

To the Graduate Council:

I am submitting herewith a dissertation written by Joseph William Beck entitled “Advancing Safety Validation & Testing of Autonomous Vehicles Using Advanced Simulation Techniques.”

I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Mechanical Engineering.

Subhadeep Chakraborty, Major Professor

We have read this dissertation
and recommend its acceptance:

Dr. Subhadeep Chakraborty

Dr. Asad Khattak

Dr. Hairong Qi

Dr. Zhenbo Wang

Accepted for the Council:

Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Advancing Safety Validation & Testing of Autonomous Vehicles Using Advanced Simulation Techniques

A Dissertation Presented for the
Doctor of Philosophy
Degree
The University of Tennessee, Knoxville

Joseph William Beck

December 2024

© by Joseph William Beck, 2024
All Rights Reserved.

*To my family, friends, and everyone who has been with me on this journey,
thank you*

Abstract

Even as extensive efforts are put in to the research and development of self-driving vehicles, many safety concerns persist. On-road testing presents unrealistic costs for manufacturers, leading to a development cycle that moves into deployment before sufficient validation has occurred. Additionally, there are a multiplicity of driving scenarios that combine in various ways to define what safe driving behaviors look like, and it is not clear what finite set of scenarios would constitute a valid safety test. In this work, we thoroughly examine virtual simulation as a tool to overcome these obstacles toward the validation of self-driving vehicles. There are two primary threads that tie all of the aspects of this work together: diagnostic clarity and realism. Both of these concepts are explored through original research and novel approaches, expanding the capability of virtual simulation as a tool for validation of real-world self-driving cars.

Table of Contents

1	Introduction	1
1.1	Overview	1
1.2	Background & State of the Art	2
1.2.1	Vehicle Simulation Platforms & Tools	2
1.2.2	Analysis of Accident Reports	4
1.2.3	Validation & "Edge-Case" Discovery	5
1.2.4	Time-series Generative Models	7
1.2.5	Vehicle Perception & Label-Based Models	9
1.3	Research Objectives	11
2	Simulation as a Tool: Sensors and Accident Reconstruction	13
2.1	Objective	13
2.2	Study Design	14
2.2.1	Framework	14
2.2.2	California tester program	15
2.2.3	Real World Scenarios	15
2.3	Methodology	19
2.3.1	Complete Data Pipeline	19
2.3.2	Data Processing	21
2.4	Perception Simulation on KITTI Dataset	23
2.4.1	Location Estimating Demo	24
2.5	CARLA Simulator	28
2.5.1	Simulation Scenarios	28
2.5.2	Simulated Sensor Data	29
2.5.3	Data Visualization	31

2.5.4	Data analysis for detailed crash reports	32
2.6	Limitations	33
2.7	Summary	33
3	AV Behavior Diagnosis, Prediction, and "Edge-Case" Discovery	37
3.1	Objective	37
3.2	Methodology	38
3.2.1	Virtual World	39
3.2.2	Perception Model & Controller	39
3.2.3	Vehicle-in-the-loop Simulation	41
3.2.4	Software-in-the-loop Simulation	43
3.2.5	Model-in-the-loop Simulation	44
3.3	Experimental Setup	44
3.3.1	Driving Types	45
3.3.2	Weather, Lighting, & Lead Vehicle	47
3.3.3	Safety, Performance, & Predictive Metrics	48
3.4	Results	50
3.4.1	Performance Results	51
3.4.2	Response Diagnosis	53
3.4.3	<i>MIL</i> Simulation for Response Prediction	55
3.5	Summary	59
4	Fully Embedded Time-Series Generative Adversarial Networks	60
4.1	Objective	60
4.2	Proposed Architecture	61
4.3	Problem Formulation	63
4.3.1	Adversarial Autoencoder	64
4.3.2	First Above Threshold (FAT) Operator	65
4.3.3	Complete Model	68
4.4	Implementation	69
4.4.1	Experimental Setup & Datasets	70
4.4.2	Distribution Matching	70
4.4.3	Selective Sampling	71

4.4.4	Performance Results	71
4.5	Limitations & Ethical Considerations	73
4.6	Summary	75
5	Improving Realism Through Perception Error Models	76
5.1	Objective	76
5.2	Problem Formulation	77
5.2.1	Modified <i>FETSGAN</i>	77
5.2.2	Perception Error Models	78
5.3	Methodology	79
5.3.1	Perception Model & Dataset	79
5.3.2	Simulation Strategies	82
5.3.3	Experimental Setup	83
5.4	Results	84
5.4.1	Training Dynamics	86
5.4.2	Model Validation	86
5.4.3	Vehicle Response	90
5.5	Summary & Limitations	98
6	Conclusion	99
6.1	Original Contributions	99
6.2	Impact	100
6.3	Limitations & Future Work	100
	Bibliography	102
	Vita	116

List of Tables

2.1	Key Statistics of AV Crashes in California from 2019 to 2020 (N=148)	17
2.2	Information on selected AV-involved crashes	18
2.3	CARLA Simulation Accident information	30
2.4	Overview of the data input for Simulation #1 and Simulation #2.	30
3.1	Number of experimental runs n conducted (<i>VIL</i> , <i>SIL</i> , and <i>MIL</i>)	46
3.2	Pearson Correlation Coefficient (PCC) statistical test results	58
4.1	Predictive & Discriminative Scores	74
5.1	Quantitative Comparison: Real $\mathcal{D}$ vs. PEM $\hat{\mathcal{D}}$	91
5.2	Predictive & Performance metrics for VIL, SIL, and MIL experiments	96

List of Figures

2.1	Framework to incorporate AV data into crash investigation analysis	16
2.2	Scenes of crash reconstruction	20
2.3	Proposed data pipeline for accident analysis.	20
2.4	Data post-processing procedure	26
2.5	KITTI visualization of YOLO, LiDAR, and <i>Adabins</i> depth detection	26
2.6	KITTI object tracking demonstration	27
2.7	CARLA Simulation #1	35
2.8	CARLA Simulation #2	35
2.9	Qualitative Analysis is performed for CARLA Simulation #1	36
3.1	Data Flow of each simulation modality	40
3.2	Complete <i>VIL</i> Simulation setup.	42
3.3	Experimental setup, weather, & lighting	46
3.4	Performance metrics	52
3.5	Raw data for the stopping experiments	54
3.6	<i>VIL</i> predictive metrics are plotted against <i>VIL</i> response metrics.	56
3.7	<i>MIL</i> predictive metrics are plotted against <i>VIL</i> response metrics.	56
3.8	Results for the <i>MIL</i> simulation experiments.	58
4.1	An overview of the <i>FETSGAN</i> training scheme	62
4.2	A demonstration of the utility of the <i>FAT</i> operation is shown for the Sines dataset	67
4.3	Qualitative visualization results	72
4.4	A demonstration of selective sampling of the prior distribution p_z	72
5.1	An image from the <i>Comma2k19</i> dataset	81
5.2	The CARLA setting for the SIL experiments	85
5.3	Comparison of Lead Distance and Lead Velocity (raw model outputs) across time	88

5.4	Response and perception are shown for the Stopping experiments of Section 3.3.1	93
5.5	Predictive and Performance metrics are analyzed for VIL experiments for all three experiment types.	95

Chapter 1

Introduction

1.1 Overview

Research and Development of self-driving vehicles has been well underway since the *DARPA* Grand Challenges [1] twenty years ago. The undertaking of bringing autonomous vehicles (AVs) to the public at large represents a massive interdisciplinary engineering challenge. Sensing, perception, robotic control, and all the powertrain components of modern vehicles have had massive technological leaps during that time span. However, trepidation still exists in the public consciousness around the safety of self-driving technology. Currently, 93% of Americans have concerns about the safety and reliability of self-driving technology [2], with noteworthy psychological barriers to accepting such a dramatic shift in their approach to driving [3]. Indeed, driving without constant supervision requires a high degree of trust. As more advanced self-driving technology moves directly from development to deployment, accidents and unexpected behavior can sow distrust, and perhaps move us further away from adoption of self-driving technology. To rectify this, it is crucial that the transportation industry develop robust validation procedures that provide confidence the vehicle will respond safely and predictably to the near infinite number of scenarios encountered in every-day driving. It is unrealistic to expect this testing to be performed entirely on a physical vehicle in a closed-track setting. The costs associated with performing such tests are extremely high. Additionally, it is simply not possible to physically recreate a sufficient number of the possible situations a vehicle may encounter in a closed-track setting. Here, virtual simulation is proposed as a tool to address both of these problems. Virtual simulations can be created cheaply to cover a far wider range of scenarios than what is available to be created on a track. Additionally, virtual

simulation can be performed at a multitude of different levels, involving simplified vehicle models and controllers, or even the entire vehicle itself. In this work, we explore simulation as a tool for the safety validation of AVs, and work to overcome the challenges specific to this approach.

Given the stated goal of producing a simulation platform for validating the safety of AVs, there are a variety of unique challenges to that approach. We consolidate all of those challenges into two primary dimensions: *realism* and *diagnostic clarity*. Realism is the straightforward idea that the behavior of the vehicle in simulation should resemble the behavior of the same vehicle under identical conditions in the real-world. We can quantify this idea by borrowing the concept of *domain shift error* from machine learning literature [4, 5]. Maximizing realism is to minimize the domain shift error from the physical world modality to the virtual modality, as measured by the behavior of the vehicle. Unfortunately, it is likely that realism in simulation may not be enough to produce a valid platform for AV validation. It is commonly cited that around 9 billion miles of driving are needed to validate self-driving vehicles to the same levels of safety as human drivers [6]. Even in a virtual simulation setting, that much testing is a tall order. Validation procedures that account for *how* AVs fail instead of simply noting *when* they fail could provide the key to producing a finite set of scenarios that adequately assuage safety concerns. To that end, diagnostic clarity refers to the idea that when the vehicle is under test, causality should be apparent from the data that is gathered. To the extent possible, a causal understanding should exist throughout the entire informational flow from environment, perception, planning, control, and dynamic response. Transparency with regard to how systems fail provides an opportunity to narrow the scope of validation, allowing for testing of individual subsystems and a richer understanding of the safety pitfalls in any individual system. All of the original work described in this thesis is intended to expand the state of the art in virtual simulation strategies for AVs along one or both of these dimensions.

1.2 Background & State of the Art

1.2.1 Vehicle Simulation Platforms & Tools

At the heart of this research is the development of tools for virtual simulation of self-driving vehicles. The capabilities and limitations of these tools wholly define the kind of research that can occur, as well as the scope of the claims that can be made toward real-world impact.

