N^2 * \sum_{C_i \in F} |C_i|^2$ (see bottom left of Fig. 3.4).

Our claim is that the relational graph structure of the DRG benefits any planner using it by mitigating the quadratic complexity of the decision space introduced by $I_{intra}(\cdot, \cdot)$ and $I_{inter}(\cdot, \cdot)$, resulting in an algorithm that scales linearly with the number of feature classes and instances.

First, regarding inter-class interactions, the DRG introduces a set of homogeneity operators $H_{ij}(C_{ll}) : C_i \mapsto R_j(C_{ll})$ which map heterogeneous feature classes C_i to homogeneous representations R_j. An operator is defined over all i, and for $j \in \{1...K\}$ where importantly

K is a small constant. The implementation of the operators depends on the prior map features, and thus all H_{ij} are parameterized by C_{ll}. Together, the set of operators transforms all heterogeneous feature classes into a homogeneous *representation space* $R = \{R_1 ... R_K\}$.

Concretely, $H_{ij}(C_{ll})$ is defined for $i = \{ped, car, cyclist\}$ and $j = lanelet\ conf$ by abstracting each heterogeneous feature class into a homogeneous lanelet conflict representation as described in Section 3.4-D for pedestrians. Furthermore, $H_{ij}(C_{ll})$ is also defined for $i = \{intersection\ sign,\ blocked\ road\}$ and $j = untraversable\ lanelet$ by applying the techniques presented in Sections 3.4-B and 3.4-C.

After applying the set of homogeneity operators to the feature space F, the subsequent application of $I_{inter}(\cdot, \cdot)$ is performed on the constant-sized representation space R (see Fig. 3.4 (right)). The effect is a reduction in inter-class combinatorial complexity:

$$\Theta(I_{inter}(R, R)) = |R \times R| = \Theta(K^2) = \Theta(1) \tag{3.3}$$

Note that $\Theta(N)$ homogeneity operators need to be defined, one for each of the N feature classes.

Next we discuss interactions of intra-class instances. For this discussion either the original feature space F or the representation space R can be used; we will use the feature space notation. Note that an instance $C_i^{(j)}$ becomes independent of another instance $C_i^{(k)}$ when conditioned upon their inter-class interaction with the prior map. For example, consider the case of $i = ped$. Here, the relative positions or trajectories of two pedestrians are irrelevant and can be ignored, given their relationship (a potential lanelet conflict) to the prior map (C_{ll}). Let $I_m = I_{inter}(C_{ll}, C_i^{(m)})$ be instance m's inter-class interaction with the prior map. The effect on intra-class combinatorial complexity is then:

$$\Theta(I_{intra}(C_i, C_i)) = \Theta(|C_i|) \quad \text{given } I_m, \forall m \in C_i \tag{3.4}$$

Eq. (4) effectively states that intra-class interactions between instances can be ignored under knowledge of their interactions with the DRG backbone, allowing the decision algorithm to process each feature class in $\Theta(|C_i|)$ time. Moreover, as mentioned previously, applying the homogeneity operations and calculating the inter-class interactions on the representation space instead of the feature space yields a constant number of inter-class interactions. Overall, the decision space size under the DRG is reduced to $\Theta(\sum_{C_i \in F} |C_i|)$ (see Fig. 3.4 (right)); linear in the number of observed features. This theoretical analysis is tested in Section 3.7.1, where a simulation with a crowd of pedestrians is carried out.

In summary, by solving the generic behavioral planner task $F \mapsto E$ in the context of the DRG structure, we are able to limit or completely eliminate inter- and intra- class

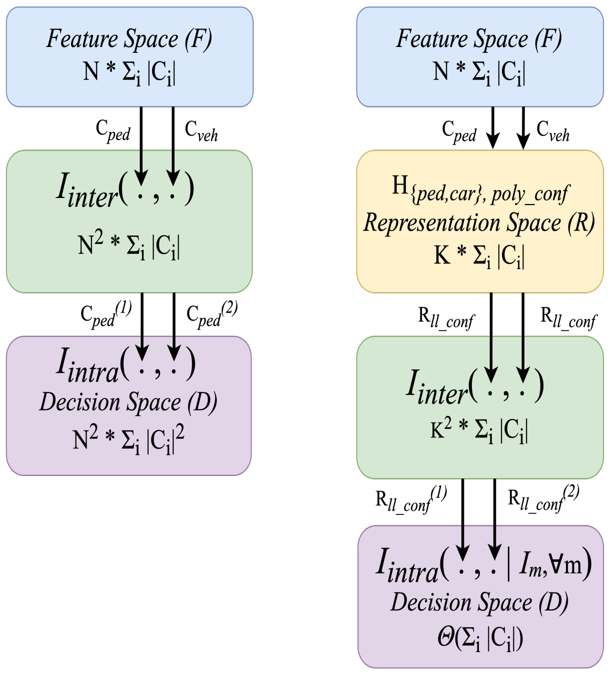


Figure 3.4: Diagrams illustrating the combinatorial complexity of decision making in different contexts. See Section 3.5 for symbol definitions.
Left: In the context of a generic planner. Right: In the context of a planner which employs the DRG. Note that under the DRG, representation space R is captured using $O(N)$ homogeneity operators, resulting in a $\Theta(N) + \Theta(\sum_{C_i \in F} |C_i|)$ design complexity.

interaction checks, resulting in an $\Theta(N) + \Theta(\sum_{C_i \in F} |C_i|)$ algorithm (where $\Theta(N)$ represents the number of homogeneity operators needed and $\Theta(\sum_{C_i \in F} |C_i|)$ is the size of the decision space). We expect that this is a lower bound on the complexity of understanding and reacting to each detected feature instance.

3.6 Verification and Interpretability

There are three properties that make our view of the world easy to verify and interpret. The first property is the relationship graph itself, which is an explicit encoding of the behavioral planner's understanding of the world. Therefore, any behavioral decision must be derived using information in the relationship graph. The second property is inherited from the Lanelet2 design philosophy: Our feature designs (implemented as regElems) always contain the complete physical information that was used to infer the relationships. Therefore, a behavioral decision can always be traced back, through the relationships graph, to physical information that was observed. The last property is a subtle but distinguishing factor: The DRG is not ego-centric, but rather map-centric.

These properties benefit verification and interpretability in the following ways:

1. In the event of an accident, the state of the DRG at any point in time can be inspected offline and run through the behavioral planner to reproduce the sequence of decisions that lead to the accident. Other designs that do have a centralized environment model cannot do this.

2. The state of the DRG can be monitored and interpreted online by a safety driver. If at any point in time the relationship graph does not match what the safety driver observes, they can stop the vehicle before a flawed behavioral decision is made.

3. Because of the DRG's map-centric design, online verification and data sharing about the environment between multiple autonomous agents is possible. This is not possible in most behavioral planning schemes, which have an ego-centric view of the world [25] [26].

3.7 Evaluation

The efficacy of the DRG was evaluated using a stress test in simulation and real world deployment onto the WATonomous research vehicle (Bolty).

3.7.1 Simulated Pedestrian Crowd

In simulation we were able to assess and prove the reduction in behavioral planner complexity analyzed in Section 3.5. Using the CARLA simulator [27] we spawned increasingly large crowds of pedestrians (see Fig. 3.5) and measured the runtime of the DRG augmentation and behavioral planner query. As seen in Fig. 3.6, the runtime scales linearly as the number of pedestrian instances increases. Even with 25 pedestrian instances the planner runtime stays real-time, under 500 ms. This result is expected due to the elimination of intra-class interactions afforded by the DRG as analyzed in Section 3.5.

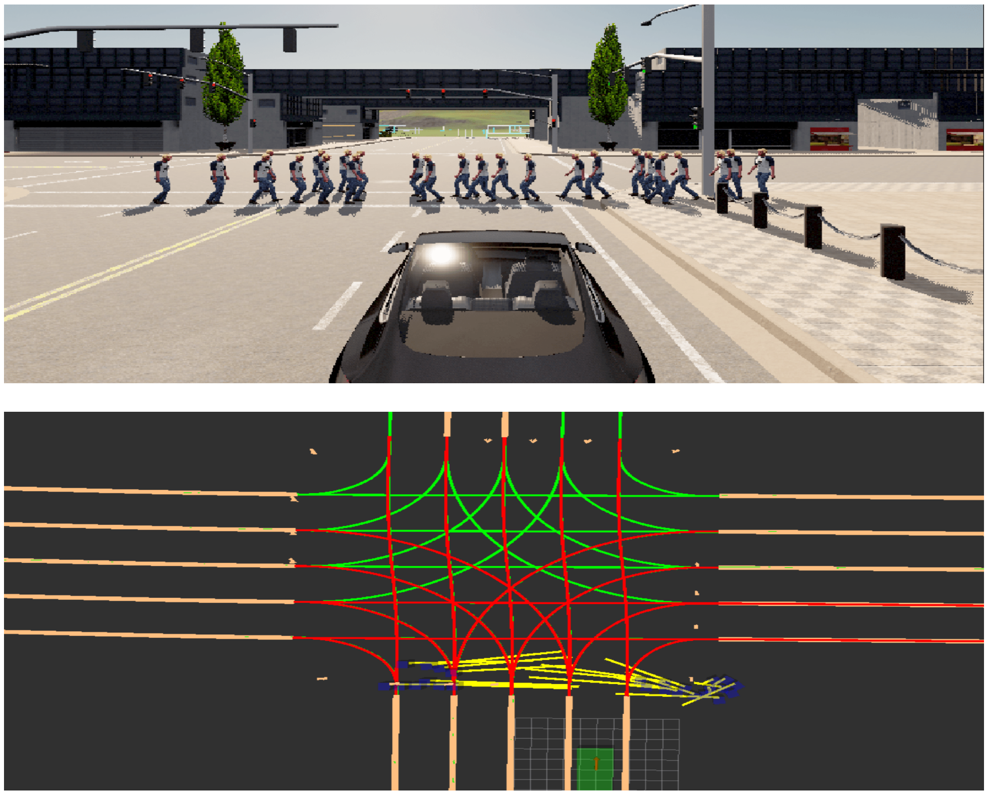


Figure 3.5: The simulated pedestrian crowd used to stress test the DRG and measure the accuracy of the claims made in Section 3.5. The yellow lines are linear state predictions of the pedestrians obtained from the simulator's ground truth, from which lanelet conflicts (indicated in red) are extracted as described in Section 3.4-D

3.7.2 Vehicle Deployment

The DRG was primarily evaluated by deploying the prototype implementation to a research vehicle (Bolty) performing AV taxi routes in a closed course. The lane geometry and relational topology was handcrafted offline using the Lanelet2 toolset. At runtime the AV system encountered stop and intersection signage, blocked roads, pedestrians, and traffic signals. All of these features were observed online (not part of the handcrafted map) and the DRG was augmented in real time, enabling the behavioral planner to obey the traffic rules.

Overall, results of the experiments were promising; the AV system was able to navigate the course and obey all traffic rules, including stops and re-reroutes, by using the DRG.

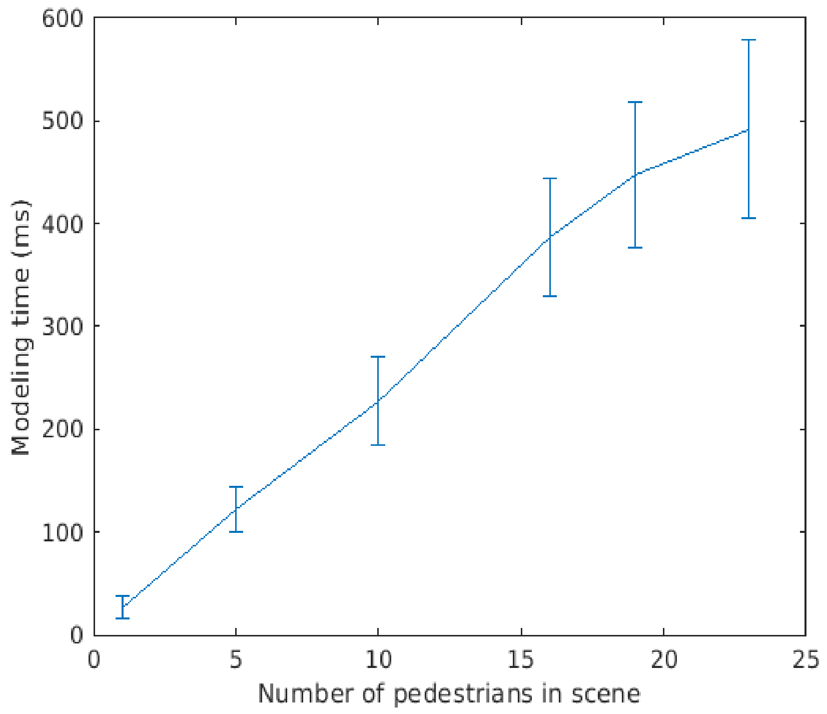


Figure 3.6: Plot showing the linear relationship between the number of observed pedestrian features and the complexity of decision making under the DRG. Execution time in milliseconds is used as the complexity metric.

The full taxi route can be viewed at https://youtu.be/DNZgheT4Y2s?t=153. Additionally, the safety driver was able to continuously monitor the state of the DRG, verifying its correctness or bringing the vehicle to a stop when inconsistencies appeared.

3.8 Conclusion

In this work, we have shown how prior relational maps can be extended to include online observations, and the advantages this approach has for reducing behavioral planning complexity and verification. The proposed Dynamic Relation Graph is a natural and effective step forward towards a unified model of encoding both prior and online information sources. We believe that the proposed scheme will standardize how AV systems encode the richness of relationships in the world, and will allow for greater collaboration in the development of AV decision-making algorithms. Our main contribution in this regard is a detailed insight into how such a relationship graph can be implemented in a real world, closed loop system in the context of a standard perception and mapping schemes.

One of the main advantages of the DRG framework is its flexibility to accommodate many different feature model implementations, which will be the goal of our future work. We plan to enhance our pedestrian model using the set reachability methods presented in [24], and employing learning based approaches where procedural logic falls short. We also aim to perform online verification and consistency checking experiments on a single DRG instance shared between multiple autonomous agents observing the environment.

3.9 Implementation Details

3.9.1 DRG and Lanelet2 API

All Lanelet2 entities (points, linestrings, etc...)	
attributes	Dictionary member variable that stores properties.
Lanelets	
addRegElem	Function that assigns ownership of a regElem to the lanelet.
besides	Function that returns the lanelet(s) to the left and right.
next	Function that returns the successive lanelet(s).
RegElems	
TSRegElem	RegElem subtype that stores traffic sign positions and types.
OccRegElem	RegElem subtype that stores an occupancy grid.
PedRegElem	RegElem subtype that stores an a pedestrian track with historical and predicted states.
.predictedStates	PedRegElem member variable storing a linestring representation of the predicted states.
.isWaiting	PedRegElem boolean function indicating whether a pedestrian is not moving based on its historical states.
.setStop	Mutates a PedRegElem by adding a point on the owning lanelet where the ego vehicle must stop for the pedestrian.
.currState	Member variable which stores the current position of the pedestrian.
DRG	
findInter	Performs Dijkstra's search to find the closest lanelet tagged with a *turn_dir*, where the edge weights are the lengths of lanelets. Then performs a max cost depth first search to find all other intersection lanelets within a distance threshold from the closest lanelet.
signControls	Checks if an intersection sign type applies to a lanelet based on the lanelets's *turn_dir* attribute.
findRoute	Performs Dijkstra's search from *currLl* to the destination over the routing graph.

within	Returns the lanelet(s) which contain the point or linestring operand, optionally accepts a distance operand which expands the search by that radius.
Global variables and functions	
currLl	The lanelet that the ego vehicle is driving in.
intersect	Returns the geometric intersection(s) of two linestrings.
RoutingGraph	
canPass	Boolean function which consumes a lanelet and a traffic participant, returning whether that lanelet is traversable by the given traffic participant.

3.9.2 Stop Signs

The DRG aims to associate a tracked stop sign to the lanelet(s) that it regulates. The objective is to first obtain a set of candidate lanelets based on a search radius[4] around the sign's position, and then to determine which of the candidate lanelets contain the ego vehicle (the controlled candidate). The controlled candidate as well as its adjacent lanelets form the controlled set, to which the sign is associated.

A schematic diagram of the process is shown in Fig. 3.7A. The green rectangle represents the ego vehicle approaching the intersection and the three parallel red lines identify the ego vehicle's lanelet. When the stop sign is observed, all lanelets in a search radius of the stop sign are considered candidate lanelets (highlighted with orange and blue lines). All lanelets that are adjacent to the initial candidate lanelets are also added to the set of candidate lanelets. If the ego vehicle is inside the bounds of a candidate lanelet, that lanelet is the controlled candidate (the red lanelet). Additionally, all lanelets adjacent to the controlled candidate are regulated (the blue lanelet). Therefore, if the ego vehicle changes lanes after the initial construction of the controlled set, it will still be regulated by the stop sign.

In our implementation we use the *TSRegElem* class, a custom subclass of Lanelet2's *RegElem* class, to model the physical stop sign. The *TSRegElem* is constructed using the stop sign's tracked location, so that offline verification of associations is possible. After being constructed, the *TSRegElem* is added to each lanelet in the controlled set via the *addRegElem* API. As a result, the behavioral planner can determine if the lanelet the ego

[4]The search radius is a parameter in this approach. In the proposed implementation, a search radius of 9.2 m is used, based on the maximum lane width of 4.6 m.

is traversing is regulated by a stop sign by looking at the *TSRegElem*(s) that the lanelet owns.

3.9.3 Intersection Signage

This subsection covers associating tracked intersection signs to the virtual lanelets inside of the intersection they control. Signs include: No Right/Left Turn, Right/Left Turn Only and Do Not Enter. In addition, we cover the implications on routing after association.

Our objective is to construct relationships between traffic signs and upcoming virtual intersection lanelets in order to block potential routes that disobey traffic rules. First, all detections of the five mentioned traffic sign types are collected from the tracker. For each tracked sign, we instantiate a *TSRegElem* tagged with the sign's type. Each *TSRegElem* stores the location and type of sign and will be used to model relationships between the sign and the lanelet(s) it regulates.

In order to determine which lanelets are regulated by a *TSRegElem*, we must query the Lanelet2 map. We begin by querying all lanelets adjacent to the ego's current lanelet. Next, we execute a search procedure, starting at each adjacent lanelet, through successor lanelets with the goal of finding all intersection lanelets tagged with a *turn_direction* attribute. This attribute is a custom *a priori* attribute on the virtual intersection lanelets which stores the turn direction (right, straight, left) of the lanelet. The search for intersection lanelets is done in two steps:

1. Perform Dijkstra's search to find the closest lanelet tagged with a *turn_direction* (i.e. the closest intersection lanelet) where the edge weights are the lengths of lanelets. Dijkstra's algorithm was selected for its simplicity and because we know that the search space is a small, constant, number of levels deep.

2. Perform a Max Cost Depth First Search to find all other intersection lanelets within a distance threshold from the closest lanelet.

We now have a set of intersection lanelets in our direction of travel that are candidates to be associated with a sign. In Fig. 3.7B the candidates are drawn in red. By comparing and matching the *turn_direction* attribute with the type of the *TSRegElem*, we add the appropriate *TSRegElem* to the lanelet(s) that are regulated by the sign. For example, a "right" *turn_direction* lanelet would have the "no right turn" *TSRegElem* applied to it (as shown in Fig. 3.7B).

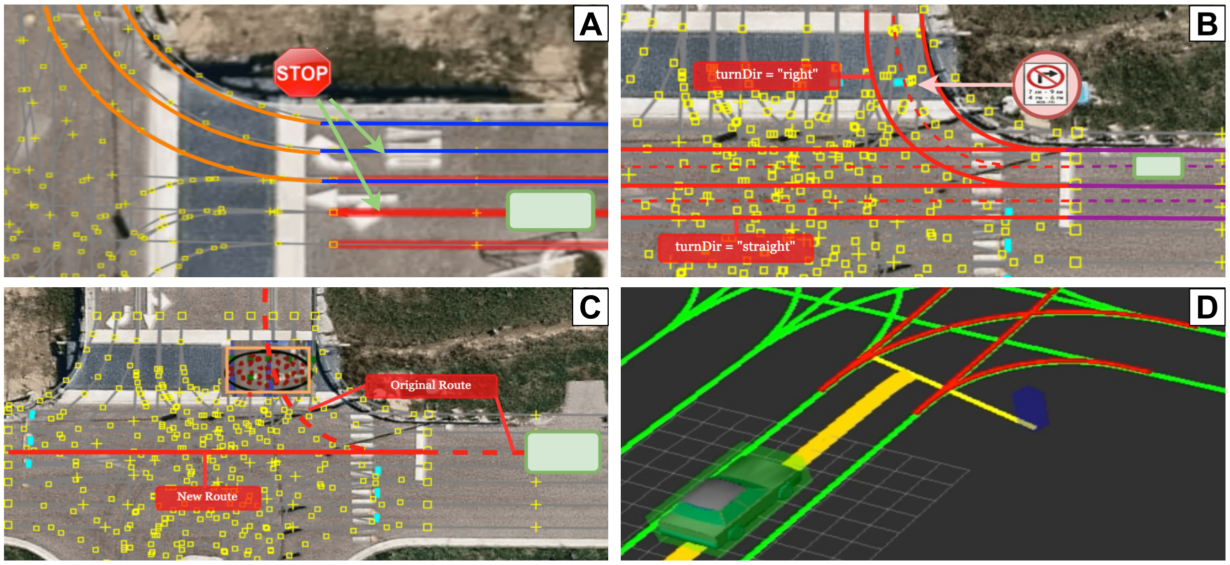


Figure 3.7: Illustrative schematics for (A) stop signage, (B) intersection signage, (C) occupancy, and (D) pedestrian modeling. The schematics are drawn on top of a 4-way intersection at the MCity Course.

By adding each *TSRegElem* to the lanelet(s) it regulates, we have a model of how the traffic signs relate to the prior map. However, we have yet to take this information into account in the context of routing decisions, that is, the Lanelet2 *RoutingGraph* still holds the old lanelet connectivity information. In order for our *TSRegElems* to affect the *RoutingGraph* connectivity, we overrode the *canPass* traversability evaluation of each lanelet to take into account *TSRegElem* owned by that lanelet. However, the current *RoutingGraph* implementation is immutable after being created, since it was designed assuming complete *a priori* knowledge of information relevant to routing. Thus, in order to update connectivity, we must destroy and recreate the entire *RoutingGraph* whenever a lanelet has its *canPass* evaluation changed.

3.9.4 Closed Roads

This subsection focuses on modeling occupancy of the road surface in a fashion that stores both the physical specification of the occupancy and the relationship it has to specific lanelets. Our goal is to determine if there are any obstacles on the road, which lanelet(s) are affected, and what impact this has on routing.

Occupancy grid estimation is done with a VLP-32 LiDAR installed on the ego vehicle. To establish a relationship between the occupancy grid and lanelets, we first align the occupancy grid coordinate frame with the map frame. Next we check if a lanelet is blocked by looping through all grid cells that overlap with the lanelet and check if any cells are occupied. We do this using functions that project from a map coordinate to a grid cell and vice versa.

At this point, the algorithm attempts to find a maneuver around the obstacle by switching lanes. The lateral shift distance needed to avoid the obstacle is based on the leftmost and rightmost bounds of the occupied area. Two outcomes are possible depending on the lateral shift distance and the existence/occupancy of adjacent lanelet(s): (1) There exists an unblocked adjacent lanelet that the vehicle can switch to and avoid the obstacle while staying on the current route, (2) all adjacent lanelets are blocked, and a reroute is necessary (Fig. 3.7A illustrates this case).

In the second case we must mutate the Lanelet2 map and *RoutingGraph* to add information about the blockage. In the map, we create a relationship between the observed physical occupancy and the blocked lanelet(s) using a custom Lanelet2 *RegElem* subclass, *OccRegElem*. An *OccRegElem* object is instantiated with the relevant occupancy cells and added to each of the blocked lanelets. In the *RoutingGraph*, the new *OccRegElem* is interpreted in the same manner as in Section 3.9.3, by overriding the *canPass* method in a custom *RoutingGraph* subclass, returning false if the lanelet owns an *OccRegElem*. Finally, the *RoutingGraph* needs to be destroyed and re-created with the new occupancy information so that a new route can be computed.