There are a handful of virtual worlds that are made available for research purposes. These worlds provide a physics engine for vehicle dynamics, a sensor simulation suite for producing raw sensor data such as camera images and LiDAR, virtual maps for creating realistic driving scenarios, and an application programming interface (API) for allowing researchers to control aspects of the simulation. The primary virtual environment used in this research is CARLA [7]. CARLA has been heavily developed over the course of the last 8 years, with relatively low performance requirements and open-sourcing that provide a low barrier to entry. LGSVL [8] is a similar solution. While providing a shorter feature list than CARLA, it is known for easy integration with the autonomous driving toolkit Apollo [9]. There are also some licensed virtual environments in the market. IPG Carmaker [10] and dSpace ASM [11] provide solutions more tailored for an automotive development cycle than research, but provide many of the same tools. In addition to the virtual simulator itself, it is common to implement a toolset for self-driving functions such as perception, path planning, and control, which eliminate the need to develop the entire AV software stack. Primarily, there are two such toolkits: Apollo [12] and Autoware [13]. Both systems are designed to be run on physical hardware and are built in the ROS ecosystem [14]. For our own purposes, we developed an adapted version of the *openpilot* open-source SAE Level 2 controller [15]. While Apollo and Autoware offer a more complete feature set, *openpilot* is lightweight and easier to implement on existing vehicle hardware. This allows for a greater degree of flexibility with respect to simulation realism, depending on the research objectives.

“Virtual simulation” can be a slightly ambiguous term, because simulation can occur at a variety of different fidelities. While pertinent information can be gathered regardless of fidelity, it is important to note the spectrum of realism in state-of-the-art solutions. While there are no industry standard terms to describe the fidelity level of virtual simulators, The automotive industry has coined three types of simulation with respect to a hardware development cycle: model-in-the-loop (*MIL*) [16], software-in-the-loop (*SIL*) [17], and hardware-in-the-loop (*HIL*) [18], where cost and fidelity generally increase with each respective modality. Vehicle movements and sounds can be simulated with a varying level of realism. In a lower fidelity system, the driving simulator is usually a screen-based desktop simulator which has a limited FOV and the virtual vehicle is controlled via a set of steering wheel and pedals without a motion base. The National Advanced Driving Simulator (NADS-1) and NADS MiniSim at the University of Iowa are separate examples of high-fidelity and medium-fidelity driving

simulators [19]. In the context of full-vehicle simulation, we also adopt the vehicle-in-the-loop (*VIL*) nomenclature. In *VIL* simulation, the entire vehicle is incorporated into the simulation loop, where hardware restraints and real-vehicle dynamics lead to real vehicle response to virtual stimulus. The work described here follows similar methodologies as [20], where a vehicle in an open environment running Autoware [13] is used in a virtual-reality approach to vehicle simulation. There is also MCity[21], a closed track environment for AV testing, with multiple mixed reality tools available. For our purposes, an in-house modified version of *openpilot* [15] works specifically as a *VIL* simulation controller with a vehicle on a dynamometer. This combination represents a novel tool for high fidelity simulation at a relatively low cost, and short development time compared to similar solutions found in the research community.

1.2.2 Analysis of Accident Reports

In this work, we explore an approach to AV safety validation that takes into account a specific *diagnosis* of what caused the vehicle to respond the way that it did. Toward that end, literature in automotive accident analysis is a natural starting point. This literature has widely discussed the association of factors contributing to crashes and has shown that human error contributes to more than 90% of crashes. However, in the majority of such crashes, more than one factor contributes to crash occurrence. Generally, human errors and failures are considered the main factor, ignoring other reasons and confounding factors between them, especially the surrounding environment and vehicle characteristics. Several studies attempted to perform crash investigations using police report data [22, 23, 24]. Event Data Recorders (EDRs) are one of the most important data sources for diagnosing crashes, and they are estimated to be in most of new vehicles with procedures for accessing and analyzing the data. Much information can be retrieved from an EDR including time, speed, changes in speed (ΔV), throttle and brake position, seatbelt, and airbag status. Several studies utilized EDR to integrate pre-crash vehicular movements in their analysis [25, 26, 27, 28, 29]. For example, [29] studied the drivers' pre-crash maneuvers in terms of timing and kinematics using 5 seconds of vehicle speed, brake, and yaw rate data before crashes. Other studies have used Principal Direction of Force (PDOF) recorded by the EDR as a surrogate measure to estimate the damage side of crashes.

One limitation of EDR devices is that they only collect data from the subject vehicle, ignoring the surrounding vehicles, environment, and driver behavior. Furthermore, EDR does not account for emerging advanced driver assistance systems (ADAS), which allow a driver to dynamically transfer vehicle control responsibilities to a driving assistant system for prolonged periods. As low-level automated vehicles gradually penetrate the market, new data sources generated by such vehicles enable researchers to perform novel analysis. Using such data, they were able to determine the times and locations where autopilot systems were used, instances where warnings were generated to the driver, and the times where the driver interacted with the steering wheel prior to the crash. Sensor data can also be used to analyze accidents and perform accident prevention on a case-by-case basis. [30] demonstrate the conditions from which sonar can replace rear-view cameras as an effective means to prevent collisions with pedestrians, while [31] show how vehicle kinematics gleaned from advanced sensing can be used to provide additional insight in accident analysis. The recent research that highlights the utility of advanced sensor data in accident analysis justifies the work done in this research to organize, visualize, and explain how this sensor data could be collected and distributed to interested parties. AV instrumentation with a wide range of sensors help us collect unique data and incorporate it into the current state of the art. Focusing on safety analysis, a few studies have used roadside and vehicle LiDAR data to process vehicular safe movements and the environment [32], track pedestrians and vehicles [33], identify occurrence of a crash and near-crash events [34, 35], and predict crash parameters [36]. By reviewing the literature, we can infer that there is a lack of framework dedicated to utilizing data generated by LiDAR, camera, and radar toward diagnosing the behavior of AVs and the surrounding vehicles, particularly leading up to a crash. Exploring the sensor data available during crashes involving self-driving vehicles is not only a step toward advancing our goal of validating these vehicles for safety, but providing tools for understanding why and how they may fail.

1.2.3 Validation & "Edge-Case" Discovery

The safety validation of AVs generally refers to providing some guarantees or assurances against unexpected behavior of the system. From literature, safety guarantees have been broadly pursued along two disparate paths: (i) *robustness of perception systems* when subjected to adversarial or benign/natural perturbations, and (ii) *robustness of control systems* that leverage reinforcement learning (RL). Oftentimes when discussing vehicle safety, we are not

concerned with trivial scenarios that might be straightforwardly safe or obviously dangerous. Instead, most validation efforts attempt to find "edge-case" scenarios, that are at the limits of what the vehicle is expected to perform safely. Any system that pushes this boundary toward a human-driver level of competence can be considered robust. Adversarial robustness of image recognition systems have been investigated extensively over the past five years [37, 38] under a variety of settings; more recently the focus has shifted towards natural robustness [39, 40]. A similar line of research can also be found for RL models, primarily with respect to adversarial perturbations to state/observation and action spaces, with different approaches pursuing various styles of robustness objectives [41, 42, 43, 44, 45, 46, 47]. However, the large majority of the above approaches use somewhat ad-hoc notions of safety/robustness, and do not explicitly address system-wide safety and especially concepts of safety aligned to human values.

Another exciting direction for ensuring safety in robot behavior is the use of formal methods [48, 49, 50]. Addressing concerns of predictability and safety in learned robot behaviors, [51] proposed a formal methods approach to interpretable RL for robotic planning, leveraging formal methods and control theory. To ensure safety in the RL agent behaviors, the approach provides a formal specification language to explicitly define undesirable behaviors and uses control barrier functions (CBF) to enforce safety constraints. Similarly, in [48], the authors present an approach that uses formal methods to assess the safety of an RL agent by designing moving adversaries that identify potential safety violations and implementing defense mechanisms to strengthen the agent's policy. Moreover, in [49] the authors present a technique that combines the exploration and optimization capabilities of learning with the safety guarantees of formal verification to ensure safe control through proof and learning. Overall, these papers highlight the importance of safety in RL for robotic planning and offer promising approaches to ensure the safe behaviors of RL agents.

Comparing to our work, it is fairly common in literature to leverage virtual environments toward the goal of AV validation. Given the flexibility virtual environments have for providing a wider range of environmental scenarios than on-track testing, these types of simulation have increasingly become tools for edge-case discovery, with recent work leveraging virtual platforms for insights into AV behavior including accident automated edge-case discovery [52] and the development of formalized safety-based scenario definers like *Scenic* [53]. However, the vast majority of the validation literature focuses on edge-case discovery as an end-to-end endeavor, where the only noteworthy metrics are in the form of the dynamic vehicle response.

While we also focus on these metrics in our work, we believe it is important to also *diagnose* poor vehicle performance, and understand how individual subsystems contribute to response. We believe that exploring the causal links between these subsystems, even for only a single software stack in *openpilot*, represents a noteworthy literature gap.

1.2.4 Time-series Generative Models

A large part of this work is centered around advancing vehicle simulation fidelity with generative models [54], described fully in Chapter 5. To that end, we found a significant literature gap in the field of generative models for real-valued time-series data, which led to the formulation of the original architecture described in Chapter 4. As a generative model, our original work builds on the adversarial training scheme of RCGAN [55] with mixed supervised and unsupervised learning elements, along with a Recurrent Neural Network (RNN) style architecture for the Encoder, Decoder, and Discriminator, which are common for many sequence modeling tasks [56]. The work most closely matching this description is *TimeGAN* [57], where the key distinction is our use of an adversarial autoencoder (AAE) [58] applied to encoding full time sequences, as opposed to the element-wise encodings in TimeGAN. There is also the application specific R-GAN which combined some time-series heuristic information like Fourier representations with a WGAN approach to produce synthetic energy consumption data [59], [60]. COT-GAN is a recent approach to this problem that defines an adversarial loss through an expansion of the Sinkhorn Divergence into the time domain [61]. Due to the inherent instability of adversarial training, [62] regularizes training with a contrastive model and a training scheme grounded in imitation learning. Finally, [63] incorporates two separate approaches using diffusion models and more traditional constrained optimization techniques to produce time-series data. Beyond producing data for the purpose of vehicle simulation, real-valued time-series data generation has a range of applications in fields such as medicine [64, 65], energy management [66, 67], and sensor simulation [68]. Additionally, the use of transformers [69] has become very popular in the field of sequence generation, particularly with respect to large language models (LLMs) such as the popular Generative Pre-trained Transformer (GPT) architecture [70]. While we did experiment with transformer-based approaches, we found that the method described in Chapter 4 outperformed these methods.

We postulate the difference in performance was due to transformer-based methods being autoregressive in nature, exposing the model to inference bias that was not present in our non-autoregressive approach.

Beyond the direct comparisons with other models that produce realistic time-series data, our novel approach also incorporates an interpretive latent space, allowing the selective sampling of the posterior distribution at inference time to reach some desired effect, specified by the user. This bears a direct connection to the field of representation learning, where data is compressed in a meaningful way in order to accomplish some downstream task. This has been accomplished on real valued time series data, as data is embedded in RNN style architectures for the purpose of forecasting [71], supervised learning [72], and data imputation [73]. Perhaps the most well-known interpretive generative models are variational autoencoders (VAEs) [74]. The alternative AAE used here has a close resemblance to this approach, as described in [58], with the primary benefit in our use case being the ability to choose an arbitrary prior distribution instead of a standard Gaussian. The simplistic AAE used in this work has also been extended to utilize Wasserstein loss [75] in the image-generation space, demonstrating some improved stability that is typical of WGANs.

A theme of our work and all recent work on the creation of synthetic time-series is the regularization or complete abandonment of adversarial training [4] as a means to produce more stability, and thus higher quality data. Obviously, this challenge is not unique to the time-series domain, and inspiration can be drawn from any generation process that utilizes adversarial learning, particularly image creation. Given the *seq2seq* translation element described in our work, it stands to reason inspiration could be drawn from image-to-image translation techniques. We can see that the regularization effect of reconstruction loss or cycle loss is present in many adversarial approaches [76], [77]. WGANs [59] and LSGANs [78] focus primarily on the output layer of the discriminator and the loss function to produce a regularizing effect on reducing exploding or vanishing gradients. Spectral normalization [79], particularly in the weights of the discriminator, assures Lipchitz continuity in the gradients, further ensuring training stability. While not directly applicable to the RNN architecture itself, batch normalization in the discriminator has also demonstrated an ability to speed up adversarial training [80]. In our work, spectral normalization and batch normalization are effectively used to regularize training of our architecture. Ultimately, we found there is a gap in literature for generative time-series models that adequately capture the complex temporal

relationships in real-valued data. By addressing this gap, we provide a tool for generative modeling toward autonomous vehicle development and many other fields as well.

1.2.5 Vehicle Perception & Label-Based Models

In the context of self-driving vehicles, we refer to the perception as the robotic subsystem that takes as input high dimensional sensor data such as LiDAR, camera images, and radar data. In the informational flow of a self-driving vehicle, the perception subsystem represents a kind of pivotal threshold. Upstream, complex driving scenarios and high dimensional sensor data are much more likely to be highly stochastic, prone to noise and uncertainty. Downstream, the path planning, control, and vehicle response are likely to be much more deterministic and easier to model. With an eye toward safety validation, understanding the behavior of the perception system is a primary focus of most of our work. In the case of end-to-end driving [81], the perception system encompasses the path-planning system as well as the control system, and the output of this system is actuator commands. More commonly, the perception system compresses high-dimensional sensor data into low-dimensional labels. These labels are then used as environmental descriptors to inform the path-planning and control systems [82]. 2D object detection methods such as the YOLO architecture [83, 84, 85] are common in robotic perception. They take in images from a camera and return bounding boxes around objects of note in the scene, such as pedestrians and other vehicles. It is not uncommon for images to also be used for depth estimation, where each pixel in the output image corresponds to a depth estimation for the input image [86, 87]. Often, a temporal element is necessary for detecting elements in the environment. LiDAR and vision-based object tracking perform object detection, while also attempting to assign identifiers to each detected object [88, 89]. Finally, it is common for many models to feature a combination of these tasks. For instance, the *Supercombo* model [90] featured in this work combines object detection, object tracking, lane detection [91], and depth estimation.

All the perception systems described here are intended to function on vehicles that operate in the physical world. The domain shift error when these perception systems face synthetic sensor data instead of real sensor data is undefined, and thus, it is a challenge to apply safety validation procedures in simulation while making any definitive statements regarding the applicability to real-vehicles. For this reason, we are not deeply concerned with the neural-network based mechanisms that underpin these perception models. While the field of transfer

learning attempts to address these types of problems [92, 93, 94, 77], we analyze a different approach. Instead, we would like to analyze the format of the labels that these perception models produce. Perception Error Models (PEMs) [95] define a methodology for modeling a perception system, irrespective of sensor modality, as the sum of its ground truth label and a stochastic error term. This produces a model that can theoretically provide a probabilistic representation of the vehicle perception system that encompasses the full distribution of possible output. This type of model is ideal for simulation, where ground truth labels such as depth information and bounding boxes are inherently known. This approach has been used sporadically in literature. [96] uses a Hidden Markov Model (HMM) approach to modeling perception errors. Polar defined regions around the vehicle are defined, while the transition matrix of the HMM defines whether or not objects are detected, and emission probabilities model the error term that defines the deviation from the ground truth. This model is then applied to situational driving scenarios in Apollo and LGSVL [9, 8]. This approach does come with certain theoretical guarantees on the distribution of the error, but by simply sampling gaussian distributions at each time-instance, more complex temporal relationships in the error term are disregarded. This is a theme in other applications found in literature. [97] uses a simple feedforward neural network at each time instance to model the bounding box error of an object detector. Other approaches define PEMs to approach modeling LiDAR from various different ways [98, 99]. Most of the approaches in current literature are not focused on the fidelity of the PEM model. Rather, most are concerned with how a probabilistic error model can be leveraged to discover safety edge-cases. [100] looks to apply a known set of failure modes across an error distribution to measure vehicle response. [101] uses importance sampling on a simple PEM at models missed detections to quickly determine edge-case scenarios as an alternative to Monte-Carlo sampling. Finally, [102] uses an RCGAN [103] approach to modeling PEMs, but notes significant deviation in the synthetically reproduced error model, and does not apply this PEM model to a vehicle within simulation. Many of the models described lack a high fidelity approach to capturing subtle temporal relationships in the PEM error signal. This is due to the literature gap in strong time-series generative models for real-valued data described in Section 1.2.4. Additionally, there is a gap in applying PEM models in highly realistic vehicle-in-the-loop environments described in Section 1.2.1.

1.3 Research Objectives

There are three primary chapters corresponding to the original work that forms this thesis. Each chapter, along with how the work within fits into its respective literature gap, corresponds to an objective listed here.

1. **Simulation as a Tool: Sensors and Accident Reconstruction** (Chapter 2): A virtual simulation strategy is explored from a "data first" perspective, showing how sensor data from an AV can be analyzed for a rich understanding of vehicle response. A high level sensor simulation strategy is also used to explore simulation as a tool for diagnosing and understanding recreations of real-world accidents that involve self-driving vehicles. This work advances the idea of incorporating intermediate information to fully understand and diagnose the behavior of autonomous vehicles.
2. **AV Behavior Diagnosis, Prediction, and "Edge-Case" Discovery** (Chapter 3): Using model-in-the-loop (*MIL*), software-in-the-loop (*SIL*), and vehicle-in-the-loop (*VIL*) simulation strategies, we explore the causal links between perception, vehicle control strategy, and vehicle response. The effects of weather, lighting, driving type, and other objects in the environment are correlated against the behavior of the vehicle from a safety context in search for safety "edge-cases". Using our custom implementation of the *openpilot* controller, we also form causal links between the perception system and vehicle response.
3. **Fully Embedded Time-Series Adversarial Networks** (Chapter 4): With the goal of using generative models to advance realism in AV simulation, our new approach *FETSGAN* is formulated. We found a significant literature gap in generative modeling for real-valued time-series data. In an effort to produce generative models that fully capture complex temporal distributions, we outline an original neural network as a standalone aspect of our work. An adversarial autoencoder structure is combined with our *first above threshold* formulation for results on standard time-series datasets that advance the state-of-the-art.
4. **Improving Realism Through Perception Error Models** (Chapter 5): Our new approach to the simulation of real-valued time-series signals from Chapter 4 is explored in the context of vehicle perception systems. Perception Error Models (PEMs) are used

to stochastically model the performance of a vehicle's perception system, bypassing the laborious task of simulating sensor data itself. The resulting model is combined with the tools from Chapter [2](#), demonstrating a simple and effective validation tool for simulating the vehicle perception system and measuring the vehicle response.

Chapter 2

Simulation as a Tool: Sensors and Accident Reconstruction

2.1 Objective

According to the National Highway Traffic Safety Administration (NHTSA), more than 6 million vehicle crashes are reported across the United States each year, leading to around 2 million injuries and 37,000 fatalities [104]. As automated vehicles diffuse through the transportation system, they are expected to improve safety. However, when they do crash, often with conventional vehicles, bicycles, scooters, or pedestrians, they curate a vast amount of data originally intended for sustained sensing of the surrounding environment. This data has the potential to be used for detailed crash scene analysis, provided an event-dependent workflow, including sensor selection, processing, analysis, and visualization is standardized and made available to law enforcement. This study provides an overview of AV-involved crashes and leverages data from carefully selected AV-involved crashes that come from the California Autonomous Vehicle Tester Program. The crashes are simulated with the objective of obtaining sensor information and showing how the data can be analyzed.

More broadly, recent accidents with automated vehicles indicate that crash investigators can greatly benefit from data access and more accurate contributing factor identification (e.g., driver distraction) through sensor data [105, 106, 107, 108]. We also highlight the need for a standardized workflow, including sensor selection, processing, analysis, and visualization which are all event dependent. The end goal is to develop a semi-automated tool for accident reconstruction for law enforcement. In this research, the main objective is to use the safe

system approach to harness data generated by AV sensors, extract features from the data, and incorporate such information into crash investigation analysis. To illustrate the developed methodology, we first analyze the latest information on AV crashes that have occurred in California and select two separate sample AV-involved crashes in California. They are then simulated in the CARLA software [109]. The paper discusses how AV sensors including LIDAR, camera, and radar can be utilized to investigate pre-crash driver states, crash involved movements (e.g., vehicles, pedestrian, bike), and the surrounding environment. This is demonstrated on a sequence of frames from the real-world Kitti dataset [110] by tracking a leading vehicle from the instrumented vehicle using YOLO object detection [111] and a monocular depth estimator called AdaBins [112]. Depth from LIDAR is used as ground truth to calibrate and assess the effect of noise and errors in depth estimation using cameras. A main take-away from this study is that the end outcome (a rear-end collision, for example) is often coupled to the functioning of the automated vehicle subsystems through complex dependencies, even though a casual glance might find the following vehicle to be at fault due to failure to stop.

2.2 Study Design

This study introduces a framework for leveraging newly available AV data collected by multiple sensors to understand AV-involved crashes and extract useful information from such data to incorporate it in the safety analysis. This study takes advantage of data provided in AV-involved crashes in the California tester program. We first analyze the latest information on AV crashes that have occurred in California and then simulate two sample crashes in the CARLA simulation software to demonstrate the advantages and challenges of using AV sensors in crash investigation. Furthermore, the sensitivity of analysis on sensor resolution and data degradation due to weather conditions is studied.

2.2.1 Framework

The framework of the study is provided in Figure 2.1. The safe system framework integrates data generated from different sensors, which monitors the AV, driver, roadway, and environment. Data integration generated by AV sensors provides a full picture of crashes and helps us perform detailed crash investigations. These data parameters can help us understand the

performance of automated driving systems involved in a crash and the control actions taken by the ADS and the driver as well as the situation before and during the crash.

2.2.2 California tester program

The California state Department of Motor Vehicles features a program that requires all autonomous vehicle manufacturers to submit information corresponding to any accidents or crashes that these vehicles were involved in [113]. This information is submitted in a uniform way across manufacturers, allowing us to get a complete understanding of what kind of scenarios surround accidents where AVs are involved, regardless of manufacturer. Additionally, data from these accidents reveals information regarding the weather and lighting conditions that were present around the vehicle at the time of the accident. Data submitted as part of the 2019-2020 Tester program is shown in Table 2.1. There, we can find some key statistics that will shrink the scope of the analysis we perform in this work. We can see that the majority of crashes occur under clear and daylight conditions. We also notice that over 70% of accidents were rear-end collisions. This suggests that this may be an ideal accident condition for study, further motivating our approach in Section 2.5.

After reviewing the descriptive statistics from these 148 crashes from January 2019 to December 2020, we selected two AV-involved crashes. These crashes were chosen based on several criteria. First, as the most dominant AV-involved crash type, a rear-end conflict between AVs and conventional vehicles was selected. Second, the interaction between AVs and pedestrians is one of the key AV challenges, which can be studied in this context. Finally, a typical roadway environment is selected for simulation purposes. Detailed information regarding the selected two collisions is provided in Table 2.2.