3.9.5 Pedestrian Movement

In this subsection we aim to model pedestrian movement and interactions with the prior map. Modeling is done by calculating a lanelet conflict set for each tracked pedestrian, and constructing a relationship between the pedestrian's physical attributes and each lanelet in that conflict set. First, we examine the spatial interactions between the pedestrian's predicted movement (from the tracker) and the lanelets in the map coordinate frame. Lanelets whose polygonal boundary intersects with the predicted movement of the pedestrian are added to the conflict set (see Fig. 3.7D). However, even if a pedestrian is not moving, it could still be in behavioral conflict with a lanelet, e.g. if waiting to cross the street. Therefore, if a pedestrian is not moving and is within a distance threshold from a lanelet, that lanelet is also added to the conflict set. This behavioral conflict is an example of how our environment model explicitly encodes behavioral decisions, simplifying behavioral planning and verification as discussed in Section 3.5.

Once the conflict set has been determined, we calculate and store properties about the relationship between the pedestrian and each conflicting lanelet. These properties are used in the future tasks of behavioral planning and verification. We briefly review these properties and the motivation for including them:

1. Stop Point: the spatial point in the map frame at which the pedestrian's predicted movement intersects with the lanelet's center line; or the point closest to the pedestrian if the pedestrian is waiting. Used in behavioral planning to determine where the ego vehicle should stop for the pedestrian.

2. Predicted/Past Movement: a copy of the information provided by the tracker that was used to calculate this conflict. Necessary for verification.

The C++ instantiation of the relationship is done via a new subclass, *PedRegElem*, of *RegElem* class, which stores the physical properties of the relationship. The *PedRegElem* object is added to each lanelet in the conflict set. This implementation allows for easy access to all relevant *PedRegElem* information in the behavioral planning phase. To determine if a lanelet along the ego's current lanelet path conflicts with a pedestrian, the behavioral planner simply queries each lanelet on that path, asking it if it owns a *PedRegElem*. If so, the behavioral planner can then retrieve the *PedRegElem* object and the *stopPoint* property.

Acknowledgment

We would like to thank Maahir Gupta, Wanda Song, Christopher Mannes, and Jacob Armstrong for their help implementing and evaluating the prototype.

Chapter 4

Action Classification

Acknowledgement: Development of RAD-NetV1 was kicked off in Summer 2021 by Rowan Dempster and Chuan Tian (Ben) Zhang. Work continued on RAD-NetV2 in Summer 2022 by Quanquan Li, Eddy Zhou, and Alex Zhuang, supervised by Rowan Dempster.

4.1 Road Action Detection Network V1 (RAD-NetV1)

4.1.1 Introduction

The perception task in automated driving is to understand the scene around the ego vehicle using camera data. What constitutes "understanding" is an on-going conversation, motivated by what types of information are necessary for safe and effective driving decisions to be made autonomously. Historically, the literature has focused on agent detection and classification tasks which operate on single image frames [7]. State-of-the-art algorithms fit to this task, such as Faster R-CNN [28], have seen large increases in performance over the past decade, and have been adopted into driver assistance systems [29]. A major factor in this success is the availability of large scale datasets such as the KITTI vision benchmark [30].

However, as the field of automated driving matures, the perception research community is shifting their attention towards tasks that represent a deeper understanding of the road scene as a whole [1]. A useful task in this regard is action classification. Separating actions requires the classifier to relate spatial and temporal information across frames. Action classification is also essential for effective autonomous decision making. Especially in urban driving, understanding the intention of other intelligent actors in the scene is crucial to maneuvering safely while not being overly conservative. To this end, the University of Oxford Brookes has recently published the ROAD dataset [1], the first dataset to focus on action classification tasks in the road scene domain.

Agent detection is well studied in the road scene domain. Additionally, action classification is well-studied in the human action domain, for example in the AVA ActivityNet Challenge [31]. In this work we combine these algorithms for effective action detection and classification in the road scene domain, proposing a two-stage action detector which we coin the RAD-NetV1. Our contributions are as follows: (1) We found that the action classifiers first developed for the human action domain can also be trained and perform well on road scenes, even though they contain a wider range of intelligent actors (cyclists, vehicles, etc...). (2) Importantly, we have also discovered that transfer learning from large scale human action datasets (such as AVA) significantly improves performance of RAD-NetV1 in the road scene domain. This is evidence that RAD-NetV1 is able to learn a shared neural representation space that is useful to classify actions in various domains, similar to how humans do.

The remainder of this section is organized as follows: Related works including the ROAD tasks and baseline are discussed in Section 4.1.2, RAD-NetV1 and the algorithms used in its two stages are examined in detail in Section 4.1.3, results and comparative findings are presented in Section 4.1.4, and future work is explored in Section 4.1.5.

4.1.2 Related Work

In this section, we review the ROAD dataset, as well as prior work related to the ROAD tasks which motivated our approach discussed in Section 4.1.3.

ROAD Dataset Tasks and Baseline

The ROAD dataset proposes six tasks in the form of a new annotation database for 18 videos selected from the older Oxford RobotCar Dataset [32]. We only discuss the tasks proposed in the ROAD dataset that are relevant to this thesis.

Agent Detection: The agent detection task is to draw a bounding box around each active agent in the scene, and determine the identity class which the agent falls into. Table 4.1 shows the active agent classes, including vehicles, pedestrians, traffic lights, with their corresponding description. Fig. 4.1 shows the agent class distribution. This task has been well-studied in the literature [33] and there are several useful implementation candidates which can be found on the KITTI vision benchmark leaderboard [34].

Action Classification: Similar to agent detection, except the categories being assigned to are actions, e.g. crossing and overtaking (see Table 4.2 for further details). This task has been studied extensively in the human action domain (see Section 4.1.2). However, ROAD is the first dataset to provide such a wide breadth of action labels for road scenes.

Agent-Action (Duplex) Detection: A combination of the agent detection and action classification tasks, where active agents must be pixel-localized, and have their identity and action categories classified. The agent-action detection task is the main task we work on in this chapter, with the goal of improving on the baseline given in [1].

Alongside the dataset and these tasks, the authors also published a baseline algorithm, coined 3D-RetinaNet. 3D-RetinaNet employs a 3D-CNN architecture by inflating 2D convolutional layers trained on ResNet50 as described in [35]. Focal loss was also used, as introduced in [36] to address the imbalance between foreground and background proposals while training. In order to track detections from 3D-RetinaNet over multiple frames, forming action "tubes", the authors used the incremental approach presented in [37] which relies on solving a linear program to match new detections to existing tubes.

Table 4.1: ROAD active agent classes with description.

Label name	Description
Autonomous-vehicle	The autonomous vehicle itself
Car	A car up to the size of a multi-purpose vehicle
Medium vehicle	Vehicle larger than a car, such as van
Large vehicle	Vehicle larger than a van, such as a lorry
Bus	A single or double-decker bus or coach
Motorbike	Motorbike, dirt bike, scooter with 2/3 wheels
Emergency vehicle	Ambulance, police car, fire engine, etc.
Pedestrian	A person including children
Cyclist	A person is riding a push/electric bicycle
Vehicle traffic light	Traffic light related to the AV lane
Other traffic light	Traffic light not related to the AV lane

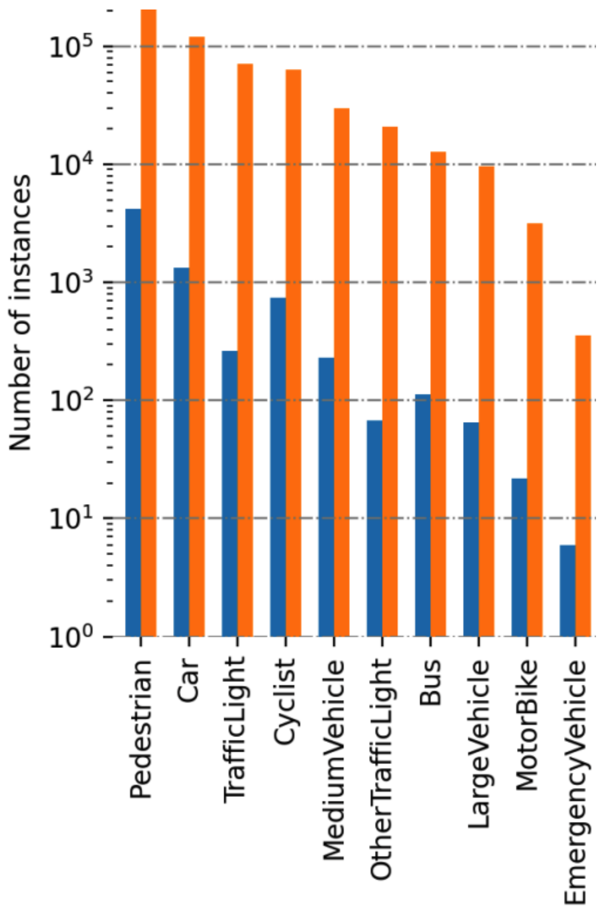


Figure 4.1: Instance counts of various agent classes as specified in [1] (orange is frame-level instances, blue is video-level). Shows an 1000x class imbalance for some agent classes (Pedestrian vs. Emergency Vehicle). Dealing with this imbalance is discussed in Section 4.2.1.

Classification of Human Actions

Datasets

Although the ROAD dataset is the first of its kind to focus on road scene action classification, there exist many long-standing datasets covering the human action domain. These include Moments in Time [38] (three second, single-event videos), AVA [39] (untrimmed 15 minute videos, spatio-temporal action labelling, multiple actions per frame), and Kinetics [40] (trimmed 10 second, single-event videos), among others. The AVA dataset and task

Table 4.2: ROAD action labels, with description.

Class name	Description
Moving away	Agent moving in a direction that increases the distance between Agent and AV.
Moving towards	Agent moving in a direction that decreases the distance between Agent and AV.
Moving	Agent moving perpendicular to the traffic flow or vehicle lane
Reversing	Agent is moving backwards.
Braking	Agent is slowing down, vehicle braking lights are lit.
Stopped	Agent stationary but in ready position to move
Indicating left	Agent indicating left by flashing left indicator light, or using a hand signal.
Indicating right	Agent indicating right by flashing right indicator light, or using a hand signal.
Hazard lights on	Hazards lights are flashing on a vehicle.
Turning left	Agent is turning in left direction
Turning right	Agent is turning in right direction
Moving right	Moving lanes from the current one to the right one.
Moving left	Moving lanes from the current one to the left one.
Overtaking	Agent is moving around a slow-moving user, often switching lanes to overtake
Waiting to cross	Agent on a pavement, stationary, facing in the direction of the road.
Crossing road from left	Agent crossing road, starting from the left and moving towards the right of AV.
Crossing road from right	Agent crossing road, starting from the right pavement and moving towards the left pavement.
Crossing	Agent crossing road.
Pushing object	Agent pushing object, such as trolley or pushchair, wheelchair or bicycle.
Traffic light red	Traffic light with red light lit.
Traffic light amber	Traffic light with amber light lit.
Traffic light green	Traffic light with green light lit
Traffic light black	Traffic light with no lights lit or covered with an out-of-order bag.

specification is most closely fit to the ROAD tasks, and is widely used as a benchmark to train and compare action classification algorithms.

View Based Approaches

Most algorithms that process video data do not look at the complete video at once, but rather process short *views* of the complete video scene. The term *view* refers to a temporal (and possibly spatial) crop of a long sequence of frames. Papers of note in this category include the Action Transformer proposed by Rohit et. al in [41]. The Action Transformer ingests short (3 second) views centered at keyframes in the AVA dataset, first passing them through a 3D convolution layer[1], and then a Region Proposal Network (RPN) [28] to extract initial candidate regions in the feature map space. The head of the network classifies and regresses each proposal using a transformer architecture, where the feature map around the proposal make up the memory (keys and values) of the attention mechanism, and the

[1]Inflated 3D convolutions (I3D) are a common building block in video action recognition networks [42, 43, 44, 45], although excluded for brevity in this review.

proposals themselves are used as the queries. The transformer is multi-headed and multi-layered (see [46]) with the motivation that self-attention will provide context from other actors and objects in the clip when the query is updated in each layer, helping to discern complex actions.

The issue with views is that some classifications are not possible given only a few seconds of frames to work with. The SlowFast design proposed by Feichtenhofer et. al [47] as well as Long-Term Feature Bank (LFB)s proposed by Wu et. al [48] provide two methods of at least partially removing this constraint. In SlowFast networks, two streams of temporal information are processed in parallel, one sampled at a high resolution (the "fast" stream), and another at low resolution (the "slow" stream). The fast stream captures detailed motion of actors in the scene, but uses a smaller CNN channel capacity and has a short view length, whereas the slow stream has a large CNN channel capacity and captures large temporal strides over the video. The two processing steams are connected via lateral connections from the fast pathway to the slow pathway, allowing fine motion details to augment the long temporal strides of the slow pathway.

LFBs [48] are similar to the slow stream of the SlowFast network, in that the bank accumulates information from a high temporal resolution stream that views only a few seconds from a 3D CNN. However, whereas SlowFast's slow steam processes its own low temporal resolution steam of information as it is fused with the high resolution stream, the LFB only accumulates information from the short term 3D CNN backbone.

With the efficacy of the SlowFast as a video feature extractors proven by its first place finish AVA ActivityNet Challenge 2019 [31], efforts in the past few years have focused on different network head architectures which compute actor-actor and actor-context relationships to separate harder to classify actions. A successful approach is ACAR-Net [3], in which RoI (from an off-the-shelf detector e.g. R-CNN) aligned feature maps from SlowFast are first concatenated and passed through a 1x1 convolution to model actor-context relations as done in the Actor-Context-Relation Network [3]. However ACAR-Net goes further by introducing a High-order Relation Reasoning Operator (HR^{2}O) which computes relationships between pairs of the actor-context relations themselves. The author's argue that an operator such as HR^{2}O is needed to discern complex actions in the human domain where the action of one actor is dependent on another actor's context (e.g. riding in a car versus driving it). In Section 4.1.4 we examine the question of if such an operator is needed in the road scene domain.

Complete Video Approaches

Recently there have been efforts in natural language processing tasks to apply the transformer architecture to large corpuses, which the quadratic self-attention computation complexity has previously prohibited. Works such as Longformer [49] and BigBird [50] propose methods to sparsify the self-attention mechanism, which they view as a graph formed by fully connected token nodes. This is achieved by introducing relational inductive biases (see [51]) to the graph structure including (1) limiting self-attention to a constant sized set of local neighboring tokens (Longformer and BigBird) possibly with dilation (Longformer), (2) attending to a constant sized set of random other tokens (BigBird), and (3) attending to a set of externally added global tokens (Longformer and BigBird).

In the video understanding domain, where frames can be viewed as tokens in the "video corpus", these sparsification techniques have also enabled self-attention to be applied over never before possible lengths of videos. The Video Transformer Network (VTN) proposed by Neimark et. al [52] is one such work, directly applying the Longformer sparsification technique to frame tokens produced by any 2D feature extractor (Neimark et. al found the Vision Transformer (ViT) [53] backbone most effective). VTN is able to attend to frames in the video that are most important for the classification task, effectively ignoring "noise" frames.

4.1.3 Methodology

Overall, our methodology behind RAD-NetV1 is to combine state-of-the-art agent detectors with strong classifiers for human actions, striving to improve on the ROAD tasks discussed in Section 4.1.2. In the remainder of the section we will discuss: (1) An overview of the two stages of RAD-NetV1; (2) our approach to object detection and classification using CenterNet; (3) our approach to action classification, comparing the SlowFast + Feed Forward Network (FFW) and SlowFast + ACAR algorithms; (4) integration details of RAD-NetV1's two stages.

RAD-NetV1

Unlike the baseline 3D-RetinaNet, RAD-NetV1 uses a two-stage approach to solve the agent detection, action classification, and duplex detection tasks. This decision was made because agent detection and action classification are individually well-studied tasks, and the strengths of existing solutions can be combined in a two-stage approach.

The first stage of RAD-NetV1 is object detection, for which we use CenterNet [54], the backbone of one of the top performing algorithms (CenterTrack [55]) from the KITTI vision benchmark that only uses RGB images. The CenterNet algorithm produces a list of bounding boxes with agent classifications, effectively solving the agent detection task.

The second stage of RAD-NetV1 deals with action classification, a task that requires special attention and treatment of the temporal dimension to separate action classes. Therefore, a different sort of network architecture is necessary. Instead of the single temporal-resolution 3D-CNN that the baseline employed, we use the more recently developed SlowFast architecture [47] as our video feature extractor. SlowFast employs and fuses multi-resolution temporal streams of frames, combined with non-local [35] blocks (see Section 4.1.2). For the head of the network (which generates the action-class scores), we experimented with two options in order to discern the effect of ACAR-Net: (1) A FFW prediction head as done in the original SlowFast paper [47]; (2) further relational-context reasoning via the ACAR-Net head design [3].

In order for RAD-NetV1 to solve the ROAD tasks, information from both the first and second stage is necessary. Specifically, the stage-two action classifier needs bounding box priors from the stage-one detector. Thus, as shown in Fig. 4.2, when a new *keyframe* is processed it is first fed through the detector to produce bounding box priors. The box priors are then used as the input, along with contextual frames around the keyframe, to the action classifier which provides action-scores on a per-box basis.

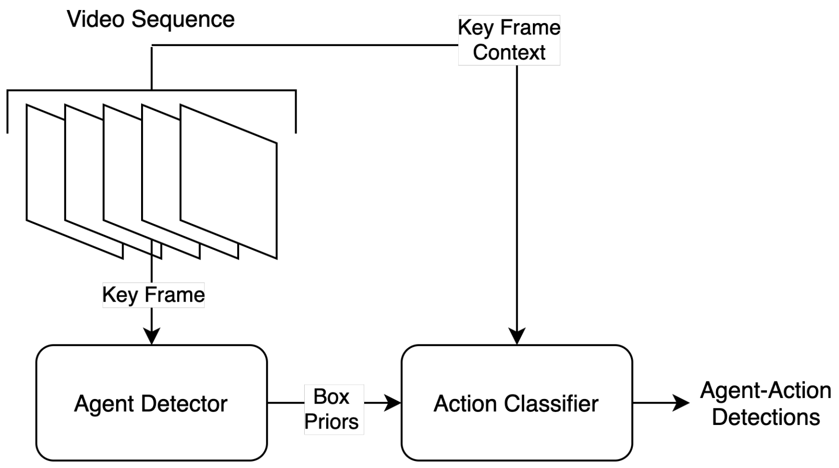


Figure 4.2: Schematic of the RAD-NetV1 agent-action detection pipeline as described in Section 4.1.3

Stage I: Object Detection

Acknowledgement: The object detection work described in this section was done by Chuan Tian (Ben) Zhang.

For the first stage of the RAD-NetV1 pipeline, we applied state-of-the-art object detection algorithms on the ROAD dataset. CenterNet is used and performed well on the MS-COCO 80-class object detection and classification dataset. We performed experiments using CenterNet under two settings: training on the ROAD dataset from scratch and training on the ROAD dataset with initial weights from a pretrained model used for the MS-COCO dataset. Because of the difference in the number of classes (11 in ROAD v.s. 80 in MS-COCO), some head weights are dropped in the transfer learning experiment.

To set up the experiments, we converted the ROAD dataset annotation format into the COCO annotation format, and adapted an existing implementation of the CenterNet algorithm to work with the transformed dataset. To evaluate the results, we used the COCO API, which implements the standard intersection-over-union (IoU) calculations and evaluates the agent labels with the equality condition. This is consistent with the definition of ROAD, which uses 0.5 as the IoU threshold to determine the correctness of bounding boxes.

The learning rate schedule used is identical to the one used for the "ctdet_coco_dla_2x" experiment in the original CenterNet implementation [56].

Stage II: Action Classification Algorithms

Starting out, we had two hypotheses regarding action classification. First, we hypothesized that algorithms which performed well in the human action domain are also well-suited for the road scene domain, even though there is a wider range of intelligent actors, i.e. cyclists, vehicles, in road scenes. Furthermore, we hypothesized that transfer learning can be applied to leverage the existing large scale human action datasets, such as AVA, and improve the action classifier's performance in the road scene domain.

To test these hypotheses we experimented with two action classification algorithms: SlowFast backbone + FFW head and SlowFast backbone + ACAR-Net head. Both of these algorithms performed well on the AVA human action dataset.

We also experimented with different weight initialization settings. Each algorithm was trained in a "scratch" setting where all weights were initialized randomly and in a "pretrained" setting where the Slowfast backbone weights were initialized using a model trained on the AVA dataset. Note that in the "pretrained" setting the head network weights were not transferred as the head network architecture changes based on which set of action classes are being predicted. Both settings were then trained (or fine tuned) on the "train-1" split of the ROAD dataset videos, using the annotated action classes found in Table 4.2, and evaluated on the corresponding "val-1" split. During training, the ground truth bounding box priors were used as inputs along with 30 (SlowFast + FFW) or 45 (SlowFast + ACAR-Net) frames of view context. An overview of the training process is shown in Fig. 4.3.

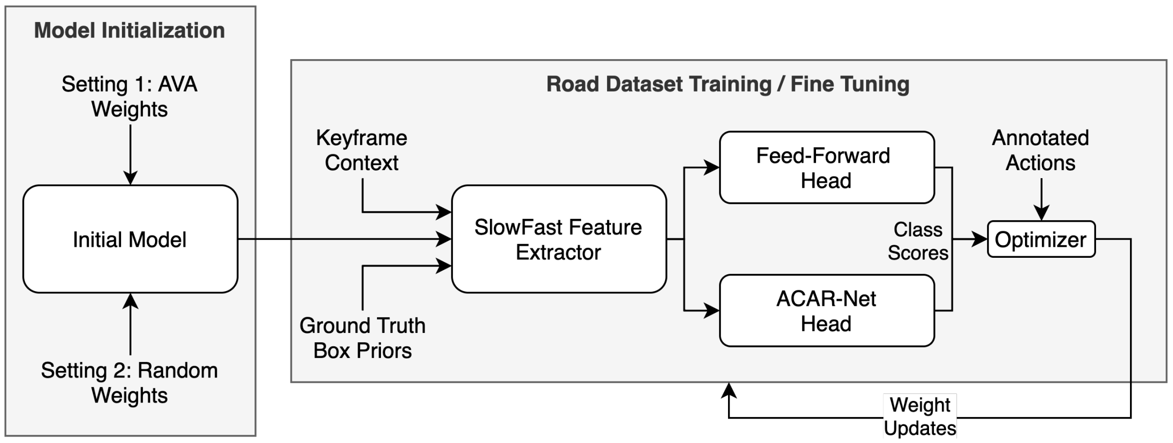


Figure 4.3: Schematic of the action classifier training pipeline as discussed in Section 4.1.3. Note that the two heads are trained separately, and are only shown together here for compactness.

The same optimizer, learning rate schedule, and data augmentation techniques were used as in the publicly available implementations of SlowFast [57] and ACAR-Net [58].

Integration Details

In order to calculate the mAP metrics which are comparable to the 3D-RetinaNet baseline, the two stages discussed previously were integrated by providing the bounding boxes generated by the detector to the action classifier as priors. To this end, the detector first generates a JSON file containing detections on each frame, uniquely keyed by the video and frame IDs. Then, when the action classifier evaluates a keyframe, the unique key for that frame is generated again and box priors and retrieved from the JSON file. The keyframe context is retrieved from the dataset as usual, and passed through the action classifier along with the predicted bounding boxes in the keyframe. This integration process allowed us to evaluate RAD-NetV1 in the same way that the 3D-RetinaNet baseline was evaluated. These results are discussed in the next section.

4.1.4 Results and Discussion

Overall, the agent detector and action classification algorithms performed strongly in their respective stages (achieving 62 mAP and 34 mAP individually). When the two stages were integrated, RAD-NetV1 performance dropped to 25 mAP, one point below the baseline 3D-RetinaNet. These initial results are nevertheless promising, and we believe that RAD-NetV1 will outperform the baseline after further refinement (see Section 4.1.5). The remainder of this section discusses in detail results of the CenterNet, SlowFast, ACAR-Net algorithms.

CenterNet Results

The two CenterNet experiments (training from scratch and transfer learning) were trained for 11 epochs each and they appear to have drastically different performance characteristics. The experiment where the CenterNet model is trained from scratch showed promising decrease in loss initially. However, both the training and validation losses worsened shortly after. The transfer learning experiment, on the other hand, exhibited expected behaviour with the training loss decreasing in each epoch.