2.2.3 Real World Scenarios

The data in this study are deemed to be of high quality, as they come from real-world crashes where basic facts were reported by the CA DMV. Under the proposed plan, the data will have come from deployed sensors and are likely to be valid. The narratives of these two crashes, available in the reports, are as follows:

- **Crash #1 narrative:** "A Cruise autonomous vehicle ("Cruise AV"), operating in autonomous mode, was traveling eastbound on 24th Street between Noe and Sanchez Streets when the

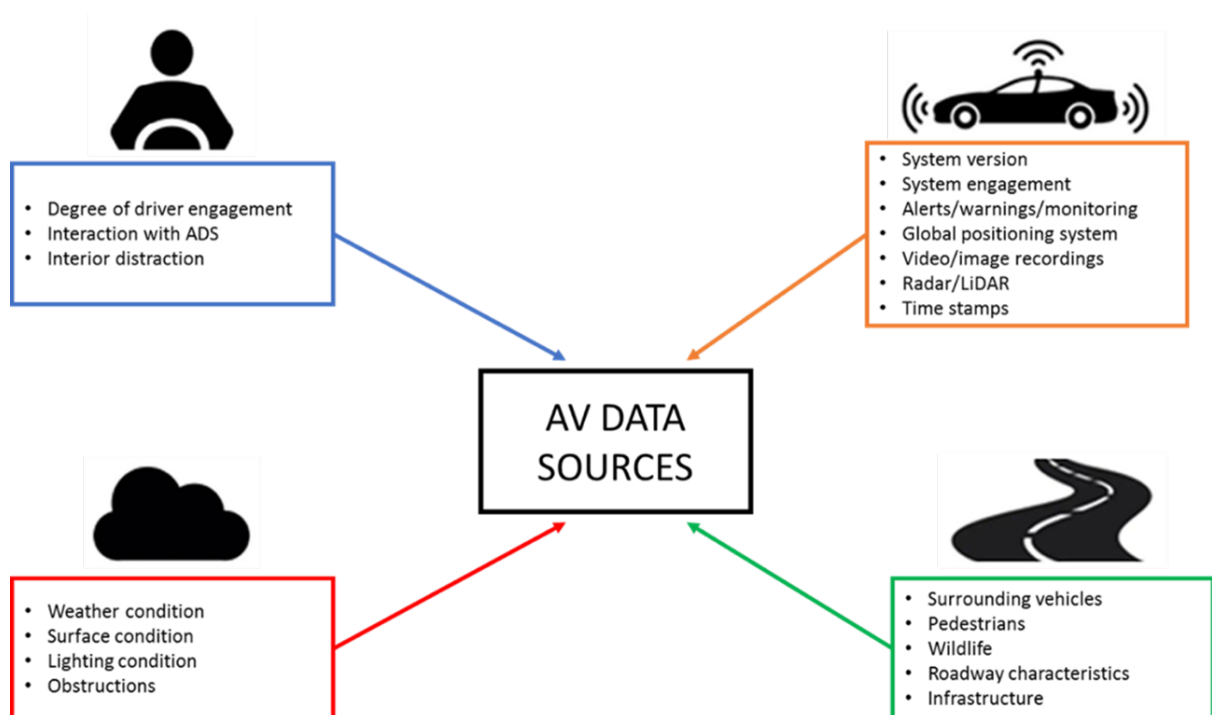


Figure 2.1: Framework to incorporate AV data into crash investigation analysis

Table 2.1: Key Statistics of AV Crashes in California from 2019 to 2020 (N=148)

Variable	Autonomous Mode		Conventional Mode	
	(N=94)		(N=54)	
	Frequency	%	Frequency	%
<i>Vehicle Manufacturer</i>				
Cruise LLC	57	60.6	19	35.2
Waymo LLC	25	26.6	11	20.4
Zoox, Inc.	8	8.5	13	24.1
Other	4	4.3	11	20.4
<i>Manual Disengagement</i>	32	34.0	NA	NA
<i>Weather Condition</i>				
Clear	86	91.5	50	92.6
Cloudy	6	6.4	3	5.6
Raining	1	1.1	1	1.9
Foggy	1	1.1	0	0.0
<i>Lighting Condition</i>				
Daylight	63	67.0	48	88.9
Dusk/Dawn	4	4.3	0	0.0
Dark with Street Lights	26	27.7	6	11.1
Dark without Street Lights	1	1.1	0	0.0
<i>Vehicle Movements (AV, 2nd Vehicle)</i>				
(Stopped, Straight)	21	22.3	12	22.2
(Slowing/Stopping, Straight)	5	5.3	1	1.9
(Straight, Straight)	16	17.0	2	3.7
(Straight, Changing Lanes)	10	10.6	4	7.4
(Left, Straight)	1	1.1	2	3.7
Other	41	43.6	33	61.1
<i>Involving Pedestrian or Bicyclist</i>	7	7.5	4	7.4
<i>Manner of Collision</i>				
Rear-End	66	70.2	23	42.6
Sideswipe	15	16.0	13	24.1
Broadside	9	9.6	3	5.6
Head-On	1	1.1	8	14.8
Other	3	3.2	7	13.0
<i>AV Damage Level</i>				
None	9	9.6	6	11.1
Minor	68	72.3	39	72.2
Moderate	15	16.0	9	16.7
Major	2	2.1	0	0.0
<i>Injury to at least one person</i>	23	24.5	8	14.8

Table 2.2: Information on selected AV-involved crashes

Variable	Description of Crash #1	Description of Crash #2
<i>Date</i>	08/07/2019	11/29/2019
<i>Time</i>	10:23 AM	10:41
<i>Location</i>	24 th between Noe and Sanchez St. San Francisco, CA	Embarcadero St. and Washington St.
<i>AV mode</i>	Autonomous	Autonomous; Disengaged just before the crash
<i>AV movement</i>	Stopped	Moving- Slowing
<i>Other vehicle</i>	Moving-Entering traffic	Moving- Going straight
<i>Collision type</i>	Rear end	Rear end
<i>Weather</i>	Clear	Clear
<i>Lighting</i>	Daylight	Daylight
<i>Road condition</i>	No unusual conditions	No unusual conditions

Cruise AV yielded to a pedestrian crossing the roadway midblock outside of a crosswalk and for clearance to maneuver around a double-parked vehicle immediately ahead of it. While the Cruise AV waited, another vehicle pulling out of a driveway and into the Cruise AV's lane and made contact with the left rear corner of the Cruise AV, damaging the Cruise AV's left rear bumper, fender, and radar. There were no injuries and police were not called."

- **Crash #2 narrative:** "A Cruise autonomous vehicle ("Cruise AV"), operating in autonomous mode, was traveling southeast bound on The Embarcadero at the intersection with Washington Street when the Cruise AV slowed down. The driver of the Cruise AV disengaged from autonomous mode and, shortly thereafter, another vehicle made contact with the rear bumper of the Cruise AV, damaging the Cruise AV's rear fascia assembly and radars. There were no injuries reported at the scene by either party and police were not called. Both of the Cruise AV test operators later mentioned neck and back pain."

The approximate locations of each accident are shown from Google Street View in Figure Fig. 2.2. Using these descriptions as representatives of accidents involving AV's, we intend to show in the remainder of this chapter the value of AV sensors in crash analysis by recreating a combination of these two accidents inside a driving simulation.

2.3 Methodology

2.3.1 Complete Data Pipeline

This study introduces a framework to collect, organize, and analyze data from sensors that are commonly found on modern developmental autonomous vehicles. Under normal operation, these sensors are used to localize the AV and detect relevant objects around the vehicle in order to plan motion. This data is typically of much higher quality than what may be found in the EDR on conventional vehicles. As such, this data can be used for crash reconstruction and analysis that is of much higher fidelity than what can be obtained from basic vehicle information or witness testimony. As shown in Fig. 2.3, we propose a complete data pipeline that takes raw sensor data from the vehicle and automatically processes the data for visualization and analysis.



Figure 2.2: Scenes of crash reconstruction. Crash #1 (Top), Crash #2 (Bottom). Both Images taken from Google Street View in approximate locations described in their respective police narratives

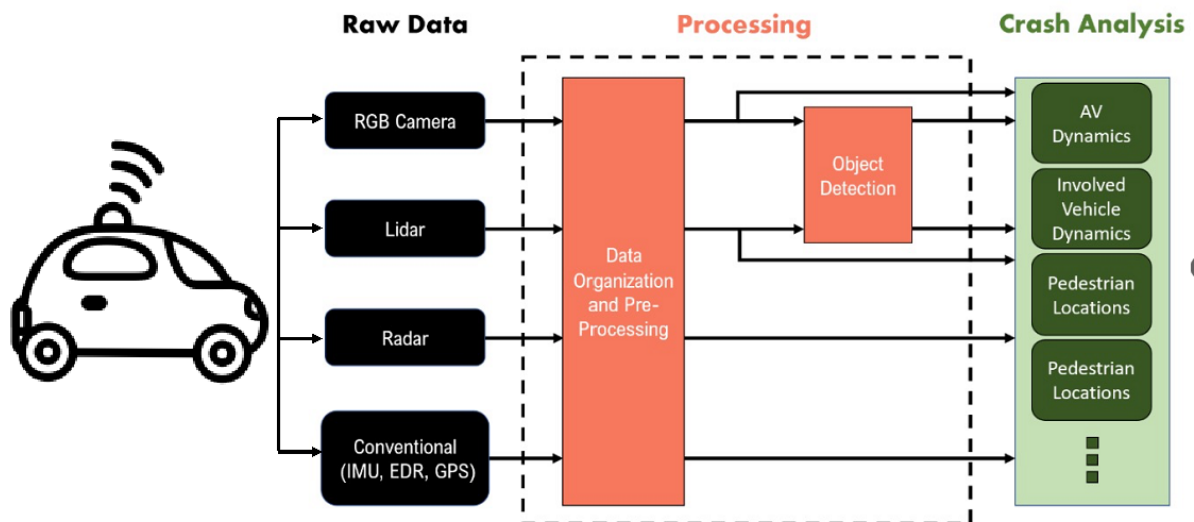


Figure 2.3: Proposed data pipeline for accident analysis. Raw data collection and processing take place in a "black box", effectively removing the need for underlying knowledge of vehicle sensors to analyze driving behavior & crash reconstructions.

2.3.2 Data Processing

Given a stream of high-quality data from the sensors listed in the above section, this data must be sorted and processed to give results that can be visualized and are actionable. For visualization and analysis, we will focus on the three most common data modalities used by CAVs for the purpose of high-fidelity crash reconstruction:

1. A LiDAR 3D point cloud scan of the surrounding environment collected across time.
2. Video from all available cameras leading up to and during the crash.
3. Position and velocity of the AV, other vehicles in the scene, and other relevant factors such as bicyclists or pedestrians. All data are collected across time.