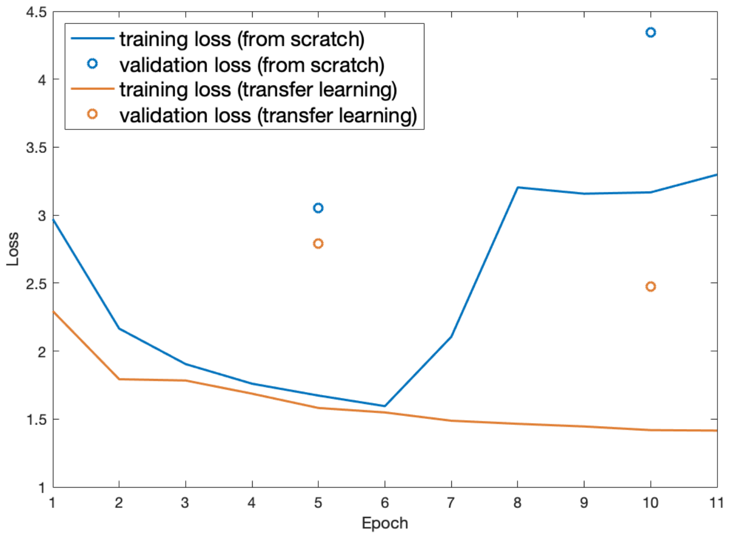


Figure 4.4: Training and validation loss progression the CenterNet experiments. The validation loss is evaluated every 5 epochs.

Fig. 4.4 shows the loss at each epoch of training. The experiment where training started from scratch reached its minimum training loss at epoch six. Then the training loss degraded rapidly. This may be due to the learning rate being set too high. The transfer learning experiment, on the other hand, shows a steadily decreasing loss. Due to time and resource constraints, the training for both experiments was stopped after completing epoch 11.

Type	IoU	TL	FS	Baseline
AP	0.50	0.620	0.549	0.445
AP	0.75	0.321	0.240	N/A
AR	0.50-0.95	0.487	0.430	N/A

Figure 4.5: Object detection and classification: best average precision (AP) and average recall (AR) values evaluated on the val-1 split of the ROAD dataset. Two experiments are shown: training CenterNet from scratch (FS), training CenterNet with transfer learning (TL). The 3D-RetinaNet baseline result as described in [1] is also shown for reference.

Fig. 4.5 shows the best (among all epochs) average precision and average recall values for various experiments. The transfer learning experiment clearly outperforms the experiment where the training is done from scratch. Both transfer learning and training from scratch surpassed the baseline results.

SlowFast and ACAR-Net Results and Comparison

Three different implementations are compared here: (1) SlowFast + FFW (Scratch) has the SlowFast feature extractor trained from scratch (random weight initialization, see Fig. 4.3) and a FFW head; (2) SlowFast + FFW (Pretrain) is the same as SlowFast + FFW (Scratch) but with the SlowFast feature extractor weights initialized from a model trained on the AVA Dataset; (3) SlowFast + ACAR (Pretrain) is the same as SlowFast + FFW (Pretrain) but with the head from ACAR-Net used instead of the FFW head.

Fig. 4.6 shows the mAP performance on the "val-1" split versus number of epochs trained on the ROAD dataset for each implementation. Note that these metrics were generated using the ground truth box priors from the annotation database, in order to evaluate the efficacy of the action classifier in isolation.

These mAP metrics are comparable to those reported in the AVA leaderboard [31]. This confirms our first hypothesis, that action classification algorithms developed for the human action domain also perform well in the road scene domain.

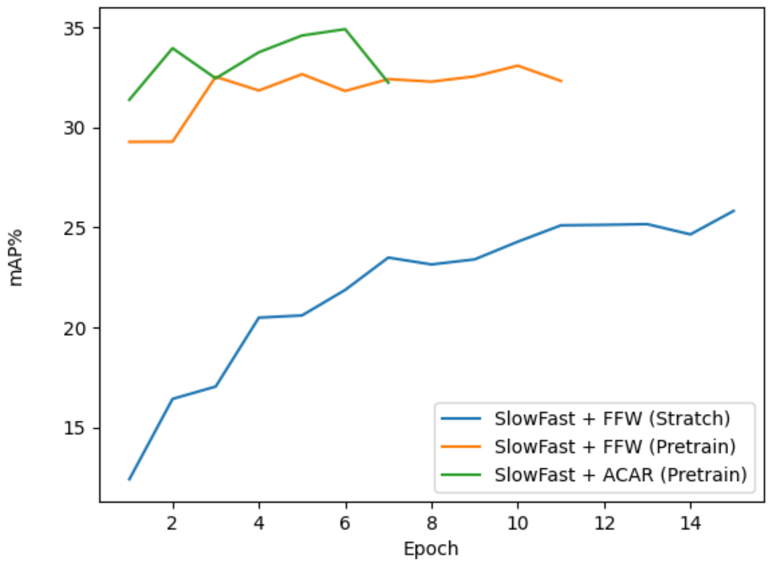


Figure 4.6: mAP versus number of epochs trained on the ROAD dataset for each implementation variant of the action classifier described in Section 4.1.4

Importantly, there is a profound increase in performance when using transfer learning from the AVA human action dataset (see Fig. 4.6 blue vs. orange plots). In fact, the performance is nearly saturated after a single epoch of fine tuning on the ROAD dataset (see Fig. 4.6 orange plot). These findings are evidence that RAD-NetV1 is able to learn shared representations of actions between different domains (human and road scene). This domain transfer phenomena is well-documented in the image classification task, but to our knowledge these findings are first documented evidence in the action classification task.

Lastly, we also note that the ACAR head was able to outperform the FFW head. However the gain was smaller than reported for the AVA Dataset (3 points versus 5 points), leaving effectiveness of ACAR's relational-context design in road scenes up for discussion.

To get a more-fine grained interpretation of these results, Fig. 4.7 shows the performance of each implementation variant split by each action class. Looking at the per-class differences, we see that pretraining the SlowFast feature extractor improved performance across the board. Comparing the FFW and ACAR heads, ACAR noticeably improves performance (by about 10 points) on the "crossing" (directional, waiting, etc...) related actions. These actions are visually similar, and can only be separated by looking at the context of the scene and relation to other actors, areas that the ACAR-Net was designed for.

We also note that the more difficult (crossing, turning) and most difficult (overtaking) actions have a large temporal scope compared to the easier actions (being stopped, or moving away). Implementation of a LFB (see Section 4.1.2), or variants on the long-term attention mechanism like VTN (see Section 4.1.2) may improve performance in these difficult classes.

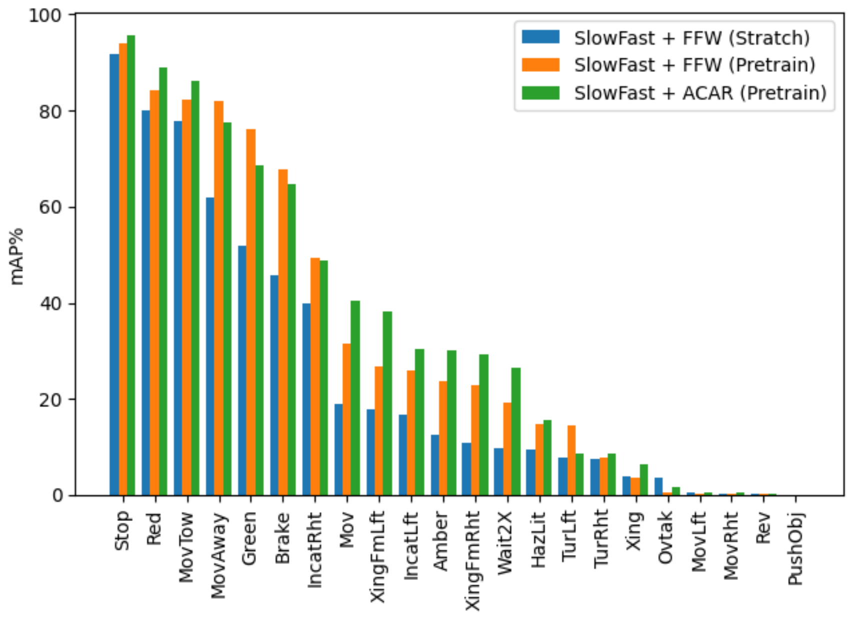


Figure 4.7: mAP versus action class for each action classifier variant discussed in Section 4.1.4

RAD-NetV1 Results and Discussion

Quantitative Results: To assess the quantitative performance of RAD-NetV1, the ground truth box priors used for the action classifier training were replaced with box priors from the object detector. Object detector priors with confidence scores less then 0.3 were filtered out. The introduction of imperfect box priors decreased the mAP of the action classifier by nine points, to 25 mAP for RAD-NetV1.

Qualitative Results: Qualitative results produced by RAD-NetV1 can be viewed here: https://www.youtube.com/watch?v=asylL5mFisw, using one of the videos from the val-1 split. Fig. 4.8 shows example keyframes taken from this video which emphasize why action detection work encouraged by the ROAD dataset is crucial for automated driving systems.

Although our initial implementation of RAD-NetV1 lags behind the baseline's 26 mAP performance on the action task, refinements (discussed below) are available that will close this gap and improve on the baseline.

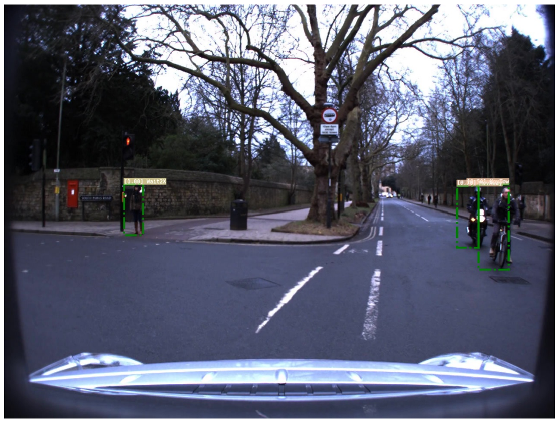
4.1.5 Conclusions and Future Work

This section described our original two-stage approach, RAD-NetV1, to the action detection and classification tasks posed by the ROAD dataset. The proposed agent detector and action classifier stages both perform well individually. However, the performance degrades when the two algorithms are combined for RAD-NetV1 to solve the duplex task. Refinements are discussed in the next section to address this drop in performance.

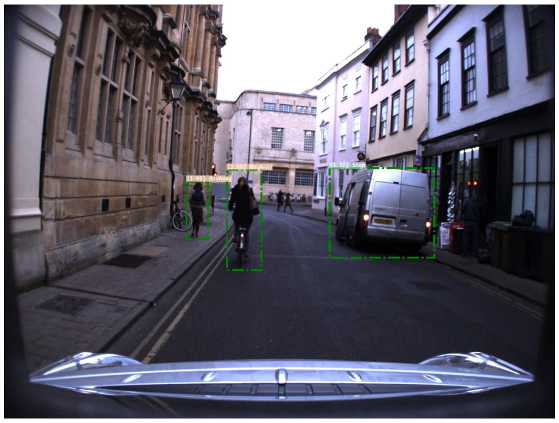
Specifically, the concept of "active-agent" detection needs to be addressed. In the ROAD dataset, only active-agents, defined as agents that are actively performing an action, are labeled. However, the CenterNet detector (and other common detectors) are not designed for such a discriminative task, as inactive and active agents may appear identical in a single frame. Thus, the drop in performance when using the detector boxes as action-classification priors instead of the ground truth boxes can be partially explained by a large number of false positive active-agent detections being present in the detector boxes, but not the ground truth boxes. Improvements on active-agent detection is discussed in Section 4.2.1.

Furthermore, the current single keyframe box prior approach is ill-suited for the difficult actions with large temporal-scope, as the actor of interest may occupy different regions of the contextual frames, relative to the keyframe. Thus, adjustments the RoI Align algorithm from [59] (which both SlowFast and ACAR-Net use to splice the prior boxes out of the CNN feature maps) are necessary to support varying box priors across the contextual frames, as discussed in Section 4.2.2.

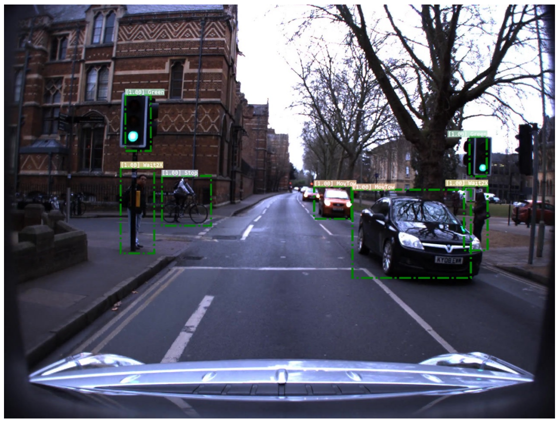
Lastly, the imbalance of the action classes in the ROAD dataset also needs to be addressed (see Fig. 4.9 for class instance counts). One simple solution is to over-sample the low-instance classes while training, which is usually done via a denser data augmentation regime for those classes. Another approach is to use a loss function which is less sensitive to classes that the network is already well-fit to. Focal loss [36], as used in the baseline



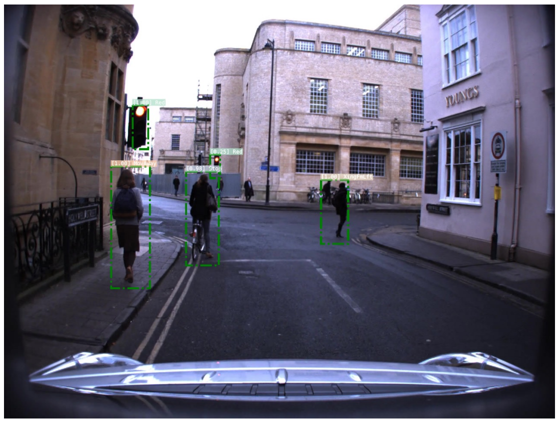
(a) In this frame the ego vehicle is beginning to make a left turn in the presence of a pedestrian. Deciding to yield or proceed is dependent on the actions of the pedestrian, which in this case RAD-NetV1 classified correctly as “Waiting to Cross”.



(b) A second scene where the safety of the ego vehicle is dependent on whether the idling van is stopped, or indicating that it will be entering the ego’s lane. In this case RAD-NetV1 correctly classifies the van as “Stopped”, and the ego proceeds.



(c) This frame presents a more complicated scene where different types of intelligent actors (pedestrians and cyclists) as well as occlusions (of the cross walk for the pedestrian on the right) are present. Deciding to proceed safely through the intersection requires the actions of the two pedestrians, and the bike to be classified, which in this RAD-NetV1 does correctly.



(d) Another challenging scene for deciding how close to the stop line the ego vehicle should be as it approaches the correctly classified red light. The information required here is that the pedestrian on the left is moving away, instead of crossing in front of the ego vehicle. Also, the cyclist is stationary on top of the stop line, forcing the ego vehicle to stop well before the line.

Figure 4.8: Example outputs of RAD-NetV1 on keyframes in a “val-1” split video from the ROAD dataset. These output frames are chosen to emphasize the importance of action classification in road scenes. Best viewed using software with zoom ability to see labels.

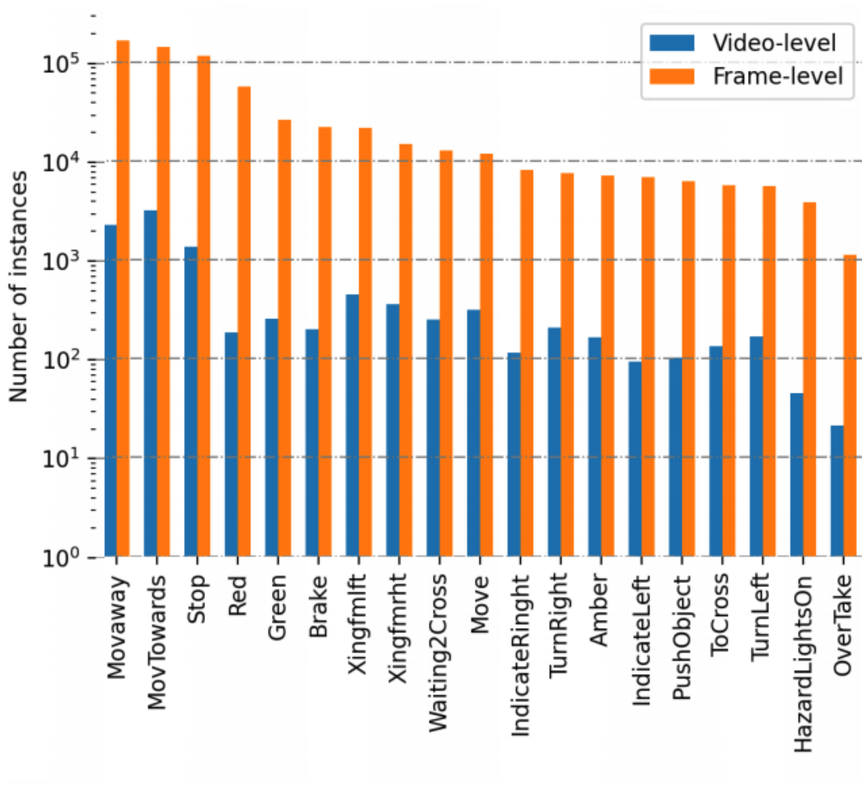


Figure 4.9: Instance counts of various actions as specified in [1]. Shows an 100x class imbalance for some action classes (Move Away vs. Overtake). These imbalances correlate with performance drops as seen in Fig. 4.7.

3D-RetinaNet, is a good candidate in this regard and its implementation is discussed in Section 4.2.3.

Although not discussed in the next section, future work could also include experimenting with convolutional-free video architectures, such as TimeSformer [60], which are built on purely self-attention computations over the space and time dimensions. Such designs, although radically different than the 3D-CNNs discussed in this report, have shown state of the art performance on the Kinetics Dataset, and can be applied to longer views.

4.2 Road Action Detection Network V2 (RAD-NetV2)

In light of the shortcomings of RAD-NetV1, various improvements were investigated, as identified in Sections 4.1.5.

4.2.1 Active Agent Detection

Acknowledgement: The active agent detection work described in this section was worked on collaboratively with Quanquan Li.

A major drawback of RAD-NetV1 was that the first stage detection algorithm had a large number of false positives of active agents, because an inactive vehicle can appear practically identical to an active one, see Fig. 4.10 for an example.

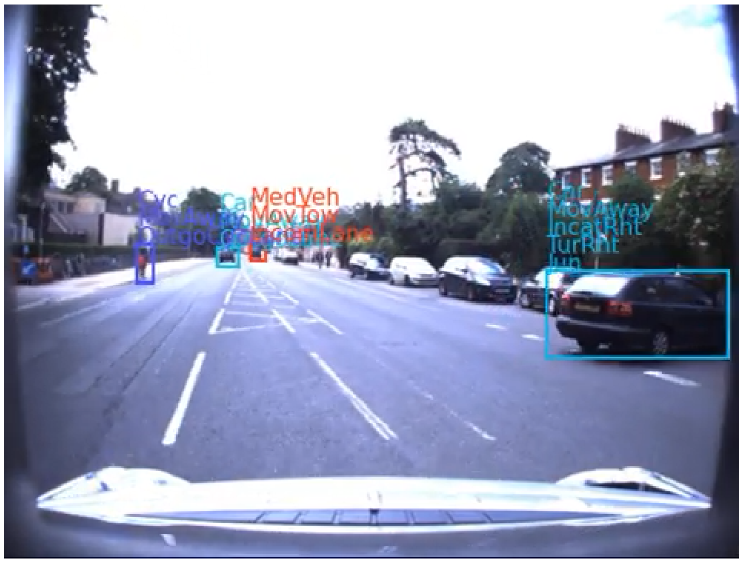


Figure 4.10: Here we see a datapoint from the ROAD dataset which contains both active (which have labels) and inactive (no labels) vehicles. Note how both kinds of vehicles (active and inactive) have a similar appearance, and thus are difficult for an appearance based detector to discriminate between.

Creating an Active-Inactive Agent Dataset

In order to discriminate between active and inactive agents, at the minimum a dataset with labels of inactive agents is required. To generate such a dataset, a common object detector trained on the COCO dataset is used in conjunction with the ROAD dataset. Then, for each frame in the ROAD dataset, let R_i be the set of labeled (active) agents in frame i of the ROAD dataset, and let O_i be the set of all agents detected by the object detector. Then, the set of inactive agents is O_i/R_i, where R_i set membership is established by an IoU of greater than 0.2. The resulting “psuedo-labels” of inactive agents are shown in Fig. 4.11.

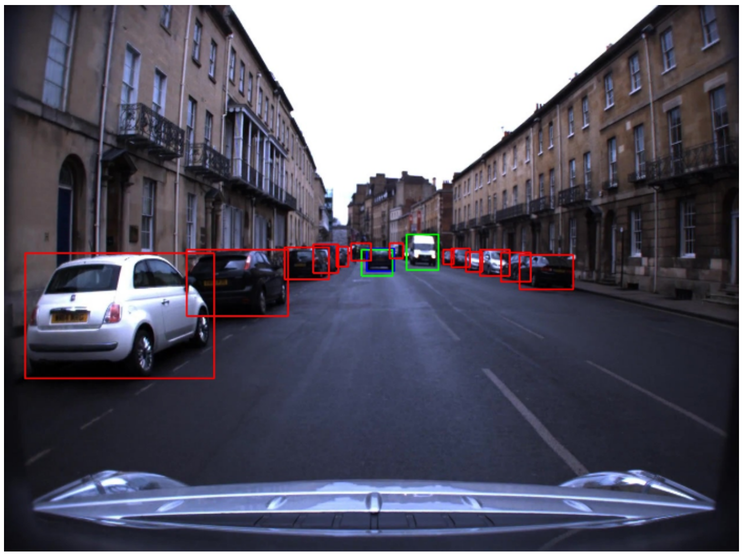


Figure 4.11: Example frame from the created active-inactive agent dataset. Agents in green are any active class (as in the original ROAD dataset), whereas agents in red are the newly created pseudo-labels of inactive agents.

Average Precision (AP) results from training on the active-inactive dataset versus training on the original ROAD dataset are presented in Table 4.3

As shown in Table 4.3, using the created inactive agent dataset for training helps the object detector differentiate bewteen active and inactive agents, yielding an improvement of 7.1 points on AP@0.5-0.95 and 6.6 points on AP@0.5, using the original ROAD validation set.

Model	AP@0.5-0.95	AP@0.5
Original ROAD Dataset	30.9	61.3
Inactive Agent Dataset	38.0	67.9

Table 4.3: Comparison of validation accuracy for training on the ROAD dataset versus the created active-inactive dataset.

Super-category Classes

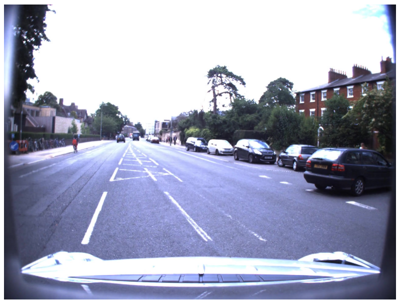
As shown in Fig. 4.1, there is an issue with agent class imbalance in the ROAD dataset, biasing the training of the detector towards the common classes and harming mAP. However, note that specific class identities are not used for the final task of action classification, only the bounding boxes are. Thus, super-classes can be created for classes that are visually similar. For example, merging MedVeh, LarBeh, Bus, and EmVeh into a new class "Vehicle" alleviates the imbalance of EmVeh shown in Fig. 4.1. In total we are left with 6 new super-classes: Ped, Vehicle, Cyc, Mobike, TL/OthTL, Inactive. The new class distribution is well balanced and improves detection mAP substantially as shown in Table 4.4. However, this is not a fair comparison since the task has fundamentally changed. To get a fairer comparison the final task of end-to-end action classification can be assessed using the bounding boxes from the 10-class dataset vs. the new 6-super-class dataset. Results for that are also presented in Table 4.4.

Super-category Class Balance	Detection mAP	Action mAP
	67.9	23.8
✓	80.4	24.9

Table 4.4: Comparison of validation accuracy for the detection model with and without super-category class balance.

As seen in Table 4.4, super-class based detection does in fact improve the end-to-end action mAP by 1.1 points.

(a) The first frame from which optical flow is derived.


(b) The second frame from which optical flow is derived.

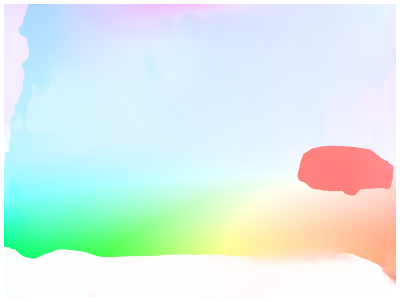

(c) The optical flow obtained from the two images.

Figure 4.12: A visualization of the inputs and outputs of the RAFT network. In the output (c) the active agent can be clearly identified, unlike in the inputs (a) and (b).

Optical Flow Signals

As seen in Figs. 4.10 and 4.11, the active and inactive classes generated in Section 4.2.1 are visually similar, and thus do not provide a strong visual discrimination signal for the detector to work with. A better signal of agent activeness is whether or not the agent is moving across time in the scene. Optical flow is a method for computing such a signal for each pixel in sequence of two images. To compute an optical flow image for each frame in the ROAD dataset, we used RAFT: Recurrent All Pairs Field Transforms for Optical Flow [2]. See Fig. 4.12 for an example of the optical flow images generated. In order to apply pretrained convolutional filters to the flow maps, a color wheel representation is used where the direction of the flow vector determines the color and the magnitude of the flow vector determines the intensity.

In order to fuse the appearance information from the RGB images, as well as the motion information from the optical flow image, either a shallow fusion or a deep fusion architecture can be used (see Fig. 4.13). In shallow fusion, the flow map is first concatenated to the RGB image before being passed through the ResNet backbone. Although simple, the main issue with the shallow fusion architecture is that ResNet filters pretrained on ImageNet and COCO cannot be used because the number of channels in the input tensor has changed. Alternatively, a deep fusion architecture can be employed where concatenation is delayed until after the individual application of the ResNet and Feature Pyramid Network (FPN) layers. The benefit of the deep fusion design is that the pretrained ResNet filters can be used individually on each of the 3-channel RGB and colorized flow maps, which increases accuracy as shown in Table 4.5.

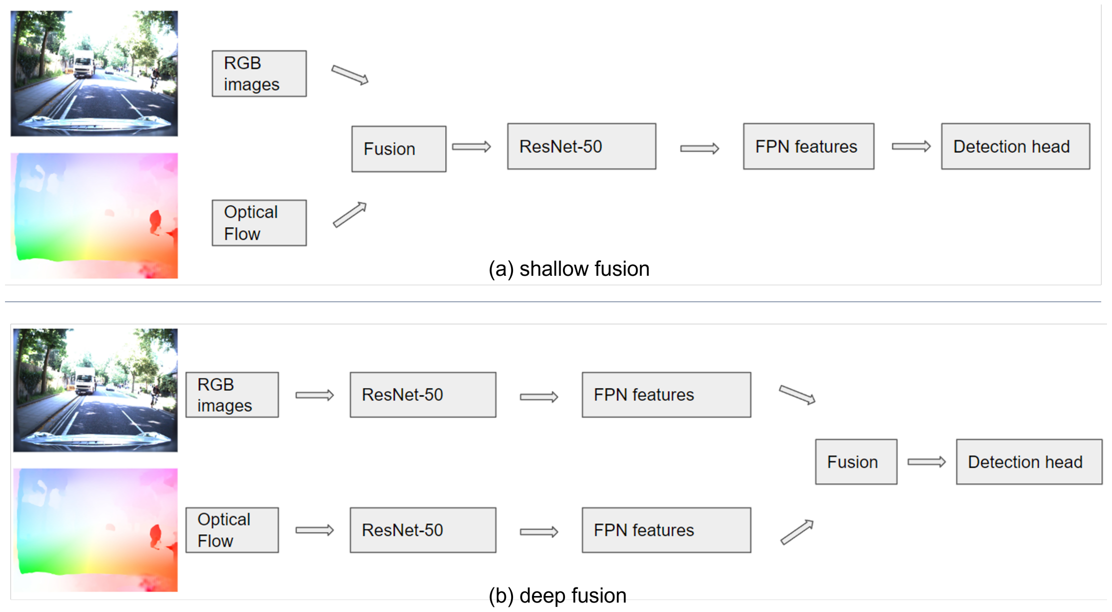


Figure 4.13: Comparing the shallow fusion architecture, which does not allow for usage of pretrained ImageNet and COCO filters, to the deep fusion architecture, which does allow for pretrained filters.

Model	AP@0.5-0.95
Baseline (RGB only)	56.2
Shallow fusion	52.2
Deep fusion	59.9

Table 4.5: Comparison of validation accuracy under different optical flow fusion strategies.

As seen in Table 4.5, introducing the flow map signal does in fact boost performance by 3.7 points when using the deep fusion strategy. However, the shallow fusion actually decreases performance by 4 points. This comparison shows the importance of using the pretrained ImageNet and COCO filters, afforded by only the deep fusion strategy.

Furthermore, as seen in Table 4.6, the deep fusion strategy continues to improve performance by 0.9 points when applied on-top of the inactive agent dataset and super-class improvements discussed above.

Model	AP@0.5-0.95
Baseline (RGB only)	80.4
Deep fusion	81.3

Table 4.6: Comparison of validation accuracy for the best performing optical flow model. The baseline includes using the inactive agent dataset and the super-class balancing strategy discussed above.

4.2.2 Dealing with High Motion Scenes

RAD-NetV1 inherited its keyframe based design from ACAR-Net, which was targeted at low-motion scenes found in the AVA Dataset. In low-motion scenes, spatial information from a single frame suffices to describe the locality of the action across the entire temporal view. However, given the high ego and non-ego motion of agents in the ROAD dataset, spatial information in the keyframe does not accurately localize the action in frames far from the keyframe (see Fig. 4.14 for an example).

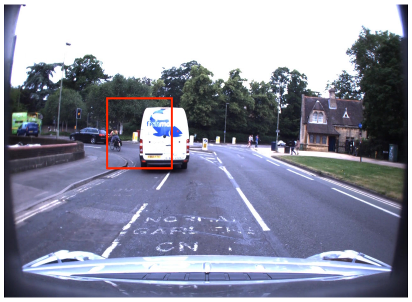
Dynamic ROI (Tube) Based ACAR

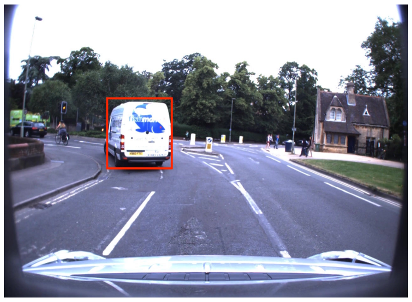
Acknowledgement: The tube-based RoI-Alignment implementation described in this section was worked on collaboratively with Eddy Zhou.

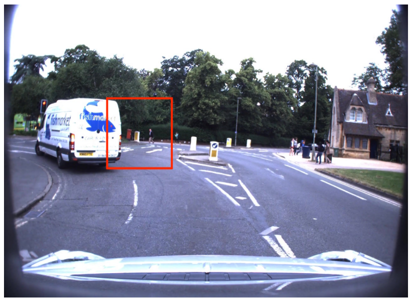
To mitigate this spatial mismatch across the view, we changed the input definition of the action classification model to take a sequence of detections across the view (i.e. a track, or tube) for each agent in the keyframe. In order to implement this change we needed to introduce a tracker on-top of the detection step, and change the ACAR-Net Head architecture to support multiple RoIs for each agent we are classifying the action of.

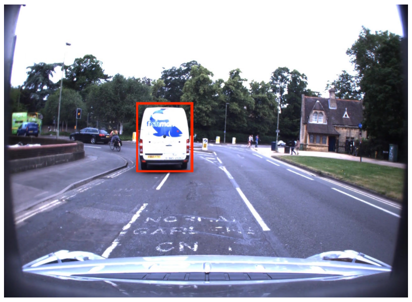
For the tracker we chose OC-Sort [61] for its state of the art performance. OC-Sort takes the detections as input, and associates a *tube_uid* with each detection that serves to associate detections over time. We note that OC-SORT is especially capable of handling occluded observations, which happen often in road scenes. In order to eliminate the possibility of different object classes in the same track, we separately track agents from different agent classes.

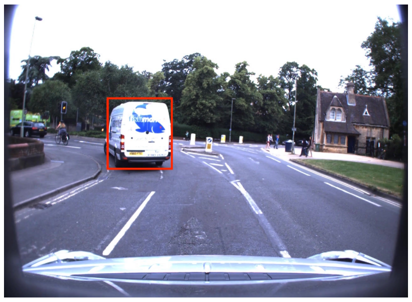
An illustration of the architecture change to the ACAR-Net Head can be found in Fig. 4.15. As seen in 4.15(a), ACAR-Net originally performed a single RoI-Align operation using the keyframe RoI, after the slow and fast feature tensors are already temporally


(a) First frame in frame-based ACAR-Net input


(b) Middle (keyframe) frame in frame-based ACAR-Net input


(c) Last frame in frame-based ACAR-Net input


(d) First frame in tube-based ACAR-Net input


(e) Middle (keyframe) frame in tube-based ACAR-Net input

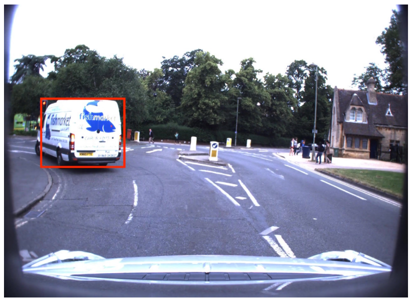
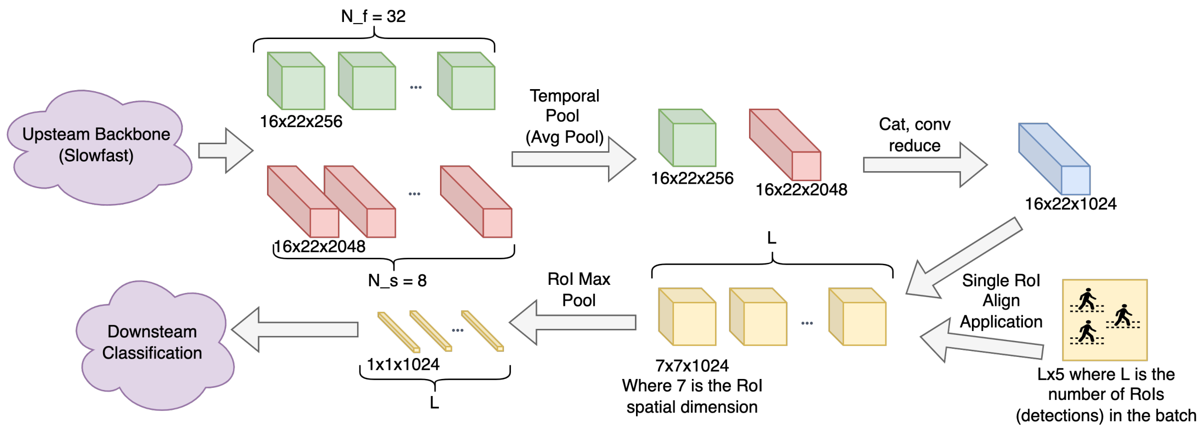

(f) Last frame in tube-based ACAR-Net input

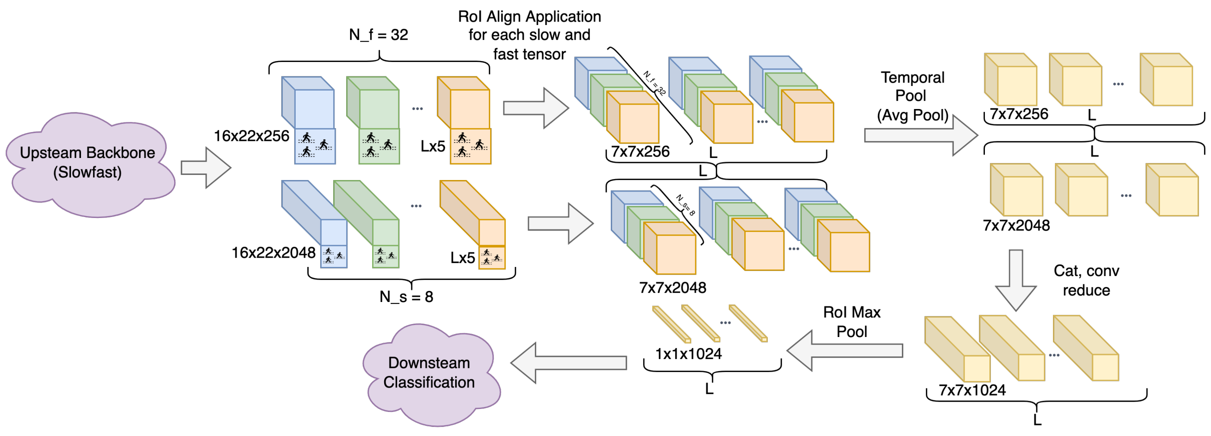
Figure 4.14: Difference between inputs to frame-based vs. tube-based ACAR-Net. The top row (a-c) is the frame-based input, obtained from just the detector. The keyframe detection (b) does not provide good spatial localization of the agent across the entire temporal view (a) and (b). In contrast, the bottom row (d-f) is the tube-based input based on the detector and the OC-Sort tracking algorithm. In combination with the modifications made to to the ACAR-Net model discussed in Section 4.2.2, the tube does provide good spatial tracking across the view, improving action classification accuracy as presented in Table 4.7.

pooled. As motivated by Fig. 4.14, we replaced this keyframe based RoI design with the design found in 4.15(b). In our new design a RoI-Align operation is performed for each fast and slow frame-tensor, using the RoIs from the tube of detections that correspond to each frame-tensor, before temporal pooling is applied. This way, the correct spatial features are extracted from each frame-tensor. This modified RoI-Align approach is outlined in Alg. 3.

The results of the tube based ACAR-Net design are shown in Table 4.7. As seen in the table, the tube-based ACAR-Net design increases performance by 2.5 points on top of the active agent improvements described in Section 4.2.1.


N_f = 32
Upsteam Backbone (Slowfast)
16x22x256
16x22x2048
N_s = 8
Temporal Pool (Avg Pool)
16x22x256
16x22x2048
Cat, conv reduce
16x22x1024
Single RoI Align Application
Lx5 where L is the number of RoIs (detections) in the batch
L
7x7x1024 Where 7 is the RoI spatial dimension
RoI Max Pool
1x1x1024
L
Downsteam Classification

(a) The (original) keyframe RoI based ACAR-Net architecture.



(b) The redesigned tube RoI based ACAR-Net architecture.

Figure 4.15: Comparing keyframe based ACAR-Net with tube based. In (a), only a single RoI-Alignment operation is performed, after temporal pooling. This design works well for low motion datasets such as AVA, but may introduce spatial noise in high motion datasets such as ROAD. In (b) we redesigned the ACAR-Net Head to apply RoI-Alignment for each frame-tensor before temporal pooling. This method provides better spatial alignment of the agent's features across the entire temporal view, but requires a tube of detections as input.

Algorithm 3 Tube RoI-Alignment Adapted to SlowFast

```
Input: slowFeats, fastFeats, bb {list of bounding boxes per frame}
Output: Feature Tensor with Encoded Tracks through Tube ROI-Alignment
1: $roi\_fast\_feats = []$
2: $roi\_slow\_feats = []$
3: $alpha = \frac{temporal\_len(fastFeats)}{temporal\_len(slowFeats)}$
4: for idx, fastFeat ∈ temporal_enumerate(fastFeats) do
5:     if $(idx + 1)/alpha = 0$ then
6:         $rois = ROIAlign(slowFeats[idx], bb[idx])$
7:         $roi\_slow\_feats.append(rois)$
8:     end if
9:     $rois = ROIAlign(fastFeat, bb[idx])$
10:    $roi\_fast\_feats.append(rois)$
11: end for
12: $fast\_feats = TemporalAvgPool(roi\_fast\_feats)$
13: $slow\_feats = TemporalAvgPool(roi\_slow\_feats)$
14: $feats = Concatenate(fast\_feats, slow\_feats)$
15: return $feats$
```

Network	Val 1
Original ACAR-Net + Optical Flow Detector	27.8
Tube ACAR-Net + Optical Flow Detector	30.3

Table 4.7: Improvements from tube ACAR-Net design, we see a 2.5 point improvement when using the tube based design discussed in Section 4.2.2. All ablations are reported on validation split 1 since it has the most even class distribution.

4.2.3 Dealing with Class Imbalance

Acknowledgement: The focal loss experiments described in this section were worked on collaboratively with Alexander Zhuang.

As examined in Section 4.1.5, there are magnitudes of difference between the number of action class instances. The baseline 3D-RetinaNet used focal loss to deal with the imbalance, however RAD-NetV1 did not, using vanilla Binary Cross Entropy loss instead. In RAD-NetV2 we adopted the focal loss implementation from 3D-RetinaNet [1] (originally

Algorithm 4 Sigmoid Focal Loss

Input: sigmoid activated *preds*, one hot encoded *labels*, number of positive samples *npos*, focusing parameter γ, weighting factor α

Output: Computed *loss*

```
1: $l1 = binary\_cross\_entropy(preds, labels)$
2: $\alpha\_weight = \alpha * labels + (1 - \alpha) * (1 - labels)$
3: $pt = preds * labels + (1 - preds) * (1 - labels)$
4: $\gamma\_focus = \alpha\_weight * ((1 - pt)^{\gamma})$
5: $loss = (l1 * \gamma\_focus).sum()/num\_pos$
6: return $loss$
```

from RetinaNet [36]). The loss implemented is presented in Alg. 4.

The results of switching to focal loss (with $\gamma = 2.0$ and α as the inverse frequency of each class) are presented in Table 4.8. As shown in the figure, focal loss effectively deals with the action class imbalance shown in Fig. 4.9, which is consistent with the results presented in [1], overall boosting performance by 3.0 points. This experiment was run only on the action classification stage, using the ground truth box priors from the ROAD dataset.

Loss Function	Val 1
Sigmoid Loss	34.7
Sigmoid + Focal Loss	37.7

Table 4.8: Improvements from focal loss design, we see a 3.0 point improvement when using the focal loss design discussed in Section 4.2.3. All ablations are reported on "val-1" split since it has the most even class distribution.

4.2.4 Final RAD-NetV2 Results

The final design of RAD-NetV2 with all the improvements from Section 4.2 is shown in Fig. 4.16.

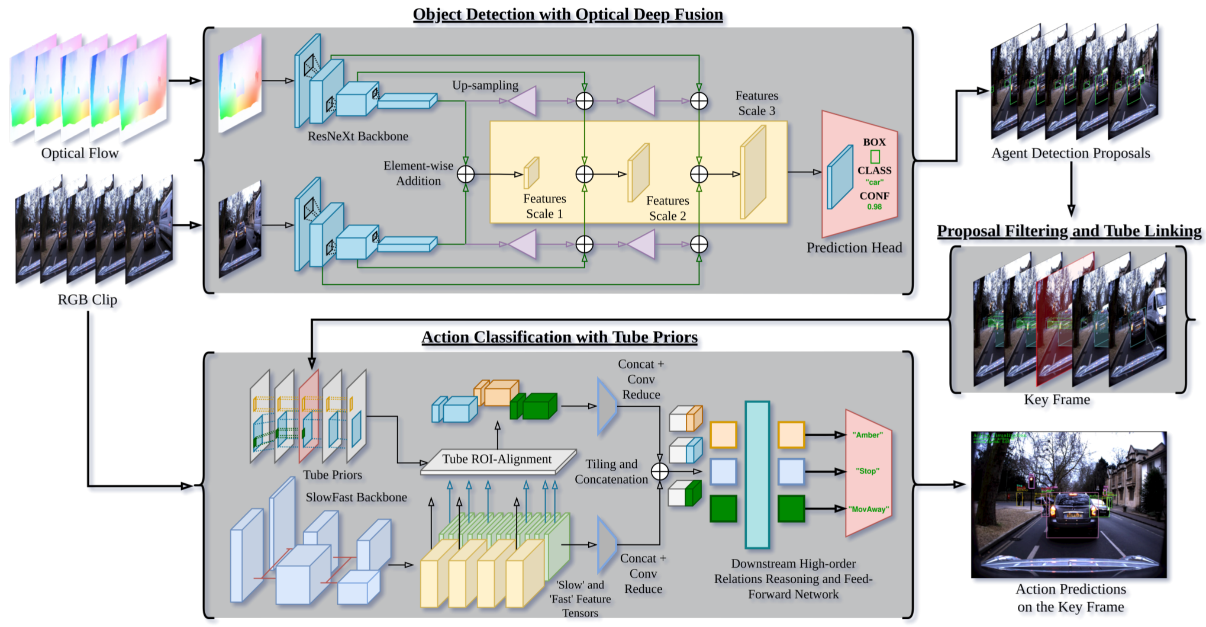


Figure 4.16: RAD-NetV2 system architecture. Given a set of frames in a clip, their optical flow is estimated using RAFT [2]. Both clips (RGB and flow) are sent frame-by-frame into the object detector where each RGB frame and its corresponding optical flow frame are encoded and summed up at multiple feature scales. We found that utilizing a pretrained RGB backbone on 3-channel optical flow improved model predictions and accuracy. Following detection, an online object tracker is used to link detections into tubes. Tubes present in the key frame of the clip are then fed into our action classifier, which encodes agent tubes through a novel RoI-Alignment procedure that takes advantage of the inherent structure of feature encodings outputted by SlowFast. Following encoding, we adopt [3]'s higher-order relations reasoning to compute the attention between each encoded agent tube and the other agents present in the clip. The result is the action predictions of agents present in the key frame.

We focus our performance evaluations on the ICCV 2021 ROAD dataset. Our results show an improvement over the baseline on average, with a significantly improved frame and video-mAP on the "val-1" split. Full results are shown in Table 4.9. Qualitative results

of the entire RAD-NetV2 pipeline (active agent detection, OC-Sort based tracking, and modified ACAR-Net Head classification) can be found at
`https://youtu.be/QQF8zlfEtlI`.

Model	Val 1	Val 2	Val 3	Avg
3D-RetinaNet (frame-level)	26.2	**11.7**	21.2	19.7
Ours (frame-level)	**30.3**	10.0	**22.9**	**21.1**
3D-RetinaNet (Online)	17.0	11.4	**14.6**	14.3
Ours (Online)	21.1	11.4	13.4	15.3
Ours (Offline)	**24.0**	**11.73**	13.87	**16.52**

Table 4.9: Frame and video-mAP scores on the ROAD dataset. We compare frame-mAP@0.5IOU and video-level mAP@0.2IOU following the 3D-RetinaNet baseline. Experimentally, it is observed that these results may fluctuate by ∼1%. For our online results, Tube-ACAR considers a temporal extent of 32 frames, so we are able to perform box interpolation within gaps of 32 frames or less.

4.3 Deployment of RAD-NetV2 to Bolty

To enhance the scene understanding capabilities of Bolty, and drive in a less conservative manner, RAD-NetV2 was deployed to the perception stack, and integrated with the environment model.

4.3.1 Perception Integration

In order to deploy the RAD-NetV2 to the ADS, the network's inputs need to be generated in a real-time manner. Refer to Section 2.3 for a refresher on the data pipeline design, the relevant pieces are copied to Fig. 4.17 in more detail.

In brief, the history of 2D observations is added to each track. The observations are then time synchronized (see Alg. 5) with a buffered temporal view from the camera stream, and both the temporal view and the tube of observations are run through the action classification model. The result is a binary classification for each class, and the class with the highest score (if it is over 0.5) is added to the action classification history for that track.

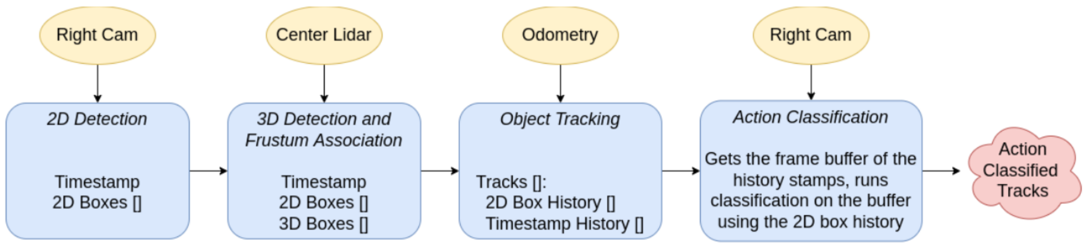


Figure 4.17: Data pipeline used to deploy RAD-NetV2 to Bolty. The resulting action classified tracks are sent to downstream modules (environment modeling), where they are used as input to behavioral planning decisions.