3D Point Clouds

3D point cloud data is collected directly from LiDAR and radar sensors. LiDAR systems collect hundreds of thousands of data points every second - so in order to deal with memory constraints, a pre-processing step is common for down sampling the data. For visualization and analysis, the data from the radar and LiDAR systems will need to be converted to the same Cartesian coordinate system. Since the LiDAR unit is typically located centrally on the vehicle, the inherent coordinate frame of the LiDAR sensor will become the coordinate frame for all other forms of data in the pipeline. Radar data can be transformed into the LiDAR coordinate frame as:

$$X'r = X_r L_{rotation} + L_{translation} \quad (2.1)$$

where X_r is any raw data radar vector, $L_{rotation}$ is the rotation transformation from the radar coordinate frame to the LiDAR coordinate frame, and $L_{translation}$ is the 3D distance between the sensors. Once this transformation is complete, radar data and LiDAR data can be combined into one data structure representing the 3D point cloud of the environment surrounding the crash. Finally, timestamps from both sensors allow the data to be synchronized leading up to the crash.

Video & Object Tracking

Scene video can be captured directly from any camera sensors on-board the AV. This video is natively timestamped for synchronization with data captured from other videos, as well as the point cloud data.

Velocity data for the AV are supplied directly from the IMU's inside the vehicle. While virtually all AV's will have access to GPS data as well for global positioning, "dead reckoning" techniques can also be combined with LiDAR information to track vehicle position [114]. Detecting the position of other actors in the scene such as third-party vehicles, pedestrians, or bicyclists first requires object detection and identification, either through pixel level segmentation or by locating the corners of a bounding box around the object of interest. Given access to a dense 3D point cloud and 2D images, there are many algorithms for effectively identifying and locating relevant objects in the scene. Some state-of-the-art algorithms produce 3D bounding boxes around objects directly from point cloud data [115, 116]. Others use sensor fusion techniques to combine 2D object detection from camera images with 3D point cloud data [117, 118]. An illustration of this entire procedure is outlined in the following section, where a YOLO based object detector followed by a monocular depth estimator called AdaBins are applied on scenes from the real-world Kitti dataset. Regardless of the detection algorithm used, the result will be the 3D position of all relevant actors in the scene. The velocity of any actor can be calculated with the discrete time derivative of position:

$$v_n(x, t) = \frac{x_n - x_{n-1}}{t_n - t_{n-1}} + v_{AV} \quad (2.2)$$

where x_n and t_n represent the current location and time of the object detection, x_{n-1} and t_{n-1} represent the last known location and time of detection, and v_{AV} represents the speed of the vehicle on which the LiDAR is attached, calculated, or read directly from the vehicle IMU. Finally, to make use of all of these sensors, they must be synchronized across time. This is a common problem in mobile robotics, including self-driving vehicles. There are many solutions in literature, and thus is not addressed in this research [119, 120].

2.4 Perception Simulation on KITTI Dataset

The purpose of the procedure outlined in this section is to add clarity to the process of scene reconstruction from RGB images and LIDAR point clouds utilizing the sensor data collected on-board AVs during accidents. We validate this procedure on a high-quality real-world dataset. We also demonstrate how the position of a leading vehicle can be accurately determined using images and point cloud data, in addition to simply highlighting the utility of visualizing raw data and the output of our image processing models directly. This will shed light on how the visualizations performed on simulated data in the remainder of this research could be produced on real data.

The dataset used for real world validation of our data processing procedure was the Kitti dataset [110]. The Kitti dataset has 7,481 training images across three scenes in Karlsruhe, Germany, and was created with the purpose of aiding research in the field of autonomous vehicles. GPS, images from two front-facing cameras, and 360-degree scans from a 64 laser Velodyne LiDAR are recorded for every timestamp. All data instances are also paired with labels that indicate what can be found in the data - namely, six classes of objects: [car, cyclist, pedestrian, tram, truck, van]. These labels also indicate where these objects can be found in the images using 2D bounding boxes. These bounding boxes take the form of a 4-dimensional value for every object: (X_c, Y_c, w, h, c) , where X_c and Y_c represent the center point of the object in pixels while w and h are the width and the height of the object, also measured in pixels. The class of the object is denoted by c .

In instances where simply visualizing the raw sensor data may not provide enough information, additional data processing tools are needed. You Only Look Once (YOLO) is a mature object detection algorithm that passes images through a convolutional neural network (CNN) and attempts to correctly predict the labels of the image, as provided by the Kitti dataset [111]. Specifically, we used the Ultralytics YOLOv5 architecture to perform our proof of concept [121]. The output of the model is in the same format as the labels in the dataset, with an additional confidence value: $(x_c, y_c, w, h, c, conf)$. This *conf* value represents the likelihood that the object at that location is really there and accurately detected, and falls in the range of [0,1]. The user sets a confidence threshold, below which all detections are ignored. The confidence threshold of our model was set at 0.4.

There are kinds of tracking demonstrated here, using the KITTI dataset as an example of raw data input.

- **Case 1: Tracking with LIDAR and camera:** As shown in Fig. 2.4, in this workflow, object detection with YOLO is accompanied by projecting the LIDAR point cloud onto the image plane. The procedure for projecting LiDAR points onto the image are provided in [110]. Once this projection is done, LiDAR points and camera images are in the same coordinate system, where each LiDAR point contains a 2D pixel location (x_L, y_L) , and a depth measurement (d_L) in meters. The LIDAR depth measurements (d_L) are now interpolated to provide $(d_L(x_c, y_c))$, which serves as an accurate estimate of the object distance for tracking. It may be noted that LiDAR and camera images are not fused within the YOLO model. Instead, the YOLO model provides (x_c, y_c) while LiDAR provides the depth estimates. This procedure can be repeated for any sensor modality that produces point cloud data. While the KITTI dataset only contains point cloud data in the form of LiDAR, many vehicles are equipped with radar sensors. Since radar is also in the form of 3D point cloud data, the procedure is identical to LiDAR.
- **Case 2: Tracking with only camera:** In the absence of a LIDAR, we need an alternate way of estimating distance to objects of interest. For this application, we used a monocular depth estimation model that provides a depth estimation for every pixel in an image using CNNs. For our purposes, we chose to use the monocular depth estimator AdaBins [112]. This model was chosen due to the fact that a pre-trained model for the Kitti dataset is readily available, making integration into a workflow seamless, as well as the fact that it currently ranks highly on the Kitti depth estimation leader board [122]. Both the YOLO model and the AdaBins model were trained using the entire Kitti dataset. At this stage, finding the distance to the object of interest is as simple as finding the pixels that corresponds to the lead vehicle and recording the distance from the model's depth prediction.

2.4.1 Location Estimating Demo

Our accident reconstruction procedure assumes access to GPS/IMU data that can be used to track the location of the vehicle that contains the advanced sensors (LiDAR, radar, cameras). Given that this is a rather trivial procedure of reading the data from the GPS/IMU, here we focus on demonstrating the ability to track the position of an un-instrumented legacy vehicle

with respect to the ego vehicle. For this purpose, two hundred consecutive data instances over 20 seconds were taken from the Kitti dataset, where a "lead" (legacy) vehicle is in front of the automated vehicle. The legacy vehicle is initially close to the automated vehicle, pulls away slightly, and then becomes close again by the 200th frame. Being able to track this behaviour is important for scene reconstruction as will be demonstrated in the next section. First, the camera frame at each time instance is given to the 2D object detection model. Since the model will produce detections for every object class in the image, the user must specify which vehicle they are interested in. Because this scenario is a "vehicle following" scenario, we are interested in the movements of the vehicle closest to the centre of the image. Once the vehicle is identified, we can track, i.e., estimate the relative distance to that vehicle, either using both camera and LiDAR (Case 1) or with just camera data (Case 2). The raw data, as well as the ability of our two image processing models, are summarized in Fig. 2.5. The output of our object detection model is shown in Fig. 2.5a. The LiDAR points in the dataset are projected over the image in Fig. 2.5b. The output of our camera-based depth estimation model is shown in Fig. 2.5c. The depths for both LiDAR and the estimation model are in the range of 0-80m, as shown. Note that the distances for the depth model are only estimates based on the inner workings of the CNN behind the model, while the LiDAR values are direct measurements and thus can be assumed to be ground truth.

The results of tracking this vehicle are shown in Fig. 2.6. The 1st, 100th, and 200th frames are shown (at 0, 10, and 20 seconds respectively), and the distance calculations across time are plotted for both LiDAR (ground truth) and monocular depth estimation. Figure 2.6 also provides a plot of the error between the ground truth LiDAR and the image-based depth estimation model. This demonstrates that even with the use of data from only a single monocular camera, a setup that is expected to be deployed on all automated vehicles, reliable estimates of relative distances and relative speeds can be obtained by training and using the correct deep neural nets. This also illustrates that all sensor modalities are not equally accurate, and the uncertainty incorporated by sensor and process (depth estimation model) noise may be relevant when trying to reconstruct the causes of an accident.

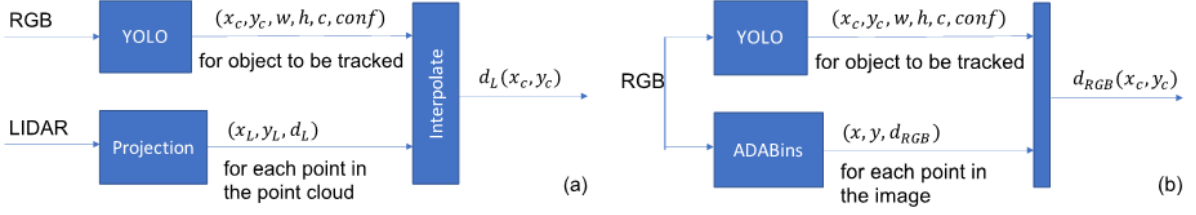


Figure 2.4: Data post-processing procedure. (a) Data and processing pipeline using (a) both LiDAR and RGB images, and (b) only RGB images with AdaBins providing depth estimates