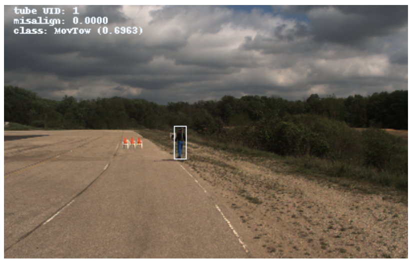
Algorithm 5 Detection - Frame Buffer Time Sync

```
Input: Frame $timestamps$, observed $bboxes$ (both sorted from most recent to oldest)
Output: For each frame $timestamp$, finds the temporally closest $bbox$ and $time\_delta$
 1: $match\_i = 0$
 2: $matched\_bboxes = []$
 3: $time\_deltas = []$
 4: for $stamp \in timestamps$ do
 5:     while TRUE do
 6:         $match\_delta = abs(stamp - bboxes[match\_i].stamp)$
 7:         $next\_match\_delta = abs(stamp - bboxes[match\_i + 1].stamp)$
 8:         if $match\_delta < next\_match\_delta$ OR $match\_i + 1 >= len(bboxes)$ then
 9:             BREAK
10:         end if
11:     end while
12:     $matched\_bboxes.append(bboxes[match\_i])$
13:     $time\_deltas.append(abs(stamp - bboxes[match\_i].stamp))$
14: end for
15: return $matched\_bboxes, time\_deltas$
```

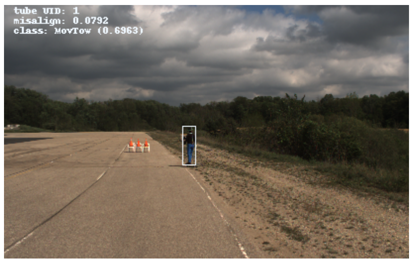
Integration Results

Real world testing was carried out at the Waterloo Region Emergency Services Training and Research Centre (WRESTRC). The entire test scenario can be viewed at `https://youtu.be/KdrLhPqi4Rk`, which shows a pedestrian stopped on the side of the road. Fig. 4.18 presents the qualitative results of the deployment, showing the input and output of the deployed action classification model. As seen in the figure, the spatial footprint is tracked

(a) First frame in the temporal view given to ACAR-Net



(b) Middle (keyframe) in the temporal view given to ACAR-Net



(c) Last frame in the temporal view given to ACAR-Net

Figure 4.18: Figure showing the visual and RoI inputs to the ACAR-Net model, as well as the top classification score across the entire tube, and the temporal misalignment for the frame.

across the temporal view to provide a better visual description of the region of interest, as discussed in Section 4.2.2.

Table 4.10 presents the quantitative results of the deployment, comparing the keyframe based model against the tube based model. As seen in the table, the Tube ACAR-Net model produces the correct classification *stopped* the majority of the time, whereas the Keyframe ACAR-Net model misclassifies the majority of the time. This can be attributed to the poor region of interest input to the Keyframe ACAR-Net model, as analyzed before in Section 4.2.2.

Model	Stop	MovTow	MovAway
Keyframe	0.15	0.00	0.85
Tube	0.54	0.26	0.20

Table 4.10: Ratio of classifications over the test scenario.

4.3.2 Integration with Environment Model

The only change to the environment model is that incoming tracks of agents now have a history of action classifications attached to them. The environment model uses this additional information to better discern whether or not to create relations between the tracks and other entities in the DRG.

Algorithm 6 Handle Pedestrian With Action

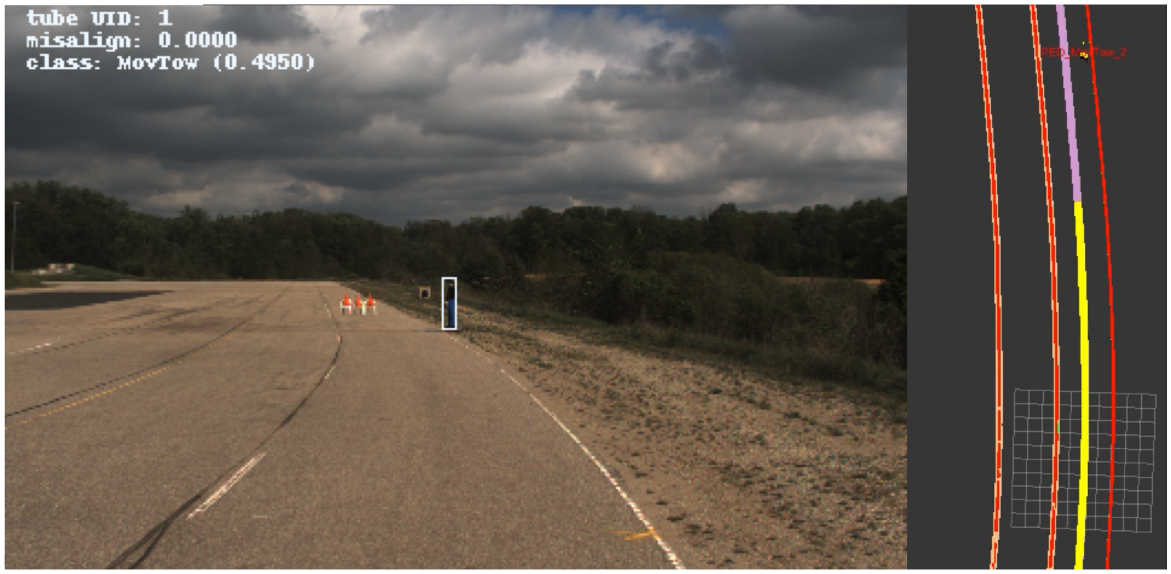
Input: Tracked *ped* with history of action classifications, history time t to consider, set of safe actions $safe\_set$.

Output: Boolean indicating if the pedestrian should be inserted into the conflict graph.

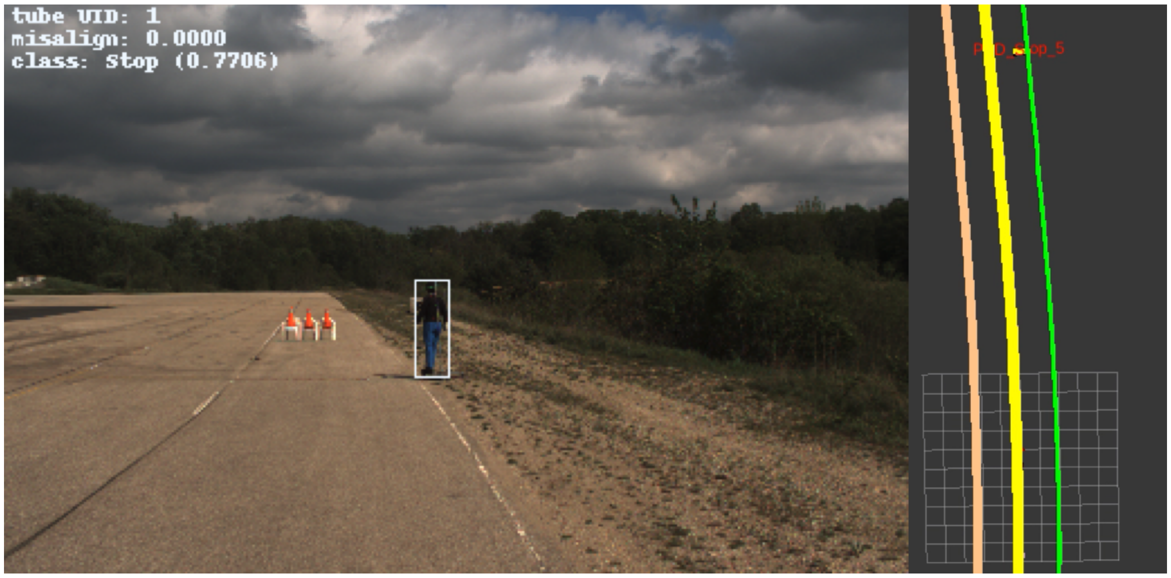
```
1:  $n\_interfere = 0$
2:  $n\_total = 0$
3:  for $action \in ped.action\_hist$ do
4:      if $time :: now() - action.stamp > t$ then
5:          BREAK
6:      end if
7:      $n\_total + = 1$
8:      if $action.label \notin safe\_set$ then
9:          $n\_interfere + = 1$
10:     end if
11: end for
12: return $\frac{n\_interfere}{n\_total} > 0.5$
```

For example, refer to the original modeling of pedestrian tracks and their potential conflict relations with lanelets as laid out in Alg. 2. In that original implementation of the algorithm, the decision of whether or not a pedestrian conflicts with lanelet was based on (1) whether the linear trajectory prediction of the pedestrian intersects with the lanelet and (2) if the pedestrian is stationary, whether it is within some radius of the lanelet. It is clear that these geometric heuristics will lead to a large number of false positive conflict relationships being instantiated. Not all pedestrians whose predicted linear trajectory enters a lanelet will in-fact enter that lanelet - perhaps there is simply a t-intersection in the sidewalk - and certainly not all pedestrians standing close to the road will jump out into traffic (see Fig. 4.19(a) for an illustration of this failure case).

The action classification history helps to limit these false positive cases by classifying the intentions of pedestrians based on their appearance descriptions, rather than geometrical descriptions. Alg. 6 implements a less conservative version of Alg. 2 by taking advantage of these appearance based intention classifications. Fig. 4.19 compares the resulting DRG state under the conservative Alg. 2 to the resulting DRG state under the improved Alg. 6 (using $safe\_set = \{Stop, Wait2X\}$). In the figure we see that the new implementation does improve decision making, allowing the AV to proceed past the stopped pedestrian.



(a) Shows the overly conservative behavioral decision for the ego vehicle to stop when there is no visual indication that the pedestrian will interfere with the ego vehicle. This flawed behavior can occur when only using the geometric heuristics specified in Alg. 2, or when using an inaccurate action classifier as discussed in Section 4.3.1.



(b) Shows the reasonable ego behavior of driving past a stopped pedestrian, enabled by accurate action classification as specified in Alg. 6.

Figure 4.19: Comparing different ego behavior in the stopped pedestrian scenario under different DRG implementations.

Chapter 5

Trajectory Planning and Control

5.1 Introduction

In the SAE AutoDrive Challenge I [62], SAE and GM set forth guidelines for vehicle dynamics metrics that should be obeyed by AVs to ensure comfortable and safe urban driving. These guidelines include limits on longitudinal acceleration and jerk, as well as lateral acceleration in the vehicle's body frame. The guidelines are to be adhered to as the AV accomplishes the Dynamic Driving Task (DDT). Generally, the DDT can be seen as progressing towards a goal state in a feasible and efficient manner. In the context of urban driving, feasible means without collision and while obeying traffic rules, and efficient means operating near the speed limit.

5.1.1 Motivation

There are several papers in the literature that address the real-time obstacle avoidance problem using optimal control techniques. However, all have drawbacks that make them ill-suited for the problem presented above. The authors of [63] apply a nonlinear MPC method, similar to the one presented in Section 5.3, to the obstacle avoidance problem. However, the controller must be instructed by the perception system on which direction to avoid the obstacle (left or right). This assumption is unsatisfactory because the directional decision in itself needs to consider the dynamics of the vehicle to know if a maneuver to the left or right of the obstacle is most optimal. The authors of [64] take an interesting approach towards integrated obstacle avoidance by representing obstacles as a potential field in the cost formulation. However, this paper, as well as [65], [66], and [67] have the issue that the avoidance maneuver controller (which they separate from the lower level actuator controller) is based on a particle representation of the vehicle. Therefore, these approaches have the same issue as using a higher level planner: the planned states generated by the lower fidelity planning model may not be dynamically feasible and therefore are unsafe.

In [68], a cubic polynomial is utilized to describe the lateral deviation from the reference line. Similarly, the authors of [69] and [70] use a sigmoid function. However, none of these papers provide an analysis of why these functions were chosen in the context of their OCP objective function. In contrast, the scheme proposed in this chapter avoids obstacles in a manner that is consistent with its OCP objective function because the obstacles themselves are part of the OCP formulation.

In [71], an MPC-based technique for short-term path planning among multiple moving objects is presented. Experiments are carried out in simulation and present promising results in a variety of scenarios. However, the work depends on a linearized bicycle model assumption, which may not be a valid assumption during avoidance maneuvers that require large road wheel angles. The authors also assume that all obstacle information is known before system start-up and that the operating environment is a straight road, which makes their approach not applicable to a real-world setting.

In [72] the authors take a Dynamic Programming (DP) approach to optimal obstacle avoidance. The paper claims that with a prediction horizon of 50 their method has "high accuracy and fast computing time, which can satisfy the requirements for application of autonomous vehicle driving on real roads." [72]. However, the authors do not present any quantitative data relating to computation time of the DP solution. Since DP solutions are usually prohibitively slower than other optimal control techniques (especially with a large prediction horizon), these claims should be viewed with skepticism until full experimental data is released.

In [73] the authors implement a path following MPC controller similar to the one presented here and deploy it to a VW Golf VII. However, no constraints for static obstacles, dynamic obstacles, nor passenger comfort were designed. This strictly limits the operational domain of the deployment to the simple scenario of following a reference line.

In [74] the authors use hard constraints to avoid moving obstacles, similar to the work presented here. However, the application of [74] is high speed ground vehicles in unstructured environments. The authors realize the benefit of a single-level design where motion planning and reference tracking are combined into a single optimization program. The authors also compare hard-constraint vs. soft-constraint (e.g. potential fields) formulations of the obstacle avoiding optimization problem. The authors find that hard constraints outperform soft constraints in terms of obstacle avoidance performance and optimization time. However, even with the hard constraint formulation the optimization time was 45 seconds, putting the method well out of the domain of real-time implementation. In contrast, our work presented in this chapter operates in real-time.

In [75] the authors operate in the same context of [74], without a reference in an unstructured environment. However, in [75] the obstacles are assumed to be static. Additionally, the authors only consider constraints that enforce vehicle dynamic safety, and not passenger comfort. Additionally, like [69], the controller decides on a speed profile, which then needs to be further regulated into torque commands, which can introduce additional predictive model inaccuracy.

5.1.2 Contributions

This chapter addresses the issues presented above by introducing a novel OCP formulation of the DDT and an accompanying MPC solution. The main areas of improvement in this regard are:

1. The system model is designed to align closely with physical platforms and to allow for computation of comfort metrics and constraints, enabling the OCP to be formulated to abide by the SAE guidelines for passenger comfort.

2. Constraints are added to the OCP formulation that guarantee feasibility of control actions with respect to road boundaries, static obstacles, and dynamic vehicles. In this way, trajectory planning can be executed simultaneously within the controller, and the final control actions are guaranteed to be dynamically feasible.

3. The controller operates in real-time by employing a novel parallel-solver method which uses a warm-started "online" solver for real-time performance, while employing a parallel "exploration" solver which serves to break out of warm-started local minima.

The remainder of this chapter is organized as follows: Section 5.2 presents necessary background information on routing, reference line parameterization, and obstacle representations. Section 5.3 presents the OCP formulation, including the system model, objective function, road boundary constraints, and obstacle constraints. Section 5.4 discusses the MPC solution to the OCP, including how the OCP is expressed as a Non-Linear Program (NLP), and the novel parallel-solver method for real-time performance. Experimental results are then presented and analyzed in Section 5.5. Conclusions are discussed in Section 5.6.

5.2 Background

In order to properly understand the motion planning scheme in which the proposed controller fits, some background information is necessary. We first note the information given to the motion planning scheme:

1. A Lanelet2 map which describes lane geometry and topology.
2. The desired goal position on the map.
3. A discretized occupancy grid representation of the static environment.
4. Non-ego dynamic vehicle tracks that describe the position and longitudinal velocity of the vehicle.

5.2.1 Reference Spline Creation

In order to transform this information into reference signals that the controller can follow, we first need to find a lane-level route that describes how the vehicle can proceed from its current state to the goal state. This step is called global planning and is done using Dijkstra's search algorithm over the routing graph constructed from the Lanelet2 map [6].

However, a sequence of lanes is not yet an admissible reference signal for the controller. In order to generate a continuous and differentiable reference signal, a spline is fit to the centers of the lanes that make up the global route. In brief, CasADi's Interior Point Optimizer (IPOPT) [76] is used to find the spline coefficients that minimize squared error to the lane centers, see [69] for details.

An important note is that a spline has a limited capacity to express the complex lane geometry which may appear over a long route, incurring high squared error. To solve this problem, a sliding window is applied over the entire lane route, allowing the spline to accurately capture the simpler geometry of the lanes in the local window.

The same method is applied to generate splines for the left and right boundaries of the driveable surface. The lines that describes the driveable surface are determined using the routing rules stored in the Lanelet2 map.

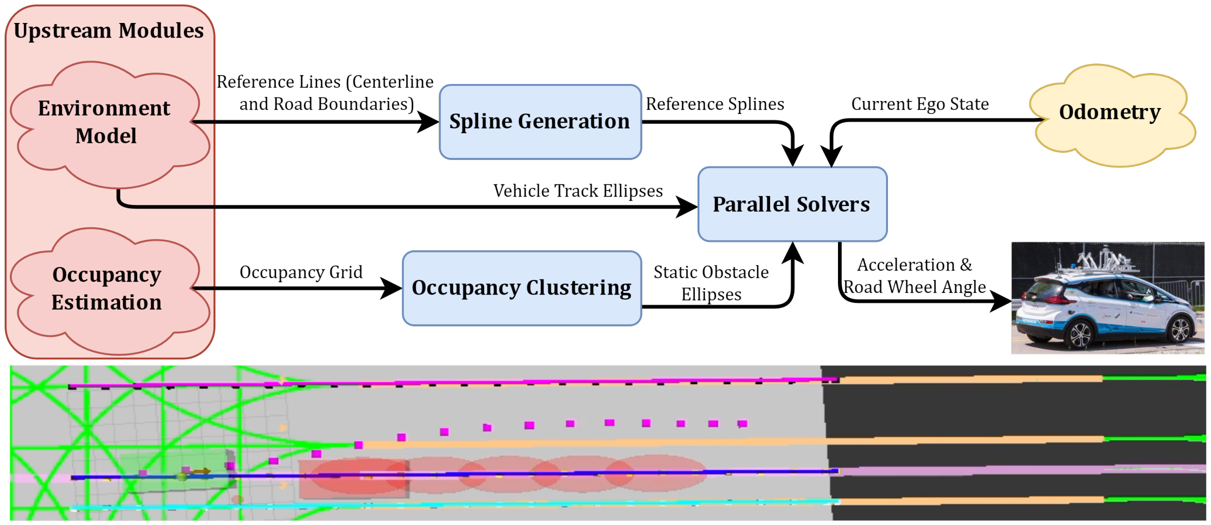


Figure 5.1: Graphical representation of the inputs giving to the motion planning scheme and the controller. The light pink line is the global route constructed from the lane centers. The blue line on top of it is the local window reference spline. The pink line to the left and the aqua blue line to the right are the drivable surface boundaries. The red ellipses show the predicted trajectory of the dynamic vehicle that the ego-vehicle is overtaking.

5.2.2 Occupied Space Representations

Aside from the centerline and road boundary splines, a reference for occupied space over time is also necessary to allow the controller to plan feasible trajectories. Two inputs are used to calculate this reference: An occupancy grid, and localized vehicle tracks from the tracker and environment model. The goal is to have a set of obstacles, and for each obstacle have a state trajectory that covers the MPC predictive horizon. The obstacle state representation used is a 2D ellipse, which has 5 degrees of freedom: X and Y of the centroid, RX (longitudinal radius), RY (lateral radius) and θ (yaw). To transform an occupancy grid into a set of obstacle trajectories, the grid is first filtered to only contain information of obstacles that are on the driveable surface. Then, a region growing algorithm is run with a neighborhood radius of four cells to cluster the obstacles in the grid (see Alg. 7). Lastly, the Minimum Volume Enclosing Ellipse (MVEE) approximate iterative algorithm [77] [78] is used to fit an ellipse to each cluster found by the region growing algorithm (see Alg. 8). The result of this process is illustrated in Fig. 5.2. Since the occupancy grid contains information about the static portions of the environment only, the occupancy ellipses are the same for each step in the MPC horizon.

Algorithm 7 Region Grow

Input: occupancy *grid*, row *r*, col *c*, radius *rad*, visited 2D array *visited*

Output: *[N, 2] array*, *N* is the number of cells in cluster

```
1: Initialize seedList ← [(r,c)]
2: Initialize cluster ← [(r,c)]
3: Initialize neighbors ← [n for c in product(range(-rad, rad + 1), repeat=2]) if n[0] != 0
   or c[1] != 0]
4: while len(seedList) > 0 do
5:     Initialize currCell ← seedList.pop(0)
6:     visited[currCell[0]][currCell[1]] = 1
7:     for neigh ∈ neighbors do
8:         Initialize tR ← currCell[0] + neigh[0]
9:         Initialize tC ← currCell[1] + neigh[1]
10:        if tmpR < 0 or tmpC < 0 or tmpR ≥ grid.height or tmpC ≥ grid.width then
11:            continue
12:        end if
13:        if grid[tR][tC] == 1 and visited[tR][tC] == 0 then
14:            seedList.append((tR, tC))
15:            cluster.append((tR, tC))
16:        end if
17:    end for
18: end while
19: return cluster
```

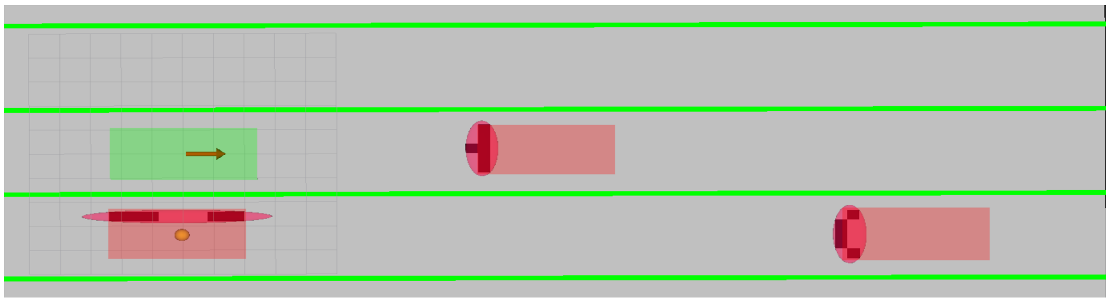


Figure 5.2: Illustration of the result of the occupancy clustering procedure described in Section 5.2.2. The discrete occupied cells are shown in black, the resulting clustered ellipses are shown in red, and the actual physical obstacles are shown in transparent red.

Algorithm 8 Grid To Ellipses

```
Input: occupancy grid, neighbor radius rad
Output: [N, 5] array, N is the number of ellipses in grid
 1: Initialize ells ← []
 2: Initialize visited ← zeros_like(grid)
 3: for r in range(grid.height) do
 4:     for c in range(grid.width) do
 5:         if visited[r][c] == 0 and grid[r][c] == 1 then
 6:             cluster = RegionGrow(grid, r, c, rad, visited)
 7:             C, rx, ry, theta = MVEE(cluster)
 8:             ells.append(C.x, C.y, rx, ry, theta)
 9:         end if
10:     end for
11: end for
12: return ells
```

Algorithm 9 Predicted Dynamic Vehicle Trajectory

```
Input: envModel, track, MPC step S (in seconds), MPC horizon M
Output: [M, 5] array, vehicle ellipse trajectory prediction
 1: path = envModel.localizeVehicle(track)
 2: startDist = toArcCoordinates(path, track.pos)
 3: traj = []
 4: for i in range(M) do
 5:     lonDist = startDist + i * S * track.lonVel
 6:     p1 = interpPointAtDistance(path, lonDist)
 7:     p2 = interpPointAtDistance(path, lonDist + 0.1)
 8:     theta = atan2(p2.y - p1.y, p2.x - p1.x)
 9:     traj.append({ p1.x, p1.y, track.rx, track.ry, theta })
10: end for
11: return traj
```

To generate the ellipse trajectories for the dynamic vehicle tracks, the environment model module (see Chapter 3) first localizes each track in the lanelet it geometrically occupies. Then, based on the estimated longitudinal velocity (assumed constant) from the tracker, and the geometry of the upcoming lanelets, the ellipses are generated following Alg. 9. This simple prediction scheme can be expanded to a more complex learning based technique, but this simple method suffices for controller design.

In summary, the information given to the controller is (as shown in Fig. 5.1):

1. A spline (denoted S_{ref}) that describes a local window of the global route.
2. Two splines (denoted S_{left} and S_{right}) which describe a local left and right drivable surface boundary.
3. A set of predicted trajectory ellipses (denoted O, where O_1 is the set of obstacles at step 1, and $O_{1,1}$ is the prediction for obstacle 1 at the 1st step in the MPC horizon).

5.3 Optimal Control Problem Formulation

This section describes the plant system to be controlled, the predictive model used in the proposed MPC controller, the objective function, and the constraints.

5.3.1 System and Predictive Models

Input Variables: The plant speed is controlled by applying a longitudinal acceleration command (denoted u_a), and yaw is controlled via commanded road wheel angle (denoted u_δ). Additionally, an input variable referred to as *path progress* u_ξ controls how far along the reference spline the next state will progress. The usage of this path progress input in the context of path following is explained below.

State Variables: The state vector was chosen in order to maintain the vehicle dynamic metrics that are necessary to compute the constraints presented below. Specifically, the state vector maintains the vehicle's inertial pose tuple $[x_X, x_Y, x_\psi]^T$ as well as longitudinal and forward body frame velocities (denoted x_{v_x}, and $x_{v_{fwd}}$, respectively). Additionally, a state variable referred to as the *path integral* x_Ξ is defined as the integration of the path progress input mentioned above. The path integral variable keeps track of how far along the reference route the ego is, and its functionality in terms of the objective function is explained below.

Output Variables: The system output variables are the odometry signals generated by the INS. The INS directly measures inertial pose, as well as longitudinal and lateral velocities and accelerations in the body frame. The current path integral is estimated using a discretized linear search over the reference spline's parameter range.

Predictive Model: To describe the evolution of the system over time, a nonlinear model is necessary due to the nonlinear behavior of vehicular systems under large road wheel angles [79]. In an urban driving setting, maneuvers that require large road wheel angles are common. This is in contrast with a highway driving setting that may only require a few degrees of road wheel angle and hence operate inside of a linear dynamics range. To this end, a nonlinear kinematic bicycle model is employed, as illustrated in Fig. 5.3 and described by Eqs. 5.1, 5.2:

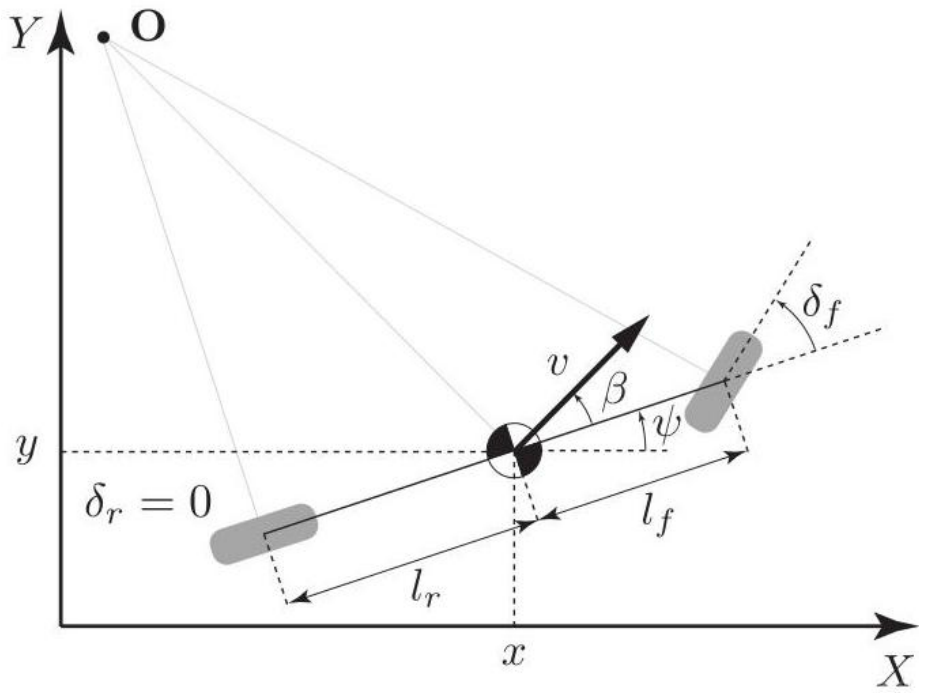


Figure 5.3: Schematic of the kinematic bicycle model representation of a vehicle. In the figure, δ_f is the commanded road wheel angle, l_f and l_r are the front and rear wheel bases, v is the direction of travel, and β is the slip angle of the vehicle at its CoG.

$$\dot{x}_X = x_{v_{fwd}} cos(x_\psi + \beta) \tag{5.1a}$$

$$\dot{x}_Y = x_{v_{fwd}} sin(x_\psi + \beta) \tag{5.1b}$$

$$\dot{x}_\psi = \frac{x_{v_{fwd}} cos(\beta) tan(u_\delta)}{l_f + l_r} \tag{5.1c}$$

$$\dot{x}_{v_x} = u_a \tag{5.1d}$$

$$\dot{x}_\Xi = u_\xi \tag{5.1e}$$

where

$$\beta = \arctan\left(\frac{l_r tan(u_\delta)}{l_f + l_r}\right) \tag{5.2}$$

is the slip angle of the vehicle at its CoG.

In [79] the usage of the kinematic bicycle model as a predictive model for MPC is examined in depth, and the authors come to the conclusion that "the kinematic bicycle model is a good modeling for low-speed vehicles but seems to not be precise enough for vehicles at high speed" [79]. Since the target platform has a speed restriction of 25MPH, the kinematic bicycle model is expected to capture the dynamics of the simulated high fidelity vehicle model well enough for the presented MPC implementation.

In summary, a nonlinear kinematic bicycle model was chosen as it is a simple predictive model that captures the nonlinear system dynamics for low-speed urban navigation [79], and thus is the best trade-off between NLP computation time and accurate prediction.

5.3.2 Objective Function

The terms included in the OCP objective function are:

1. Reference position error. Calculated as squared error between x_X, x_Y and S_{ref} at arclength x_Ξ.
2. The distance of the path parameter to route completion. Calculated as $(1 - x_\Xi)$.

Giving the final objective function:

$$J = \sum_{k=1}^{N} \left[\begin{bmatrix} x_X^k \\ x_Y^k \end{bmatrix} - \begin{bmatrix} x_{ref}^k \\ y_{ref}^k \end{bmatrix} \right]_2 + Q(1 - x_\Xi^k) \tag{5.3}$$

where Q is a positive number and $x_{ref}^k = S_{ref}^x(x_\Xi^k)$. Thus, the higher the Q value, the more incentivized the ego will be to complete the route, even at the cost of deviating from the reference route if needed. The outcome of the tuning process of Q was two separate weighting settings, one for nominal path following and another setting for when the controller is avoiding an obstacle. In the path following setting, a $Q = 1$ weight is used which ensures accurate following of the reference spline, whereas in the obstacle avoidance setting (when the state is within 5 meters of an obstacle), $Q = 1000$ is used to encourage the vehicle to deviate from the reference if necessary to avoid the obstacle.

Table 5.1: Optimal Control Problem Constraints

Variable(s)	Constraint	Type	Reasoning	Calculation
$\boldsymbol{x}_{k+1}$	$f(\boldsymbol{x}_k, \boldsymbol{u}_k)$	Non-Holonomic	Dynamics of system described via constraints as per multiple shooting [80]	RK4. See Section 5.3.1 for system used to model f.
u_δ	$-\frac{\pi}{4} \leq u_\delta \leq \frac{\pi}{4}$	Non-Holonomic	Physical range of wheel shaft	N/A, obtained from fact sheet
x_{v_x}	$0 \leq x_{v_x} \leq v_{MAX}$	Legal	Vehicle cannot exceed speed limit	N/A, supplied by Lanelet2 map
$\dot{x}_{v_y}$	$-3.5 \leq \dot{x}_{v_y} \leq 3.5$	Comfort	SAE bounds on lateral acceleration	From [81]: $\dot{x}_{v_y} = \frac{x_{v_x}^2 u_\delta}{l_f + l_r}$
u_a	$-3.5 \leq u_a \leq 3.5$	Comfort	SAE bounds on longitudinal acceleration	N/A, input variable
$\dot{u}_a$	$-10 \leq \dot{u}_a \leq 15$	Comfort	SAE bounds on longitudinal jerk	$\dot{u}_a = \frac{u_a^k - u_a^{k-1}}{T}$ Where T is the sample time
$[x_X, x_Y]'$	$[x_X, x_Y]' \in S^{drive}$	Feasibility	The vehicle must be on the driveable surface	See Sec 5.3.3
$[x_X, x_Y]'$	$[x_X, x_Y]' \notin \chi^{occ}$	Feasibility	The vehicle can only occupy free space	See Sec 5.3.3
x_Ξ	$0 \leq x_\Xi \leq 1$	Feasibility	Vehicle required to stop at end of route	N/A, state variable

These two simple terms embody the non-safety constraint portions of the DDT: Drive close to the desired route, and make efficient progress towards route completion. The rest of the DDT (the safety constraints) is implemented as hard constraints on the NLP and are covered below.

5.3.3 Constraints

All vehicular systems are non-holonomic and therefore an OCP aiming to control such a system must take the non-holonomic constraints into consideration. These non-holonomic constraints include the system model presented above, as well as further physical constraints like maximum steering angle. There are also constraints enforced for the comfort of the passenger as per the SAE guidelines, which include limits on longitudinal and lateral acceleration as well as jerk. Most importantly, there are the constraints that enforce the feasibility of the predicted vehicle trajectory: The vehicle must stay in free space, defined as inside the road boundaries and not in collision with any obstacle. A complete list of the OCP constraints can be found in Table 5.1.

Road Boundary Enforcement

There are multiple ways to enforce that a 2D point $[x_X, x_Y]'$, i.e. vehicle position, must be inside a set of two splines. The method we found worked best was a “sidedness” test, enforcing that $[x_X, x_Y]'$ is to the right of S_{left}, and to the left of S_{right}. The first step in the formulation is to obtain a 2D bound vector $[b_0, b_1]'$ that we enforce the sidedness constraint with respect to. b_0 is calculated as the boundary spline at arclength x_Ξ and b_1 is calculated as the boundary spline at arclength $x_\Xi + la$ where la is a small look-ahead (e.g. 0.01). Then, the (left) sidedness constraint can be enforced as in Eq. 5.4 (right constraint is < 0).

$$(x_X - b_0.x) * (b_1.y - b_0.y) - (x_Y - b_0.y) * (b_1.x - b_0.x) > 0 \tag{5.4}$$

Obstacle Avoidance Enforcement

For each obstacle, and for each step in the MPC horizon, the vehicle’s footprint cannot intersect with any obstacle ellipse. Eq. 5.5 *InEllipse(p, el)* implements the basic operation needed for such a constraint, testing if a 2D point p is inside an ellipse el (assuming that the point and the ellipse are in the same reference frame, a detail that is worked out in Alg. 10).

$$\frac{(p.x - el.x)^2}{el.rx} + \frac{(p.y - el.y)^2}{el.ry} < 1 \tag{5.5}$$

Now all that is left is to call *InEllipse(p, el)* for each ellipse we want to avoid at each step in the MPC horizon. Alg. 10 implements such a routine, where the $expandFootprint$ subroutine simply returns 6 points around the border of the vehicle’s footprint, and $ENFORCE$ enforces the enclosed constraint in the IPOPT solver. Note that in Alg. 10 we are creating $6 * |S| * |O_i|$ constraints ($|S|$ is the MPC horizon, $|O_i|$ is the number of obstacles). This can be on the order of 100s of constraints for a nominal $|S| = 15$ and $|O_i| < 10$.

5.3.4 Conversion to Torque Commands

In the CARLA simulator longitudinal commands can be applied directly, but in production they must be converted to torque commands to send to the CAN Bus. Alg. 11 presents how that conversion is done, where $Cd = 0.3$, $Cr = 0.02$, $A = 2.0$, $\rho = 1.2$, $m = 2500$, $r = 0.4$, $rat = 7.05$, $g = 9.81$.

Algorithm 10 Obstacle Avoidance

```
Input: Symbolic set of ellipse trajectories $O$, symbolic state trajectory $S$
Output: For each matching $O_i$, $S_i$ in the horizon, enforce that $S_i$ does not conflict with
    any object in $O_i$
 1: for $O_i \in O$, $S_i \in S$ do
 2:     for $O_{i,j} \in O_i$ do
 3:         th = $O_{i,j}.theta$
 4:         rot = [cos(th), sin(th); -sin(th), cos(th)]
 5:         el = {rot * $O_{i,j}.cent$, $O_{i,j}.rx$, $O_{i,j}.ry$}
 6:         for $S_{i,j} \in expandFootprint(S_i)$ do
 7:             ENFORCE(! InEllipse(rot * $S_{i,j}.pos$, el))
 8:         end for
 9:     end for
10: end for
```

Algorithm 11 Longitudinal Acceleration to Torque

```
Input: Desired longitudinal accel, current vehicle speed, drag coefficient Cd, rolling resis-
    tance Cr, frontal area A, air density $\rho$, vehicle mass m, wheel radius r, final gear ratio
    rat, gravity force g
Output: Torque value to command
 1: $AeroCons = 0.5 * \rho * A * Cd$
 2: $RollCons = m * g * Cr$
 3: $ResForce = speed * (AeroCons + RollCons)$
 4: $AccForce = accel * m$
 5: return $\frac{r}{rat} * (AccForce + ResForce)$
```

5.4 Model Predictive Control Solution

To solve the OCP, Nonlinear Model Predictive Control (NMPC) was used. This decision was due to: (1) The high number of constraints used to enforce non-holonomic system dynamics, comfort, and feasibility, and (2) The nonlinear dynamics of the system in an urban setting where large front wheel angles may be necessary to navigate.

In order to transform the OCP into a NLP, temporal discretization was applied to the system dynamics over a finite prediction horizon ($N = 15$). Specifically, the Fourth Order Runge-Kutta (RK4) method was used with a time step of $T = 0.25s$. As proposed by Bock and Plitt in [80], multiple shooting was used to enforce the dynamics of the system using

Algorithm 12 Variable Obstacle Solver Delegation

```
Input: Max number of obstacles $max\_obs$
Output: Delegation to solvers that handles variable numbers of obstacles
1: $controller\_map = \{\}$ {At system startup}
2: for $i \in range(max\_obs)$ do
3:     $solver\_map[i] = Solver(i)$
4: end for
5: while true {At runtime} do
6:     $obs\_set = PERCEIVE\_SCENE()$
7:     $sol = solver\_map[len(obs\_set)](obs\_set)$
8:     $EXECUTE\_SOL(sol)$
9: end while
```

constraints, reducing the nonlinearity of the objective function especially in the latter steps of the prediction horizon.

To solve the NLP, the interior point optimizer (IPOPT) [76] from the CasADi [82] software package was used. Computational concerns regarding real-time IPOPT solutions are discussed below. With the NLP solution obtained, receding horizon control is applied.

5.4.1 Dealing With a Variable Number of Objects

One nuance of obstacle avoidance via optimal control is the fact that the NLP must be specified at startup time. Coupled with the fact that a different number of objects in the scene leads to a different number of constraints and thus a different NLP formulation, it would seem required to know the number of obstacles in the scene *a priori*. To get around this impractical assumption, a collection of NLP solvers are initialized at system startup, each with a different number of constraints according to the number of obstacles the solver expects. Then, at system runtime, a solver is delegated to based on the number of obstacles actually observed in the scene, which may change over time. This procedure is outlined in Alg. 12.

5.4.2 Warm-Starting, Escaping Local Minima via Parallel Solvers

Real-time performance of the controller is defined as the NLP solver operating faster than the sampling time of the controller (0.25s), i.e. greater than 4Hz. In order to achieve real-time performance, warm-starting the iterative IPOPT solver with a "decent" solution is essential, allowing the solver to reach an acceptable solution after only a small number of iterations. The warm-start used at time t is normally the solution obtained by the solver at time $t-1$, following from the assumption that the parameters of the optimization change little from one solve to the next. However, over time this assumption may no longer be true, and the solver may become stuck in a series of warm-started local minima.

An example is the controller attempting to avoid an obstacle. To avoid an obstacle a large change from the previous solution is needed; diverting from the reference line and then re-joining it is a very different solution than just proceeding along the reference. Naive warm-starting does not allow for such a divergence from the previous solution. Given a warm-start that follows the reference line, and a small number of solver iterations, IPOPT will never find a solution that diverges around an obstacle (see Section 5.5.4 and Fig. 5.8 for an illustration of this failure case).

A method of "breaking out" of this local minima is required, a way to search for a more global minima given more IPOPT iterations and a less biased warm-start. Our novel method is to run a parallel solver that consumes the same NLP as the original solver (which we will refer to from now on as the "online" solver), but does not use warm-starting (initial decision variable assignments are all zero) and is allowed 10x the number of IPOPT iterations. We will refer to this new solver as the "exploration" solver.

These two solvers run asynchronously of each other, the online solver is used to control the vehicle as normal, and the exploration solver is used to simply "suggest" new warm-starts that the online solver could use. Whenever the online solver is about to perform a new solve, it polls the exploration solver for its most recent solve. If the polled solution has a lower cost than the cost of the previous online solution, the exploration solution is used for the online solver's warm-start instead of the previous online solution. This process is outlined in Alg. 13.

The result, viewed from a unified perspective, is a controller than can operate in real-time using a warm-starting scheme but also does not get stuck in local minima (see Section 5.5.4 for results). Of course this comes at the cost doubling the computation load of the controller, we are now running two IPOPT solvers instead of one. However each of those processes only run on a single core, so in total we only take two out of the 64 cores on the WATonomous computation platform for control. Note that this parallelized approach can be scaled up to an arbitrary number of exploring solvers, each running a potentially different warm-starting scheme and number of IPOPT iterations.

Algorithm 13 Warm Start Breakout

```
Input: Number of online IPOPT iterations $o\_iters$, number of exploration iterations $e\_iters$
Output: A NLP solution scheme that runs in real-time while not getting stuck in local minima
 1: $o\_solver = Solver(o\_iters)$ {At system startup}
 2: $e\_solver = Solver(e\_iters)$ {At system startup}
 3: while true {At runtime} do
 4:   $solver\_params = PERCEIVE\_SCENE()$
 5:   $warm\_start = o\_solver.sol$
 6:   if $e\_solver.has\_sol$ then
 7:     $e\_sol = e\_solver.sol$
 8:     if $e\_sol.cost < warm\_start.cost$ then
 9:       $warm\_start = e\_solver.sol$
10:     end if
11:     $e\_solver(solver\_params, \vec{0})$ {Exploration solver runs asynchronously}
12:   end if
13:   $sol = o\_solver(solver\_params, warm\_start)$
14:   $EXECUTE\_SOL(sol)$
15: end while
```

5.5 Experiments

5.5.1 Reference Line Following

In this experiment, the only reference we have to follow is the lane center as the ego vehicle completes a right turn at a 4-way intersection. The desired behavior is to have low lateral error from S_{ref}, keep the NLP solution time within the 0.25s sampling time, stay as close to the 8m/s speed limit as possible, and obey the legal and comfort constraints from Table 5.1. The results below show two controller settings: The nominal controller described above, and a no-comfort-constraint controller which has the comfort constraints from Table 5.1 disabled. The purpose of the two settings is to prove the effectiveness of the OCP design in regulating passenger comfort.

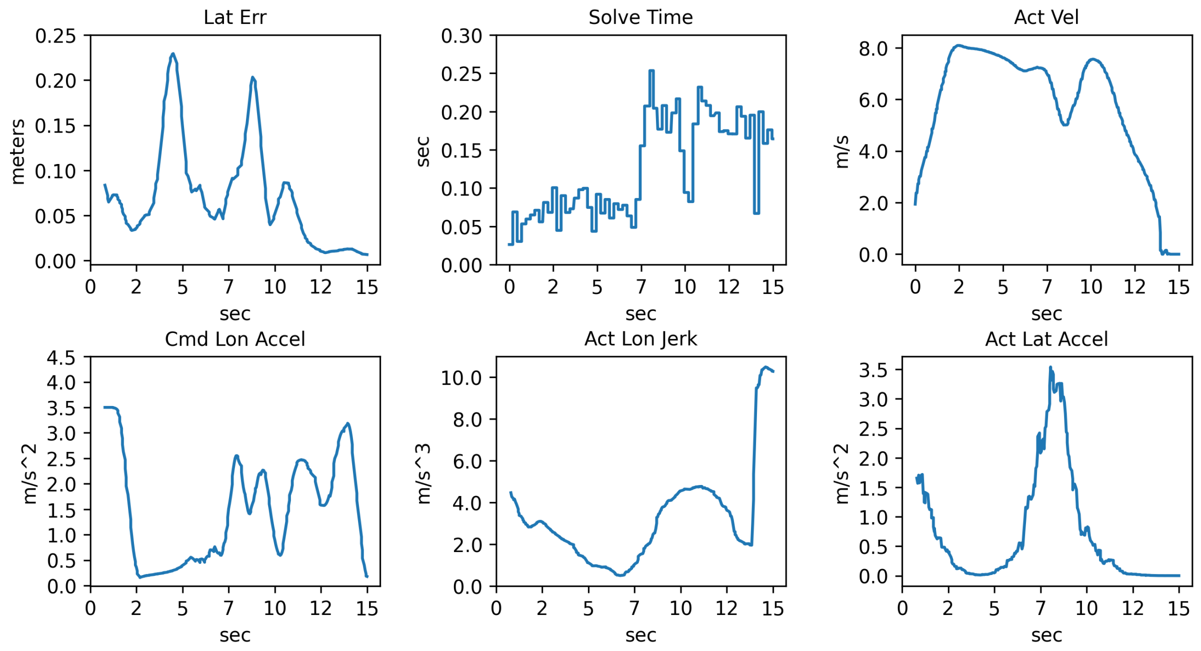


Figure 5.4: Quantitative reference line tracking performance of the nominal controller.

Qualitative Results

The qualitative results are presented via a video of the completion of the test scenario using the nominal controller, which can be found here: `https://youtu.be/2Gk3XQKlk38`.

Quantitative Results

Fig. 5.4 shows the quantitative results for the nominal controller. As seen in the figure, the lateral deviation stays below 25cm from the reference, well within the lane boundaries. The solution time of the NLP also stays below the sampling time of the controller, and the velocity stays close to 8m/s, except when necessary to slow down while taking the right turn to avoid passenger discomfort around t=548. The comfort metrics are all within the desired comfort range outlined by SAE.

Fig. 5.5 shows the quantitative results for the no-comfort-constraint controller. As seen in the figure, the lateral deviation increases (due to more jerky steering), but is still within the lane boundaries. The solution time and the velocity profile are both comparable to the nominal controller. However, in the velocity profile there is no drop in velocity around

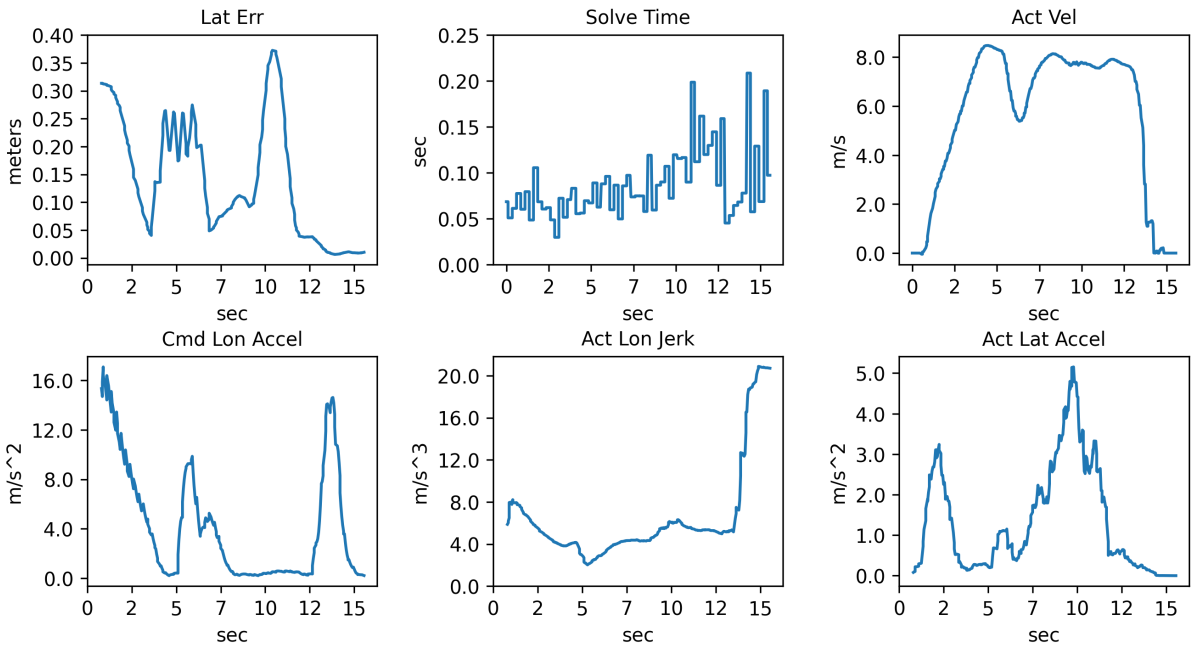


Figure 5.5: Quantitative reference line tracking performance of the no-comfort-constraint controller.

t=23805, in contrast with what we saw in the nominal controller. Taking the corner with such a high speed violates the lateral acceleration constraint, deeming the drive uncomfortable by SAE standards. Furthermore, when first accelerating and decelerating, both the longitudinal acceleration and jerk constraints are violated, which is further evidence of an uncomfortable ride for the passenger.