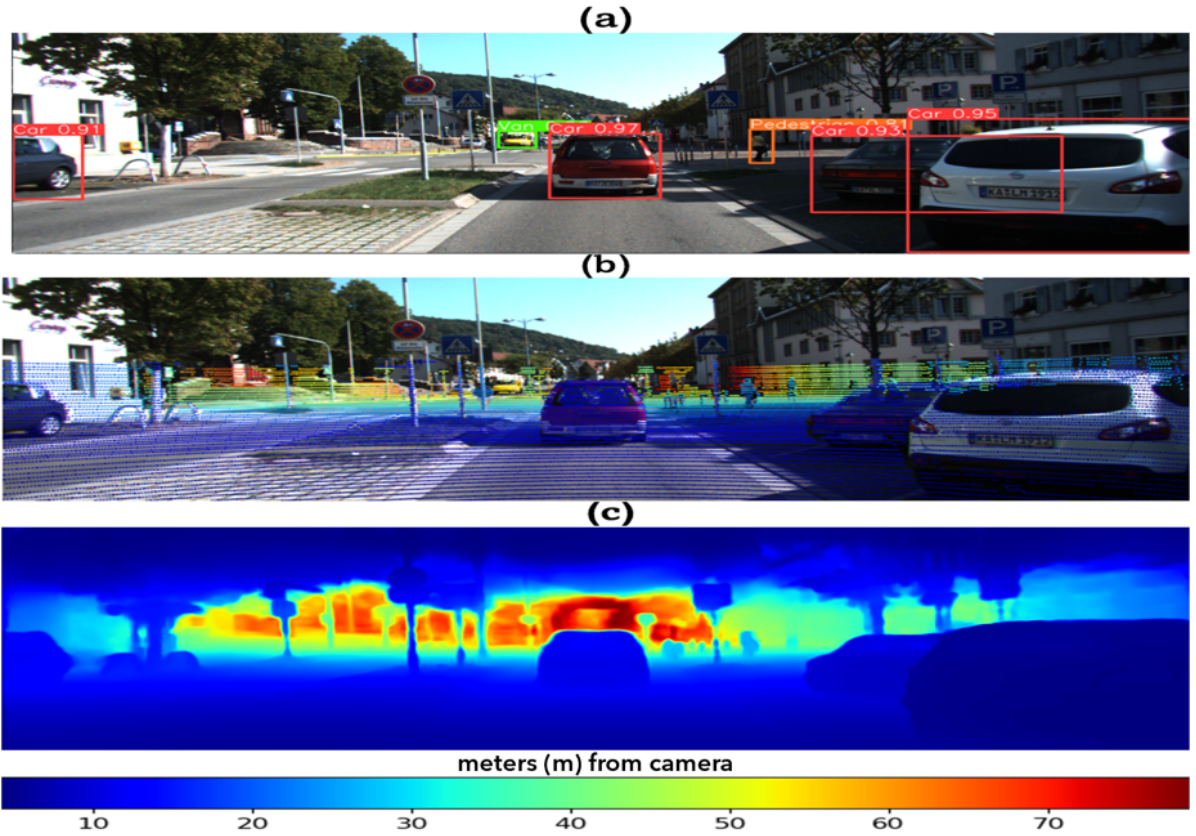


Figure 2.5: KITTI visualization of YOLO, LiDAR, and *Adabins* depth detection]. A representative frame from the Kitti dataset and subsequent processing (a) Output of our YOLO object detector (b) The LiDAR points in the dataset projected over the image. (c) The output of our monocular camera-based depth estimation model (AdaBins).

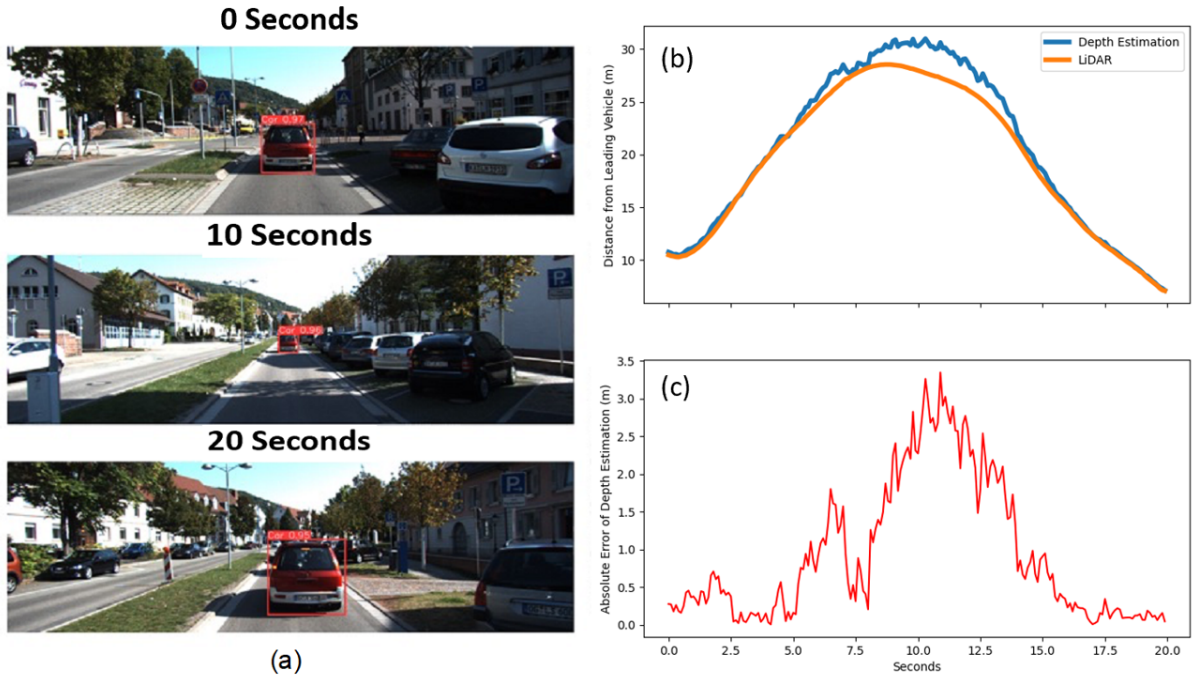


Figure 2.6: KITTI object tracking demonstration. To demonstrate the results of tracking a leading vehicle relative distance estimation on a driving sequence extracted from the Kitti dataset are shown (a) Sequence of frames – 1st, 100th and 200th frame, measured in seconds. (b) Distance to the leading car from the instrumented car across the tracking period measured from both the depth estimation network (blue) and LiDAR (orange). (c) Error estimate of visual depth estimator over time with LIDAR based depth as ground truth.

2.5 CARLA Simulator

In the previous section, real world data was used to demonstrate the feasibility of our data pipeline toward effective visualizations. However, real-world data from accidents involving AVs are sparse, so in lieu of access to complete sensor data from actual AV involved crashes, simulation can be used to produce this data instead. To simulate AV-involved crashes, the CARLA simulation platform was chosen in this work [109]. CARLA is an open-source software enabling users to configure, simulate, and generate data from several AV sensors including LiDAR, cameras, radars, and GPS. CARLA is built on top of the Unreal Engine 4 graphical development kit that is specifically designed for AV simulation. Most features present in CARLA are intended for realistic and useful simulation and training of an autonomous vehicle’s sensors and control algorithms. However, for the scope of this study, we present CARLA simply as a tool for collecting data in crashes involving a vehicle that is kitted with sensors such as LiDAR, radar, or cameras. In this scope, CARLA is used as a tool for crash recreation, data validation, or counterfactual scenarios quantification. Regarding CARLA’s dynamics, vehicles can be controlled by implementing custom vehicle software that controls the vehicle through data from simulated sensors and its measured dynamics. In lieu of implementing a complex control system that recognizes objects of interest such as vehicles, lane markings, etc. using computer vision algorithms, CARLA uses a robust lane keeping, car following and emergency stopping algorithm instead. In this scheme, waypoints are pre-programmed to indicate correct positioning for all driving lanes on the road. By supplying a destination waypoint, a simple proportional–integral–derivative (PID) controller was tuned that allows the CARLA vehicle to reach its destinations by controlling throttle, steering, and braking to minimize the error between its current position and the closet target waypoint on its set path. All vehicle dynamics in this study are controlled in this manner for the sake of simplicity, reliability, and repeatability.

2.5.1 Simulation Scenarios

The CARLA simulations shown are simplified scenarios based on common elements found while analyzing the California Tester Program police reports.

- **CARLA Simulation #1:** A cruise autonomous vehicle is driving on an urban street with a green jeep approximately 10 meters behind it, with both vehicles accelerating to

approximately 26 mph. A jaywalking pedestrian walks in front of the AV, triggering the AV to brake suddenly and avoid striking the pedestrian. The green jeep fails to stop, rear ending the AV. Both vehicles avoid contact with the pedestrian.

- **CARLA Simulation #2:** A cruise autonomous vehicle is making a left turn with a green arrow in a four-way intersection. A conventional vehicle coming from the right with respect to the AV passes through the red light at approximately 45 mph. A mid-speed collision with the rear-right side of the AV is made in the middle of the intersection.

Data collected from these simulations are oriented to be precise, making simulation of real-world crashes such as this one a potential avenue for the public to obtain a clear picture of an event described by witnesses. In these scenarios, all vehicle and pedestrian positions are known with exact precision, all sensor data streams are synchronized perfectly, and the resulting simulation is perfectly repeatable. Table 2.3 illustrates that simple visualizations of the simulated sensor data or more complex object tracking outlined in the remainder of the chapter can provide the information listed in the police report at a minimum.

2.5.2 Simulated Sensor Data

Our framework is intended to primarily accept camera, radar, and LiDAR data as input. These types of sensors are widely used in AV applications, and have varying benefits and weaknesses in different types of perception tasks [123]. All these sensors and their corresponding data types can be replicated inside CARLA with high level similarity with their real-world counterparts. For all data collection, our simulated AV was kitted with 4 sensors: 2 cameras, 1 LiDAR, and 1 rear facing radar, as well as conventional sensors that supply data regarding internal vehicle dynamics. Descriptions are provided below, with relevant parameters for each listed in Table 2.4.

- **LiDAR:** Real world LiDAR data is generated by multiple lasers that rotate 360° around the sensor while calculating time of flight on the reflection of each laser. The result is a 3-D point cloud that can supplement other sensors such as cameras to provide the exact locations of objects in space. Of the sensors mentioned, LiDAR data is considered of the highest quality. The number of data points returned in a single scan of the sensor is often very rich, and the point cloud contains a relatively small amount of noise. In simulation, a laser can be represented by a 3D graphics tool called a "ray-trace", which can return the location

Table 2.3: CARLA Simulation Accident information

Variable	CARLA Simulation #1	CARLA Simulation #2
Time	12:00 PM	7:00 PM
AV operation mode	Autonomous	Autonomous
AV Movement	Stopped	Turning Left
Legacy vehicle movement	Moving- 0-27 mph	Moving- 40-45 mph
Collision type	Rear end	Rear-side
Weather condition	Clear	Cloudy
Lighting	Daylight	Low Light- Evening
Roadway condition	No unusual conditions	No unusual conditions

Table 2.4: Overview of the data input for Simulation #1 and Simulation #2.

Sensor	Data Type	Data size	Range	Rate
Camera	Image	480x720 pixels	$\approx 10\text{m}$	60 Hz
Radar	3D point cloud	≈ 300 points	20m	50 Hz
LiDAR	3D point cloud	$\approx 30,000$ points	50m	50 Hz
GPS + IMU	Position, Velocity	1 point	N/A	> 60 Hz

of any object encountered along a straight line in the simulated world. Thus, simulated LiDAR is produced with essentially the same theory of operation as real LiDAR but without consideration of the physics of dispersion, diffraction, and reflection due to varying angles of incidence as well as material properties that affects the intensity of the reflected ray.

- **Radar:** Radars are attached to many modern vehicles and allow for varying levels of automation. Radar also uses time of flight measurements of lower frequency waves to obtain the location of obstacles and are considered a cheaper solution than LiDAR but produce data that can be used for similar purposes. Simulated radar involves much of the same tools as simulated LiDAR. Ray tracing is used inside of CARLA to obtain object locations, and a series of functions are applied to the data to make it more realistic in comparison with physical radars, such as applying noise, transforming the data to a polar coordinate system, and inferring velocity data available on any radar sensors.
- **Cameras:** Just as in real cameras, these simulated cameras provide RGB (red, green, blue) images from inside the simulation. Cameras are relatively inexpensive to implement in AV applications but are computationally expensive to apply inside CARLA simulations. However, there are many tools to apply post-processing affects to the data that can allow us to achieve specific situational reconstructions in images.
- **Conventional Sensors (IMU, EDR, GPS):** The conventional sensors provide data regarding the global position and vehicle dynamics of the vehicle. GPS will provide a known Cartesian location for the vehicle in space. Vehicle speed and velocity can either be read directly from the IMU (Internal Measurement Unit) of the vehicle, or by differentiating the incoming data with respect to time.

2.5.3 Data Visualization

Many insights into crash behavior can be made by simply visualizing the output of this data pipeline with no further analysis. As shown in the data collected from CARLA simulation #1, we can see that multiple camera angles, as well as what amounts to a 360 degrees representation from LiDAR and radar, are available for analysis with a minimal amount of technical post-processing. Fig. 2.7 shows a simple visualization of the data that can be obtained directly from an AV following a crash. This data is synchronized across all modalities, and in this case, shows all data obtained from the vehicle in the moments prior to rapid deceleration of the AV,

as well as the moment after collision. Visualization such as what is shown in Fig. 2.7 can be done easily and rapidly. Making this data available to investigators following an accident could allow a rapid and accurate understanding of what took place, verifying or alleviating the need for witness testimony. In crash analysis, it is important to understand the totality of events that culminated with the crash. Vehicles with autonomous capabilities do not typically have cameras with full 360-degree vision around the vehicle. However, the multiple sensor modalities on board the vehicle can be visually combined to show the complete picture. Figure 2.8 shows a complete visualization of CARLA Scenario #2. By combining multiple sensor modalities, we can reach a larger number of conclusions about the crash. For instance, we can see that the AV had the right of way with the green arrow when turning left. We also conclude that the LiDAR was able to locate the incoming vehicle almost a second before the rear camera saw the vehicle. All of this data can be displayed side-by-side with the trajectories of the vehicles for a complete picture of the crash, while any one sensor would be insufficient.

2.5.4 Data analysis for detailed crash reports

As shown in the previous section, a simple visualization of the data that is available following an accident involving an AV can provide valuable information that may or may not be available directly from investigation or witness testimony. However, in the case of AVs, it is extremely likely that the investigation may need to understand how the AV and humans in an accident responded before, during, and after an accident at a deeper level. When did the vehicles react? Did the AV brake too suddenly? Did the human drivers have enough time to react to the AV? Such questions can be answered by applying simple data analysis and readily available tools to the collected data. Figure 2.9 shows positional and velocity information that would typically not be available in EDR data, which only consists of vehicle kinematics pertaining to the vehicle kitted with sensors and does not contain any trajectories of surrounding vehicles, pedestrians, or bicyclists, and environmental conditions such as rain or snow. In Fig. 2.9, the six seconds prior to collision are plotted. distance from AV to pedestrian (top left), AV velocity (top right), distance from AV to colliding conventional vehicle (bottom left), and velocity of the conventional vehicle (bottom right). The green and red vertical lines indicate the time instants at which the frames displayed in Fig. 2.7a and Fig. 2.7b were collected respectively. It is obvious in Fig. 2.9 that the conventional vehicle following from behind never applied the brakes, as its velocity never decreased. Simple numerical analysis can be valuable as well, such

as observing that the AV slowed from approximately 12 m/s to a complete stop in just over a second. More complex analysis on AV sensor data can certainly be done, but this illustrates the value in applying simple techniques to AV sensor data analysis. For crash investigators, we will need to develop procedures and protocols that will facilitate obtaining and analyzing such data. The results clearly show a more complete picture of pre-crash scene and factors contributing to the crash.

2.6 Limitations

As mentioned previously, simulations and synthetic data are typically considered simplified or idealized versions of their real-life counterparts. In the construction of this simulation, there are no doubt many relevant details present in the actual crash that are both present and not present in the police report. While there are tools and techniques for accounting for an increasing number of variables as well as making sensor data more realistic, it is important to note that simulation tools such as CARLA used in this research as a supplement to real data collected on AVs and not as a replacement. Specifically, if any conclusions are to be drawn about the performance of object recognition algorithms or vehicle dynamics using simulation, careful consideration needs to be given to proper domain transfer of the synthetic data to faithfully resemble real data, as well as a complete understanding of the vehicle kinematics and the dynamics model deployed in the simulation. Also, for the demonstration purposes of our simulations, object recognition was done using tools built-in with CARLA as well as manual annotation. As such, error associated with any of the object detection algorithms implemented within this framework will be found within the visualizations that are not demonstrated here.

2.7 Summary

In this work, we outline a systematic methodology for recreating traffic accidents involving autonomous vehicles. This work directly fills a much needed literature gap in two respects. First, it allows data rich AV platforms to use their pre-existing data collection methods to provide accident investigators a richer understanding of what took place, which is currently rather sparse. Second, it outlines an approach to AV safety that uses not only the known facts of the accident itself, but also the sensor data to provide context for *how* the accident may have

occurred, especially in situations where the AV may be at fault. In Chapter 3, we build on this approach of not only looking at safety metrics related to vehicle response, but also providing the context necessary to diagnose this behavior.

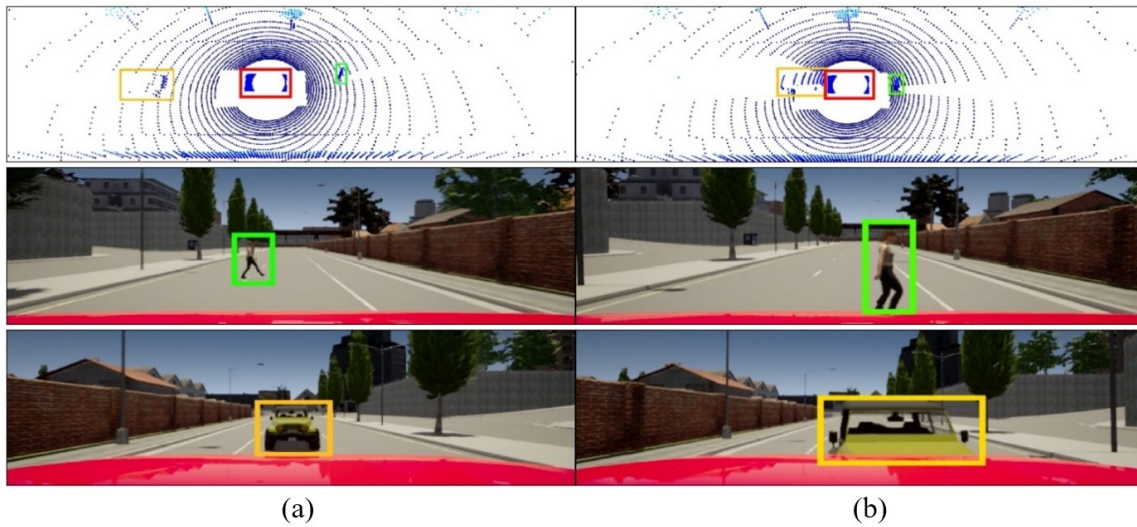


Figure 2.7: CARLA Simulation #1. Visualization of CARLA sensors in CARLA Simulation #1. LiDAR and Radar (Top), front-facing camera (middle), rear-facing camera (bottom) are shown. Frames are displayed a) 1.14 seconds before collision and b) at vehicle impact. Pedestrian (green), following vehicle (orange), and AV (red) are color coded for all sensor modalities.

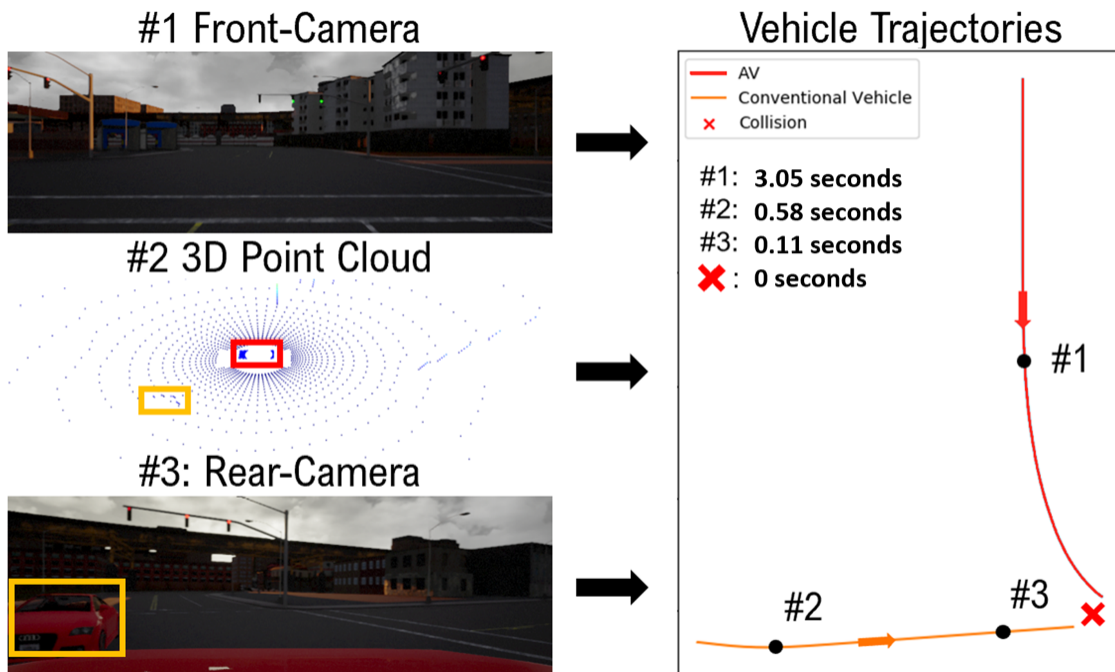


Figure 2.8: CARLA Simulation #2. Times shown are in seconds before collision. Following vehicle (orange), and AV (red) are color coded for all sensor modalities. Visual data collected (left) can be used to supplement scene dynamics data (right) for a clear understanding of the crash, even when any one sensor is not sufficient.