Overall, it is clear that introducing the comfort constraints to the OCP design forces the controller to slow down for sharp turns to keep lateral acceleration within the desired bounds, and to accelerate and decelerate more smoothly to ensure passenger comfort.

5.5.2 Static Obstacle Avoidance

In this experiment, a new reference is introduced: static obstacles. The desired behavior is the same as in Section 5.5.1, with the addition of not hitting any static obstacles. The scenario used to test this behavior is similar to that of Section 5.5.1, a straight road followed by a right turn. However an obstacle course of three static obstacles was added

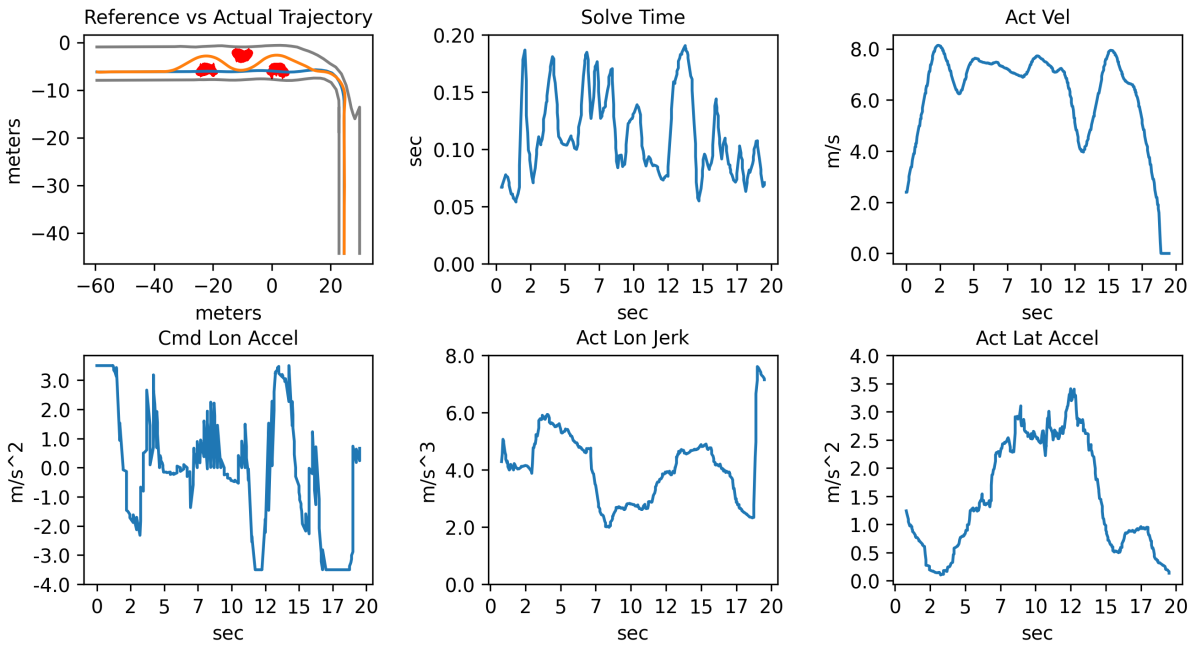


Figure 5.6: Quantitative static obstacle avoidance performance of the nominal controller.

to the straight road. The results below show the controller in its nominal setting avoiding the course of static obstacles using the formulation discussed in Section 5.3.3.

Qualitative Results

The qualitative results are presented via a video of the completion of the test scenario using the nominal controller, which can be found here: `https://youtu.be/T83wHpZDfd0`. As desired, there are no collisions with the static obstacles on the course.

Quantitative Results

Fig. 5.6 shows the quantitative results for the nominal controller. As shown in the top left subplot, the reference (in blue) is tracked well when the controller is not performing an obstacle avoidance maneuver. The controller also successfully avoids the static obstacles (in red). Furthermore, the controller performs the avoidance maneuvers while staying inside the driveable surface bounds (shown in grey). Lastly, all the desired driving behaviors from Section 5.5.1 (namely the SAE comfort metrics) are also still obeyed.

5.5.3 Road Boundary

To show the effectiveness of the road boundary constraints discussed in Section 5.3.3, they were removed in this experiment and the resulting vehicle behavior is discussed below. The same static obstacle course testing scenario was used to examine the results. The desired behavior is the same as in Section 5.5.2.

Qualitative Results

The qualitative results are presented via a video of the attempted completion of the static obstacle course over two attempts by the no-road-boundary controller, which can be found here: `https://youtu.be/YqpCCIzRw28`. As seen in the video, without the constraints the controller may choose to avoid the obstacles in an illegal manner, leaving the road surface completely. This video demonstrate the necessity and effectiveness of the road boundary constraints discussed in Section 5.3.3.

Quantitative Results

A quantitative analysis of the no-road-boundary controller is shown in Fig. 5.7. Without the road boundary constraints the controller may arbitrarily decide to avoid the obstacle on either side, possibly violating the road boundary traffic rule (see top left plots, where the orange ego trajectories violate the grey road boundaries). These plots, when compared against the plot in Fig. 5.6, shows that the road boundary constraints specified in Section 5.3.3 are necessary for correct driving behavior.

5.5.4 Parallel Solver

To show the effectiveness of the parallel solver technique discussed in Section 5.4.2, it was removed (only the online solver is used) in this experiment. The same static obstacle course testing scenario was used to examine the results. The desired behavior is the same as in Section 5.5.2. The resulting vehicle behavior is discussed below.

Qualitative Results

Qualitative results of the no-parallel-solver controller failing to complete the scenario are shown at: `https://youtu.be/cLFCUO3-1MY`. As seen in the video, without the parallel

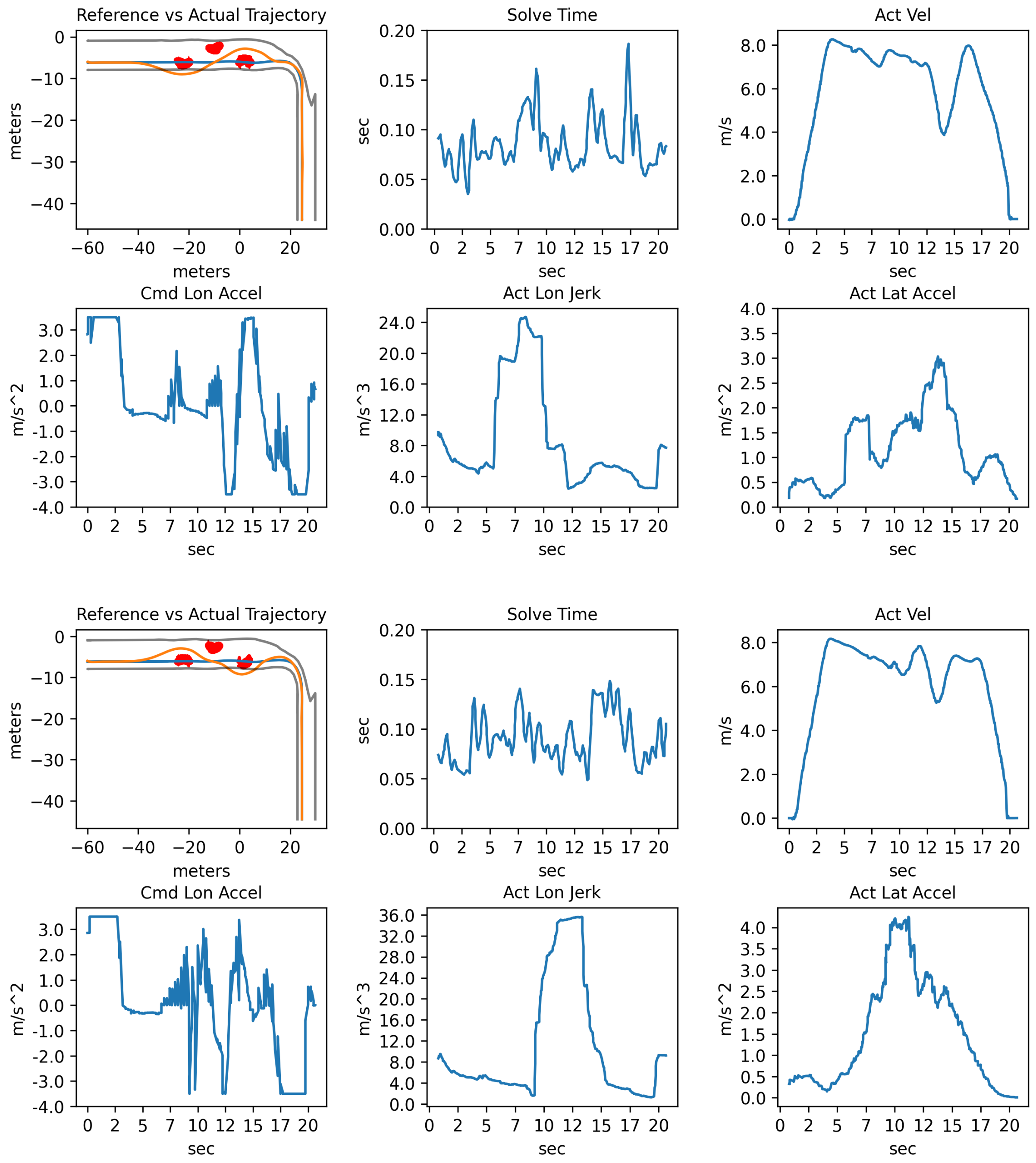


Figure 5.7: Quantitative static obstacle avoidance performance of the no-road-boundary controller over two runs.

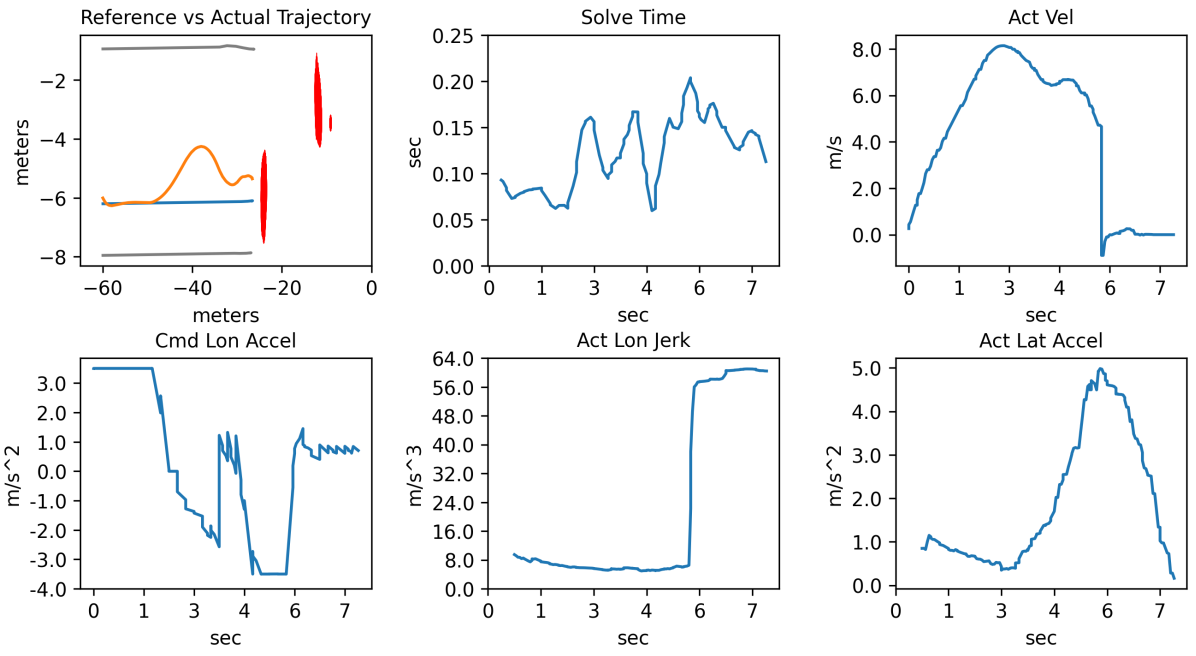


Figure 5.8: Quantitative static obstacle avoidance performance of the no-parallel-solver controller.

solver the controller easily gets stuck in a local warm-starting minima and fails to avoid even the first obstacle. This video demonstrate the necessity and effectiveness of the parallel solver technique discussed in Section 5.4.2 to avoid such local minima traps when using warm-starting for real-time performance.

Quantitative Results

A quantitative analysis of the no-parallel-solver controller is shown in Fig. 5.8. As seen in the reference plot (top left) of the figure, without the parallel solver, the controller cannot find a trajectory around even the first static obstacle on the course. The failure observed here compared to the success in Section 5.5.2 is due to the missing exploration solver discussed in Section 5.4.2. In the no-parallel-solver controller setting the controller lacks the ability to escape local-minima traps caused by the warm-starting method necessary for a real-time implementation.

5.5.5 Dynamic Obstacle Avoidance

In this experiment, a new reference is introduced: dynamic obstacles. The scenario used is a straight road with the ego obeying a speed limit of 15m/s. In front of the ego, there is a target vehicle traveling at 7.5m/s. In order to make efficient progress along the route, the ego vehicle should overtake the target vehicle in a comfortable and legal manner.

Qualitative Results

The qualitative results are presented via a video of the completion of the test scenario using the nominal controller, which can be found here: https://youtu.be/oqjdudBFZ9E. As desired, the ego vehicle overtakes the target vehicle in an efficient and legal manner.

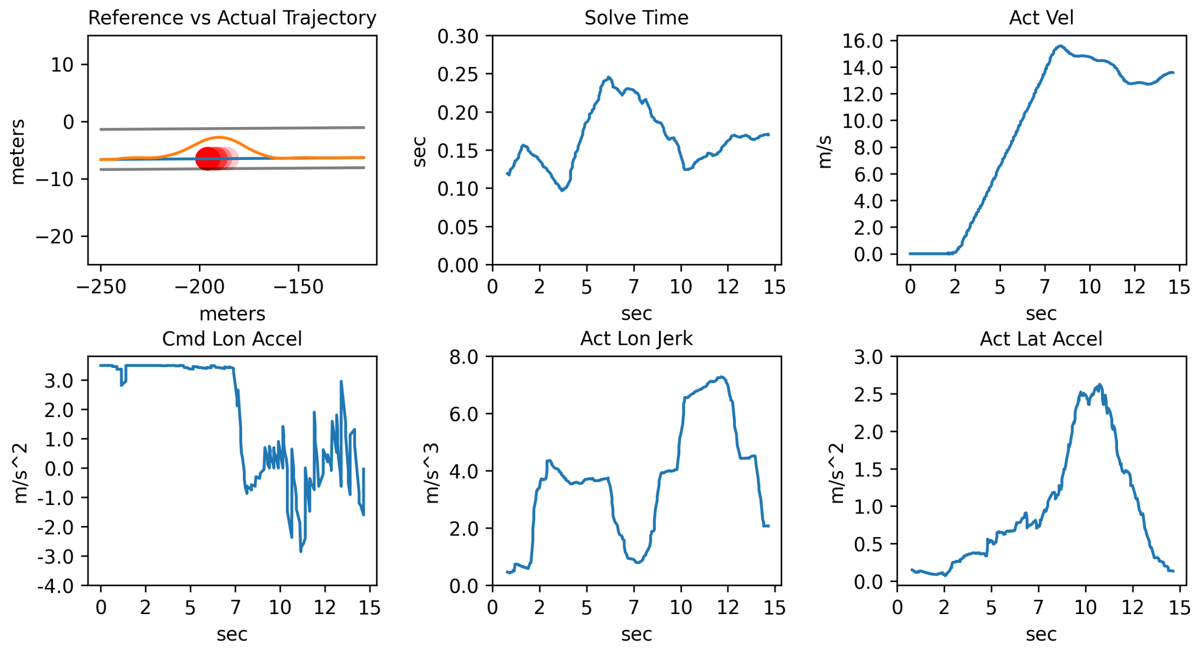


Figure 5.9: Quantitative dynamic obstacle avoidance performance of the nominal controller.

Quantitative Results

Fig. 5.9 shows the quantitative dynamic obstacle avoidance results for the nominal controller. As shown in the top left subplot, the reference (in blue) is tracked well when the controller is not overtaking the target vehicle. The controller also successfully avoids the target vehicle (in semi-transparent red), while staying inside the driveable surface bounds (shown in grey). Furthermore, the controller performs the maneuver efficiently (near 15m/s) and comfortably (SAE guidelines obeyed).

5.6 Conclusion

This chapter presented a novel OCP formulation and MPC solution to the DDT that allows for unified trajectory planning and control in a feasibility guaranteed manner. The presented scheme is shown to have key advantages over previous DDT motion planning and execution schemes, including direct adherence to SAE passenger comfort guidelines, as well as free space and road boundary conscious control via the OCP constraints. Promising experimental results were achieved for obstacle avoidance, overtaking, and turning maneuvers.

As for future research directions, incorporating learning techniques for more accurate predictions of non-ego vehicle trajectories will enhance the motion planning scheme[1]. Furthermore, we plan to integrate the proposed motion planning and control framework with learning-based behavioral planners, e.g. [83], for developing feasible decision-making schemes in complex urban environments.

[1]Note that the OCP design will not have to change, only the predictive accuracy of the inputs does.

Chapter 6

Conclusion

In this chapter we conclude by summarizing the work and contributions presented in this thesis, and discussing future research directions. The main focus of this thesis was a study of the software architecture and data pipeline to make autonomous driving work in practice. There was a special focus on the environment modeling and behavioral planning, action classification, and trajectory planning and control modules of the ADS. The main contributions proposed are:

1. The DRG, a standard tool for extending prior maps with online observations to create a unified environment model, allowing for greater transparency and collaboration on decision-making algorithms.

2. RAD-Net, which showed that action classifiers originally built for human action recognition are also effective in the road scene domain, and in fact benefit from pretraining on large human action datasets due to shared neural representations of actions between the two domains.

3. A unified trajectory planning and control OCP framework, in which constraints were added that guarantee the feasibility of control actions with respect to road boundaries, static obstacles, and dynamic vehicles. In this way, trajectory planning can be executed simultaneously within the controller, and the final control actions are guaranteed to be dynamically feasible.

6.1 Software Architecture and Data Pipeline

In Chapter 2, the WATonomous software architecture and ADS data pipeline were presented. The software architecture allowed for rapid prototyping of new software modules, and rapid onboarding of new team members due to its Dockerized and cloud-based design. Furthermore, there was no software constraints on how many ADS instances could run in parallel on a WATonomous server cluster VM due to the Dockerized simulation setup and network isolation design. Lastly, the entire ADS developed in simulation is able to be deployed to the physical research vehicle without modifications to the ADS modules due to the replication of the physical platform in the Carla ROS Bridge sensor configuration.

The data pipeline that enables autonomous driving in the ADS was also presented, from raw sensor input to the CAN Bus interface, as well as human-computer interfacing. The sensor drivers that broadcast the sensor data into the ROS network were outlined. The computer vision and tracking algorithms used to transform the high dimensional sensor data into low dimensional descriptions that the environment model can understand were discussed. The Lanelet2 mapping and environment modeling techniques were briefly mentioned, but illustrated in depth in Chapter 3. Similarly, trajectory planning and control were briefly summarized but were analyzed more deeply in Chapter 5.

6.2 Mapping, Environment Modeling, and Decision Making

In Chapter 3, the environment modeling module and relevant sub-modules (Lanelet2 mapping and behavioral planning) were discussed. The discussion was focused on the methodology behind and implementation of the DRG, a novel relational graphing tool for augmenting a prior map at system runtime to serve as a unified environment model. The benefits of the DRG with respect to behavioral planning complexity analysis, verification, and information sharing were studied. Evaluations were conducted to empirically analyze the DRG's affects on behavioral planning complexity, and to show the real-world effectiveness of the DRG for enabling decision making in urban environments. In-depth implementation details were given for common DRG augmentation routines, the same routines that were used when the DRG was deployed to the research vehicle during the Year 4 SAE AutoDrive Challenge I (demo: `https://youtu.be/DNZgheT4Y2s?t=153`).

6.3 Action Classification

In Chapter 4, the computer vision task of action classification in video streams was examined using the ROAD dataset. Furthermore, the resulting classification pipeline was deployed to the ADS, yielding appearance based relationship modeling in the DRG. RAD-NetV1 was introduced first, making strides in two-stage action classification and showing the effectiveness of transfer learning from large scale human action datasets in the context of road scene action classification. Next, RAD-NetV2 was presented as a way to improve on the shortcomings of RAD-NetV1, which included focuses on active-agent detection, the explicit handling of high-motion scenes, and dealing with class imbalance. Whereas RAD-NetV1 fell short of the 3D-RetinaNet baseline by 1.0 point on the "val-1" split, RAD-NetV2 beat the baseline by 4.1 points on the same split, showing the effectiveness of the improvements identified in Section 4.2. The deployment of RAD-NetV2 to the ADS was also successful, enabling appearance based and therefore less conservative environment modeling and decision making in the context of pedestrians (see Section 4.3.2 for details).

6.4 Trajectory Planning and Control

In Chapter 5, the tasks of trajectory planning and control were tackled via optimal control techniques. Historically, these two modules have been separated into two modules in automated driving stacks. After arguing that such a separation is unnecessary and in-fact flawed due to the mismatch between planned trajectories and what the controller can feasibly execute, we proposed a unified approach to these two tasks in the form of an OCP. Our OCP is formulated to allow for computation and enforcement of constraints for passenger comfort, road boundary rules, as well as avoidance of static and dynamic obstacles. We showed how such an OCP could be solved in real-time under the MPC paradigm by employing a novel parallel-solver technique that reaps the benefit of warm-starting the IPOPT solver while not getting stuck in local minima. A rigorous suite of experiments and ablation studies were conducted to prove the effectiveness of the proposed OCP constraint design and the parallel-solver methodology.

6.5 Future Research Directions

6.5.1 Software Architecture and Data Pipeline

Currently, WATonomous is working on the next generation of the *watod* ecosystem, featuring improved simulation environments based on NVIDIA's Isaac Sim [84] and NerF [85] for photorealism and Chrono [86] for vehicle dynamics realism. These environments will serve as a more accurate representation of the real world, allowing for computer vision models to be trained in simulation, and less controller tuning when we go from simulation development to real world deployment.

6.5.2 Mapping, Environment Modeling, and Decision Making

Currently, WATonomous is working on the next generation of world modeling in which a full-fledged physics simulator is implanted in the DRG for better modeling of a highly dynamic environment.

Another interesting direction of future research is a shared environment model between multiple autonomous agents operating in the same region, as done for localization purposes in [87]. Since the DRG is map-centric instead of ego-centric, data sharing between individual DRG instances is a natural extension to the current design.

Future work in non-ego trajectory prediction based on a neural fusion of prior maps and online observations will increase the modeling accuracy of the DRG and thus improve the safety of the control scheme. The review in [88] provides a good starting point for such work.

6.5.3 Action Classification

The performance of the action classification model can also be iterated on, perhaps using one of the convolutional-free video architectures, such as TimeSformer [60], which are built on purely self-attention computations over the space and time dimensions. Such designs, although radically different than the 3D-CNNs discussed in this thesis, have shown state of the art performance on the Kinetics Dataset, and can be applied to longer views.

Towards the future, we welcome more datasets that set a standard for road scene action recognition by including more varied and better defined agent actions. We believe that it may also be possible to achieve additive performance benefits from existing approaches by combining our contributions with existing post-processing ideas that work well, such as the cube proposal approach from the Argus++ authors [89].

6.5.4 Trajectory Planning and Control

In terms of trajectory planning and control, extensions of the parallel-solver framework are possible. As mentioned in Section 5.4.2, any number of parallel solvers can be introduced, each with a different warm-starting design. Several more complex candidates, compared to the current zero-vector approach, are possible. For example, using a classical planning algorithm, or a learned approach to trajectory planning, to generate warm-starts is possible.

In the literature, recent strides have also been made in achieving accurate predictive models via data driven approaches based on optimizing Gaussian Process (GP)s [90] [91]. Similar methods could be applied for a data drive approach to modeling the dynamics of Bolty, expanding its operational domain to driving surfaces where kinematic based models fall short (e.g. slippery surfaces).

References

[1] Gurkirt Singh, Stephen Akrigg, Manuele Di Maio, Valentina Fontana, Reza Javanmard Alitappeh, Suman Saha, Kossar Jeddisaravi, Farzad Yousefi, Jacob Culley, Tom Nicholson, et al. Road: The road event awareness dataset for autonomous driving. *arXiv preprint arXiv:2102.11585*, 2021.

[2] Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *European conference on computer vision*, pages 402–419. Springer, 2020.

[3] Junting Pan, Siyu Chen, Mike Zheng Shou, Yu Liu, Jing Shao, and Hongsheng Li. Actor-context-actor relation network for spatio-temporal action localization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 464–474, 2021.

[4] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, NanoCode012, Yonghye Kwon, TaoXie, Jiacong Fang, imyhxy, Kalen Michael, Lorna, Abhiram V, Diego Montes, Jebastin Nadar, Laughing, tkianai, yxNONG, Piotr Skalski, Zhiqiang Wang, Adam Hogan, Cristi Fati, Lorenzo Mammana, AlexWang1900, Deep Patel, Ding Yiwei, Felix You, Jan Hajek, Laurentiu Diaconu, and Mai Thanh Minh. ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference, February 2022.

[5] Self-driving uber car that hit and killed woman did not recognize that pedestrians jaywalk. http://www.nbcnews.com/tech/tech-news/self-driving-uber-car-hit-killed-woman-did-not-recognize-n1079281. Accessed: 2021-02-17.

[6] Rowan Dempster, Mohammad Al-Sharman, Yeshu Jain, Jeffery Li, Derek Rayside, and William Melek. Drg: A dynamic relation graph for unified prior-online environment modeling in urban autonomous driving. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 8054–8060. IEEE, 2022.

[7] Jessica Van Brummelen, Marie O'Brien, Dominique Gruyer, and Homayoun Najjaran. Autonomous vehicle perception: The technology of today and tomorrow. *Transportation research part C: emerging technologies*, 89:384–406, 2018.

[8] Alex Zhuang, Eddy Zhou, Quanquan Li, Rowan Dempster, Alikasim Budhwani, Mohammad Al-Sharman, Derek Rayside, and William Melek. Radacs: Towards higher-order reasoning using action recognition in autonomous vehicles. *arXiv preprint arXiv:2209.14408*, 2022.

[9] Rowan Dempster, Mohammad Al-Sharman, Derek Rayside, and William Melek. Real-time unified trajectory planning and optimal control for urban autonomous driving under static and dynamic obstacle constraints, 2022.

[10] Autoware. lidar_euclidean_cluster_detect.

[11] Wise automated driving system, Sep 2022.

[12] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

[13] Xinshuo Weng, Jianren Wang, David Held, and Kris Kitani. 3d multi-object tracking: A baseline and new evaluation metrics. *arXiv preprint arXiv:1907.03961*, 2020.

[14] Fabian Poggenhans, Jan-Hendrik Pauls, Johannes Janosovits, Stefan Orf, Maximilian Naumann, Florian Kuhnt, and Matthias Mayr. Lanelet2: A high-definition map framework for the future of automated driving. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 1672–1679. IEEE, 2018.

[15] OpenStreetMap Wiki. Josm — openstreetmap wiki,, 2022. [Online; accessed 29-September-2022].

[16] Yunfan Kang and Amr Magdy. Hidam: A unified data model for high-definition (hd) map data. In *2020 IEEE 36th International Conference on Data Engineering Workshops (ICDEW)*, pages 26–32. IEEE, 2020.

[17] Marius Dupuis, Martin Strobl, and Hans Grezlikowski. Opendrive 2010 and beyond–status and future of the de facto standard for the description of road networks. In *Proc. of the Driving Simulation Conference Europe*, pages 231–242, 2010.

[18] Mohammad Al-Sharman, David Murdoch, Dongpu Cao, Chen Lv, Yahya Zweiri, Derek Rayside, and William Melek. A sensorless state estimation for a safety-oriented

cyber-physical system in urban driving: deep learning approach. *IEEE/CAA Journal of Automatica Sinica*, 8(1):169–178, 2020.

[19] Mohamed A. Daoud, Mohamed W. Mehrez, Derek Rayside, and William W. Melek. Simultaneous feasible local planning and path-following control for autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–13, 2022.

[20] Fabian Poggenhans and Johannes Janosovits. Pathfinding and routing for automated driving in the lanelet2 map framework. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–7. IEEE, 2020.

[21] Jörn Knaup and Kai Homeier. Roadgraph-graph based environmental modelling and function independent situation analysis for driver assistance systems. In *13th International IEEE Conference on Intelligent Transportation Systems*, pages 428–432. IEEE, 2010.

[22] Kai Homeier and Lars Wolf. Roadgraph: High level sensor data fusion between objects and street network. In *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1380–1385. IEEE, 2011.

[23] Simon Ulbrich, Tobias Nothdurft, Markus Maurer, and Peter Hecker. Graph-based context representation, environment modeling and information aggregation for automated driving. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 541–547. IEEE, 2014.

[24] Markus Koschi and Matthias Althoff. Set-based prediction of traffic participants considering occlusions and traffic rules. *IEEE Transactions on Intelligent Vehicles*, 2020.

[25] J. Wei, J. M. Snider, T. Gu, J. M. Dolan, and B. Litkouhi. A behavioral planning framework for autonomous driving. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 458–464, 2014.

[26] M. Fu, W. Song, Y. Yi, and M. Wang. Path planning and decision making for autonomous vehicle in urban environment. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 686–692, 2015.

[27] CARLA Development Team. Carla simulator. Available at `https://carla.org/`.

[28] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28:91–99, 2015.

[29] Xinran Li, Kuo-Yi Lin, Min Meng, Xiuxian Li, Li Li, and Yiguang Hong. Composition and application of current advanced driving assistance system: A review. *arXiv preprint arXiv:2105.12348*, 2021.

[30] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3354–3361. IEEE, 2012.

[31] Google AI. Ava challenge. Available at http://research.google.com/ava/challenge.html.

[32] Will Maddern, Geoffrey Pascoe, Chris Linegar, and Paul Newman. 1 year, 1000 km: The oxford robotcar dataset. *The International Journal of Robotics Research*, 36(1):3–15, 2017.

[33] Zhengxia Zou, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey. *arXiv preprint arXiv:1905.05055*, 2019.

[34] A. Geiger P. Lenz C. Stiller R. Urtasun. Object tracking evaluation (2d bounding-boxes). Available at http://www.cvlibs.net/datasets/kitti/eval_tracking.php.

[35] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7794–7803, 2018.

[36] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[37] Harkirat Singh Behl, Michael Sapienza, Gurkirt Singh, Suman Saha, Fabio Cuzzolin, and Philip HS Torr. Incremental tube construction for human action detection. *arXiv preprint arXiv:1704.01358*, 2017.

[38] Mathew Monfort, Alex Andonian, Bolei Zhou, Kandan Ramakrishnan, Sarah Adel Bargal, Tom Yan, Lisa Brown, Quanfu Fan, Dan Gutfreund, Carl Vondrick, et al. Moments in time dataset: one million videos for event understanding. *IEEE transactions on pattern analysis and machine intelligence*, 42(2):502–508, 2019.

[39] Chunhui Gu, Chen Sun, David A Ross, Carl Vondrick, Caroline Pantofaru, Yeqing Li, Sudheendra Vijayanarasimhan, George Toderici, Susanna Ricco, Rahul Sukthankar,

et al. Ava: A video dataset of spatio-temporally localized atomic visual actions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6047–6056, 2018.

[40] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, et al. The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*, 2017.

[41] Rohit Girdhar, Joao Carreira, Carl Doersch, and Andrew Zisserman. Video action transformer network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 244–253, 2019.

[42] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299–6308, 2017.

[43] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):221–231, 2012.

[44] Graham W Taylor, Rob Fergus, Yann LeCun, and Christoph Bregler. Convolutional learning of spatio-temporal features. In *European conference on computer vision*, pages 140–153. Springer, 2010.

[45] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015.

[46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[47] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. Slowfast networks for video recognition. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6202–6211, 2019.

[48] Chao-Yuan Wu, Christoph Feichtenhofer, Haoqi Fan, Kaiming He, Philipp Krahenbuhl, and Ross Girshick. Long-term feature banks for detailed video understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 284–293, 2019.

[49] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[50] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *arXiv preprint arXiv:2007.14062*, 2020.

[51] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[52] Daniel Neimark, Omri Bar, Maya Zohar, and Dotan Asselmann. Video transformer network. *arXiv preprint arXiv:2102.00719*, 2021.

[53] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[54] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. In *arXiv preprint arXiv:1904.07850*, 2019.

[55] Xingyi Zhou, Vladlen Koltun, and Philipp Krähenbühl. Tracking objects as points. In *European Conference on Computer Vision*, pages 474–490. Springer, 2020.

[56] Xingyi Zhou. Centernet github. Available at https://github.com/xingyizhou/CenterNet.

[57] Haoqi Fan Yanghao Li Bo Xiong Wan-Yen Lo Christoph Feichtenhofer. Slowfast github. Available at https://github.com/facebookresearch/SlowFast.

[58] Siyu Chen. Acar github. Available at https://github.com/Siyu-C/ACAR-Net.

[59] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[60] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding? *arXiv preprint arXiv:2102.05095*, 2021.

[61] Jinkun Cao, Xinshuo Weng, Rawal Khirodkar, Jiangmiao Pang, and Kris Kitani. Observation-centric sort: Rethinking sort for robust multi-object tracking. *arXiv preprint arXiv:2203.14360*, 2022.

[62] Autodrive challenge - autodrive™ challenge.

[63] J. V. Frasch, A. Gray, M. Zanon, H. J. Ferreau, S. Sager, F. Borrelli, and M. Diehl. An auto-generated nonlinear mpc algorithm for real-time obstacle avoidance of ground vehicles. In *2013 European Control Conference (ECC)*, pages 4136–4141, 2013.

[64] U. Rosolia, S. De Bruyne, and A. G. Alleyne. Autonomous vehicle control: A nonconvex approach for obstacle avoidance. *IEEE Transactions on Control Systems Technology*, 25(2):469–484, 2017.

[65] K. Berntorp. Path planning and integrated collision avoidance for autonomous vehicles. In *2017 American Control Conference (ACC)*, pages 4023–4028, 2017.

[66] Qian Wang, Beshah Ayalew, and Thomas Weiskircher. Predictive maneuver planning for an autonomous vehicle in public highway traffic. *IEEE Transactions on Intelligent Transportation Systems*, 20(4):1303–1315, 2018.

[67] Houjie Jiang, Zhuping Wang, Qijun Chen, and Jin Zhu. Obstacle avoidance of autonomous vehicles with cqp-based model predictive control. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 001668–001673. IEEE, 2016.

[68] H. Guo, C. Shen, H. Zhang, H. Chen, and R. Jia. Simultaneous trajectory planning and tracking using an mpc method for cyber-physical systems: A case study of obstacle avoidance for an intelligent vehicle. *IEEE Transactions on Industrial Informatics*, 14(9):4273–4283, 2018.

[69] Mohamed Ashraf Gameleldin Daoud. Simultaneous local motion planning and control, adjustable driving behavior, and obstacle representation for autonomous driving. Master's thesis, University of Waterloo, 2020.

[70] Shaosong Li, Zheng Li, Zhixin Yu, Bangcheng Zhang, and Niaona Zhang. Dynamic trajectory planning and tracking for autonomous vehicle with obstacle avoidance based on model predictive control. *Ieee Access*, 7:132074–132086, 2019.

[71] Alberto Franco and Vitor Santos. Short-term path planning with multiple moving obstacle avoidance based on adaptive mpc. In *2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 1–7. IEEE, 2019.

[72] J. Changhao, H. Miaohua, and S. Liyang. An autonomous vehicle motion planning method based on dynamic programming. In *2020 17th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, pages 394–398, 2020.

[73] Robert Ritschel, Frank Schrödel, Juliane Hädrich, and Jens Jäkel. Nonlinear model predictive path-following control for highly automated driving. *IFAC-PapersOnLine*, 52(8):350–355, 2019.

[74] Huckleberry Febbo, Jiechao Liu, Paramsothy Jayakumar, Jeffrey L Stein, and Tulga Ersal. Moving obstacle avoidance for large, high-speed autonomous ground vehicles. In *2017 American Control Conference (ACC)*, pages 5568–5573. IEEE, 2017.

[75] Jiechao Liu, Paramsothy Jayakumar, Jeffrey L Stein, and Tulga Ersal. Combined speed and steering control in high-speed autonomous ground vehicles for obstacle avoidance using model predictive control. *IEEE Transactions on Vehicular Technology*, 66(10):8746–8763, 2017.

[76] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57, 2006.

[77] Nima Moshtagh. Minimum volume enclosing ellipsoid. https://www.mathworks.com/matlabcentral/fileexchange/9542-minimum-volume-enclosing-ellipsoid, 2022. [Online; accessed August 16, 2022].

[78] Michael J Todd and E Alper Yıldırım. On khachiyan's algorithm for the computation of minimum-volume enclosing ellipsoids. *Discrete Applied Mathematics*, 155(13):1731–1744, 2007.

[79] Philip Polack, Florent Altché, Brigitte Novel, and Arnaud de La Fortelle. The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles? pages 812–818, 06 2017.

[80] H.G. Bock and K.J. Plitt. A multiple shooting algorithm for direct solution of optimal control problems*. *IFAC Proceedings Volumes*, 17(2):1603–1608, 1984. 9th IFAC World Congress: A Bridge Between Control Science and Technology, Budapest, Hungary, 2-6 July 1984.

[81] Jose A Matute, Mauricio Marcano, Sergio Diaz, and Joshue Perez. Experimental validation of a kinematic bicycle model predictive control with lateral acceleration consideration. *IFAC-PapersOnLine*, 52(8):289–294, 2019.

[82] Joel AE Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. Casadi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019.

[83] Mohammad Al-Sharman, Rowan Dempster, Mohamed A. Daoud, Mahmoud Nasr, Derek Rayside, and William Melek. Self-Learned Autonomous Driving at Unsignalized Intersections: A Hierarchical Reinforced Learning Approach for Feasible Decision-Making. 9 2022.

[84] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

[85] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[86] Alessandro Tasora, Radu Serban, Hammad Mazhar, Arman Pazouki, Daniel Melanz, Jonathan Fleischmann, Michael Taylor, Hiroyuki Sugiyama, and Dan Negrut. Chrono: An open source multi-physics dynamics engine. In *International Conference on High Performance Computing in Science and Engineering*, pages 19–49. Springer, 2015.

[87] Elwan Héry, Philippe Xu, and Philippe Bonnifait. Consistent decentralized cooperative localization for autonomous vehicles using lidar, gnss, and hd maps. *Journal of Field Robotics*, 38(4):552–571, 2021.

[88] Sajjad Mozaffari, Omar Y Al-Jarrah, Mehrdad Dianati, Paul Jennings, and Alexandros Mouzakitis. Deep learning-based vehicle behavior prediction for autonomous driving applications: A review. *IEEE Transactions on Intelligent Transportation Systems*, 23(1):33–47, 2020.

[89] Lijun Yu, Yijun Qian, Wenhe Liu, and Alexander G Hauptmann. Argus++: Robust real-time activity detection for unconstrained video streams with overlapping cube proposals. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 112–121, 2022.

[90] Dario Piga, Marco Forgione, Simone Formentin, and Alberto Bemporad. Performance-oriented model learning for data-driven mpc design. *IEEE control systems letters*, 3(3):577–582, 2019.

[91] Guillem Torrente, Elia Kaufmann, Philipp Föhn, and Davide Scaramuzza. Data-driven mpc for quadrotors. *IEEE Robotics and Automation Letters*, 6(2):3769–3776, 2021.

APPENDICES

Appendix A

Robo-Taxi Requirements

Acknowledgement: The following requirements were outlined by the SAE AutoDrive Challenge I Year 4 organizing committee.

A.1 Reacting Properly to Signals

- Green: Decelerate if necessary for acceleration metrics, but do not stop.
- Solid Red: Come to a complete stop within 3 meters of the limit line preceding the intersection, with no part of the vehicle past the limit line. Remain stopped until legally allowed to proceed. The vehicle must begin movement within 5 seconds of the light turning green.
- Flashing Red: Come to a complete stop within 3 meters of the limit line preceding the intersection, with no part of the vehicle past the limit line. Remain stopped until legally allowed to proceed.
- Yellow: Vehicle can either follow the rules for Green or Solid Red light. Vehicle must enter the intersection before the light turns.

A.2 React Properly to Traffic Control Signage

- Stop Sign: Come to a complete stop no more than 3 meters before any applicable limit line, and proceed into the intersection when safe and legal to do so.

- Turn Only: Complete a required turn as mandated by a Left Turn Only or Right Turn Only sign.
- Speed Limit Sign: Do not exceed the posted speed limit[2]. Roads with no posted speed limit signs or 25 mph signs are not scored for this metric.
- Do Not Turn: Do not turn the direction specified by the sign.
- Do Not Enter: Do not enter the roadway specified by the sign.

A.3 Avoid Static Objects

Avoid colliding with any static object obstructing the vehicle's path of travel.

A.4 Avoid Dynamic Objects

- Pedestrians: When required to stop, vehicles must stop before the limit line preceding the crosswalk, with no part of the vehicle past the limit line, and allow the pedestrian to completely cross the roadway before proceeding.
- Other Dynamic Objects: Avoid colliding with any dynamic objects obstructing the vehicle's path of travel.

A.5 Intersection Lane Selection

Vehicle crosses into the intersection while inside the proper lane to execute the desired turn, and exits the intersection into the appropriate lane.

Appendix B

ROS Message Definitions

B.1 Sensor Messages

Messages produced by sensors.

B.1.1 PointCloud.msg

See http://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/PointCloud.html

B.1.2 Image.msg

See http://docs.ros.org/en/noetic/api/sensor_msgs/html/msg/Image.html.

B.2 Perception Messages

Messages produced by the perception stack.

B.2.1 Obstacle.msg

```
Header header
```

```
# Obstacle Type Enums
string OBS_TP_UNKNOWN=UNKNOWN
string OBS_TP_PED=PEDISTRIAN
string OBS_TP_CYC=CYCLIST
string OBS_TP_VCL=VEHICLE
string label # see Obstacle Type Enums

# Detection confidence
float32 confidence

# Position and its uncertainty
# For 3d bounding boxes, the (x, y, z) is the center point of the 3d bounding box
# For 2d bounding boxes, the (x, y) is the top left point of the 2d bounding box
geometry_msgs/PoseWithCovariance pose

# Velocity and its uncertainty
geometry_msgs/TwistWithCovariance twist

# Dimensions of bounding box assuming BEV perspective
# x=width, y=height, z=depth
# For example, a vehicle with its bumper facing North that's
# encapsulated by a rectangular prism defines width as the
# E/W measurement, height as the N/S measurement.
# Obstacle.msg is also used as 2d bounding boxes
# In that case width_along_x_axis is the width of the 2d bounding box
#   in image coordinates
# and height_along_y_axis is the height of the 2d bounding box
#   in image coordinates
# z axis is not used with 2d bounding boxes
float64 width_along_x_axis
float64 height_along_y_axis
float64 depth_along_z_axis

# Unique ID number
uint32 object_id
```

B.2.2 TrafficLight.msg

```
Header header

# Traffic Light State Enums
string TL_ST_NON=NON
string TL_ST_RED=RED
string TL_ST_YEL=YELLOW
string TL_ST_GRE=GREEN
string TL_ST_FLA=FLASHING_RED
string left
string forward
string right

# Traffic Light Sign Direction Enums
string TL_DIR_NON=NON
string TL_DIR_LT=LEFT
string TL_DIR_RT=RIGHT
string TL_DIR_FD=FORWARD
string TL_DIR_LT_FD=LEFT_FORWARD
string TL_DIR_RT_FD=RIGHT_FORWARD
string sign_dir

geometry_msgs/Pose pose
geometry_msgs/Vector3 dimensions

geometry_msgs/Point p1
geometry_msgs/Point p2

common_msgs/StopLine stop_line

# Unique ID number
uint64 id
```

B.2.3 OccupancyGrid.msg

See http://docs.ros.org/en/melodic/api/nav_msgs/html/msg/OccupancyGrid.html.

B.2.4 TrackedObstacle.msg

```
Header header

common_msgs/Obstacle obstacle

common_msgs/TrackedObstacleState[] observation_history
common_msgs/TrackedObstacleState[] predicted_states
```

TrackedObstacleState.msg

```
# Attributes of the TrackedObject at every observation/prediction
# To be published in the Odom frame

Header header
# bounding box centroid (m) and orientation
geometry_msgs/Pose pose
# Velocity (m/s)
geometry_msgs/Twist velocity
```

B.3 Navigation Messages

Messages used by the navigation stack.

B.3.1 DestinationList.msg

```
# Ordered sequence of destinations that the ego vehicle should proceed through
# In competition this list will be hardcoded and published via the command line
# In RVIZ the 2D Nav Goal Tool is converted to a single "x,y" destintation
string[] destination_list
```

B.3.2 Odometry.msg

See http://docs.ros.org/en/noetic/api/nav_msgs/html/msg/Odometry.html.

B.3.3 Reference.msg

```
Header header

geometry_msgs/Point[] ref_line
geometry_msgs/Point[] left_bound
geometry_msgs/Point[] right_bound

path_planning_msgs/PredictedTraj[] trajectories
```

B.3.4 DesiredOutput.msg

```
Header header
float32 theta
float32 torque
# Two states: 0 = stopped, 1 = driving
int8 state
```

Glossary

Kalman Filter A statistical based filtering method that combines a measurement model with an observation model to minimize state uncertainty.

Novatel SPAN A positioning solutions company headquartered in Calgary, Alberta, Canada which supplies GNSS and INS solutions. They supplied SAE AutoDrive Challenge I teams with these solutions. Bolty uses them as its localization solution. See https://novatel.com/support/span-gnss-inertial-navigation-systems for details.

Docker A software isolation library that provides network isolation and reproducible execution environments via containers. See https://www.docker.com/ for details.

ROS The Robotic Operating System. Allows for language agnostic communication of modular software systems. See https://www.ros.org/ for details.

CARLA An AD simulator. See https://carla.org/ for details.

Visual Studio Code A popular open source IDE with high capacity support for remote development and Docker container based development. See https://code.visualstudio.com/ for details.

Carla ROS Bridge A middleware software package which allows for interfacing between the backend CARLA server and the frontend software modules that run ROS. Operates by querying the backend server for environment information, and translating that information into ROS messages that are then broadcasted to the frontend software modules over the ROS network. See https://github.com/carla-simulator/ros-bridge for details.

RViz A popular open source robotics data visualization software. See http://wiki.ros.org/rviz for details.

Lanelet2 Lanelet2 is a C++ library for handling map data in the context of automated driving. It is designed to utilize high-definition map data in order to efficiently handle the challenges posed to a vehicle in complex traffic scenarios. See `https://github.com/fzi-forschungszentrum-informatik/Lanelet2`.

JOSM JOSM is an extensible editor for OpenStreetMap (OSM) for Java 8+. It supports loading GPX tracks, background imagery, and OSM data from local sources as well as from online sources and allows to edit the OSM data (nodes, ways, and relations) and their metadata tags. See `https://josm.openstreetmap.de/`.

CasADi CasADi is an open-source tool for nonlinear optimization and algorithmic differentiation. See `https://web.casadi.org/` for details.