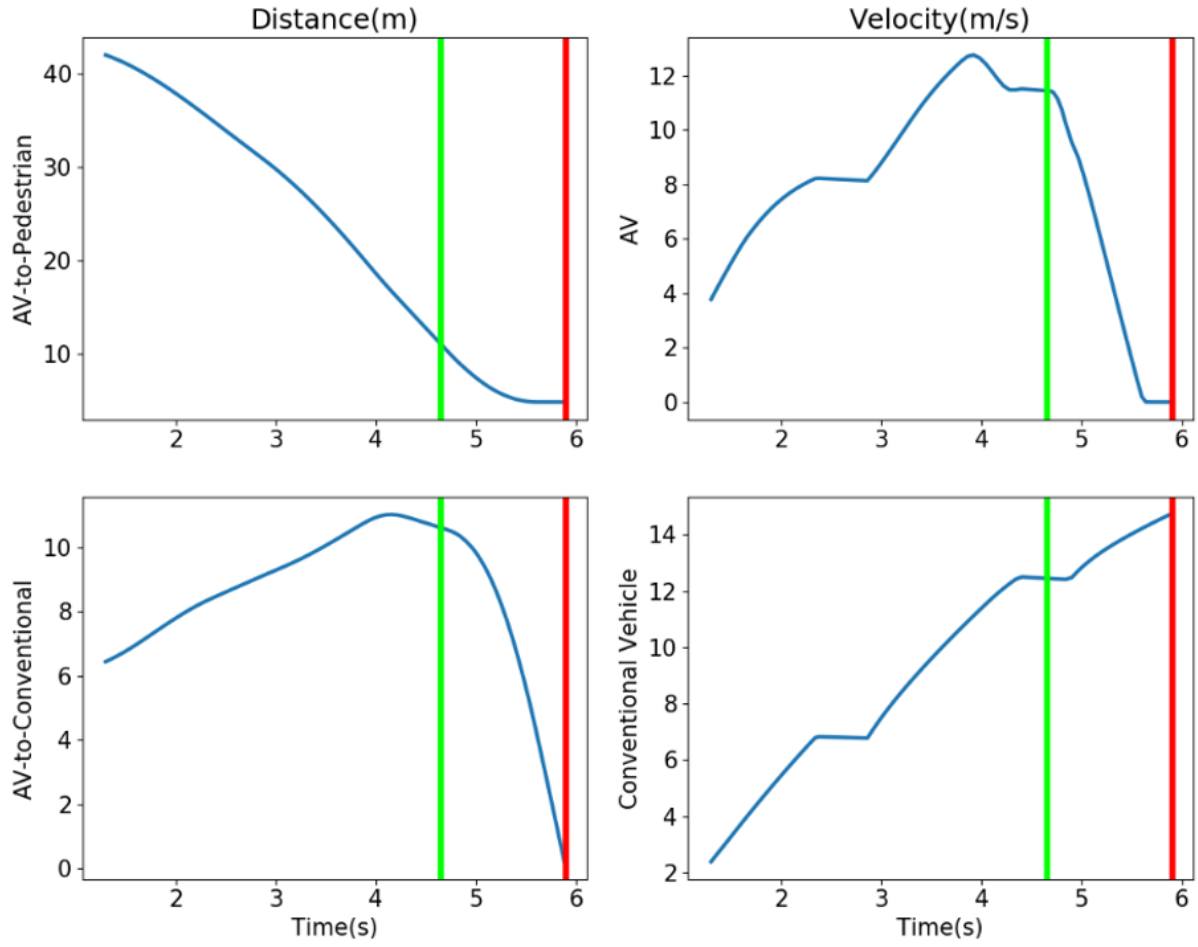


Figure 2.9: Qualitative Analysis is performed for CARLA Simulation #1. Positional and velocity information of vehicles prior to collision. The green and red vertical lines indicate the time instants at which the frames displayed in Fig. 2.7a and Fig. 2.7b were collected respectively.

Chapter 3

AV Behavior Diagnosis, Prediction, and "Edge-Case" Discovery

3.1 Objective

The question that we attempt to address in this chapter is how to systematically assess the safety of connected and automated vehicles (CAV) before driving them on public roads. The history of human-driven vehicle safety reveals a tendency to leap first and analyze later. Based on other concurrent research thrusts, it seems that there are mixed definitions for safe testing presently established, and the lack of an efficient safe method is both placing human lives at risk and threatening to stymie the development of this emerging technology. The goals of this chapter are to introduce and outline a hybrid testing protocol that can enable future creation of a set of certification standard recommendations. We focus on level 3 automation in this chapter as vehicles with this capability already exist in cars like Teslas and Cadillacs. These cars have level 3 capabilities but drivers are expected to monitor the systems to ensure proper operations. Given that several recent accidents have demonstrated that vision systems are prone to rare but catastrophic failures, the perception subsystem and the downstream effect of its inaccuracies on measurable driving metrics is an important diagnostic step in the evolution of AVs.

With the increasing penetration of learning-based perception modules in real-world systems, it is imperative to better understand the inner workings of deep-learning models designed for computer-vision, and make them safe for widespread use. Traditionally, learning-enabled sensing/perception and control modules of AVs are designed and trained in isolation without

system-level safety objectives. In this chapter, we emphasize end-to-end safety of the overall driving system both during training and deployment.

In this work, we demonstrate a safe systems testing protocol using a complete code-to-road (CTR) ecosystem, comprising of a real vehicle-in-the-loop simulator. We introduce the unique capabilities of the wheel speed synchronized steerable dynamometer facility developed at the University of Tennessee that enables full-sized vehicles to be immersed into a virtual replica of on-road scenarios, offering safe and repeatable technology evaluation. This unique platform lets us test the full operational envelope of the AVs, with the digital twin enabling the creation and evaluation of a wide range of situations with easily manageable safety consequences. The vehicle used for the experiments reported in this paper is a Level 3 SUV built and instrumented with a dedicated computer with ROS and CAN communication. The dynamometer can be programmed to mimic the road load. In this paper, we report the use of a real vehicle with hardware-in-the-loop processing, control and actuation, which lets us realistically test the effect of environmental and controller perturbations on self driving functionalities.

3.2 Methodology

In this section, our vehicle-in-the-loop (*VIL*), software-in-the-loop (*SIL*), and model-in-the-loop (*MIL*) simulation strategies are outlined. All three strategies provide a variety of challenges, as well as varying degrees of benefit. *VIL* simulation provides the most realistic vehicle response by far, in the sense that real vehicle dynamics are measured, and the control mechanism works through the actual CAN bus on the physical vehicle. However, even with our custom dynamometer setup, *VIL* is the most expensive and time consuming type of simulation; requiring a physical vehicle, a large experimental footprint, and testing that must be safety conscience and performed in real-time. *SIL* replaces the actual vehicle with a vehicle plant model. This is much easier to implement due to the lack of any hardware component, but is primarily done in this work to demonstrate the large gap that exists between *VIL* and *SIL* simulations in terms of dynamic response to identical stimuli. Finally *MIL* simulation as we define it is simply a controlled test of the perception model along a fixed trajectory. *MIL* is the easiest type of simulation to implement, and can even be performed faster-than-real-time (FTRT). Because *MIL* operates on a fixed trajectory, it is not possible to infer about vehicle response directly. However, *VIL* and *MIL* can be combined to get the benefits of both types of

simulation, while mitigating against the challenges of each. Demonstrating this combination on the real SAE L2 controller *Openpilot* constitutes the primary contribution of this work.

3.2.1 Virtual World

The vehicle simulations described in this work leverage the CARLA open-source simulation platform [124]. CARLA is a virtual simulation platform that provides tools for vehicle simulation, particularly autonomous systems. Within the scope of this work, we leverage CARLA as a testing environment in all three types of simulation. In *VIL* simulation, camera images from CARLA are generated and fed to the vehicle controller, the kinematic response is measured directly from the vehicle on the dynamometer, and this kinematic response is used to update the position of the vehicle in CARLA, completely bypassing CARLA’s built in physics model with the dynamic response from the real vehicle. In *SIL* simulation, the controller takes in camera images, the vehicle physics model in CARLA takes throttle, braking, and steering commands from the controller, and the position of the vehicle is determined by the physics model within CARLA. Finally in *MIL* simulation, a kinematic trajectory for the vehicle across time is pre-defined. The vehicle in CARLA is simply moved along this trajectory as if on rails, the camera images from CARLA are sent to the vehicle perception system, and the output of the perception system is captured. These constitute the three types of simulations used in our experiments, and are outlined visually in Fig. 3.1.

3.2.2 Perception Model & Controller

A modified version of the open-source controller *Openpilot* v8.13 by Comma.AI was used in this research [125]. *Openpilot* is an SAE level 2 controller featuring compatibility with many recent vehicle makes and models. While *Openpilot* is openly sourced, it uses a proprietary vision model dubbed *Supercombo*. This deep learning-based vision model provides two key outputs used for control. First, the model takes in time-sequential camera images and produces a target lane trajectory for the car to follow. Model predictive control (MPC) and proportional–integral–derivative control (PID) use this trajectory to produce the steering, throttle, and braking commands that are sent to the vehicle in order to follow and maintain this trajectory. While the model produces a full trajectory across multiple time-steps for every image, we focus on the trajectory prediction of 500ms in the future, denoted as $\hat{\mathbf{l}}_{t+500ms}$. This value, in Cartesian world coordinates, can be considered the model’s estimate of where the

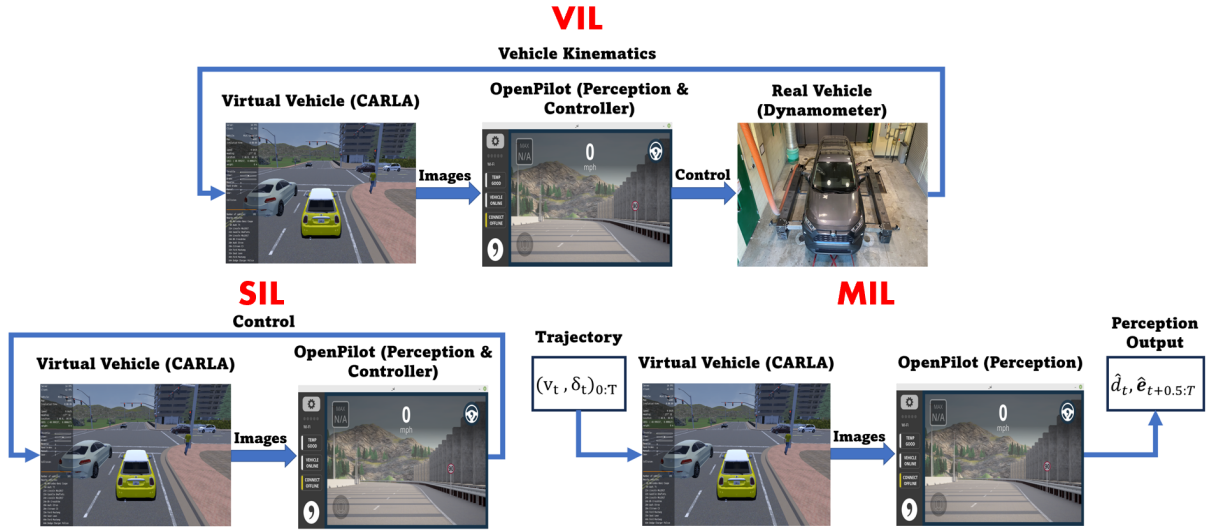


Figure 3.1: Data Flow of each simulation modality. An overview of the Vehicle-in-the-loop(VIL), Software-in-the-loop(SIL) and Model-in-the-loop(MIL) simulations used in this work. In VIL, the feedback to the simulator is vehicle kinematics measured from the CAN bus of the vehicle directly. In SIL, the controller feeds control commands to the simulator directly, and the physics model within CARLA determines the dynamic response. In MIL, the CARLA vehicle follows a predetermined trajectory, feeding images to the *Openpilot* perception model.

vehicle will be in 500ms, while $\mathbf{l}_t$ is the current position of the vehicle. Second, the model produces a distance estimate between itself and the lead vehicle. We note that *Openpilot* produces this estimate as a combination of relative speed, absolute speed, and distance estimates of the lead vehicle from *Supercombo*. For clarity, we only report the final estimate for stopping distance that is given to the longitudinal MPC controller in *Openpilot*, and denote this value as $\hat{d}_t$, where the actual distance from the lead vehicle and the ego vehicle at any given time t is d_t .

3.2.3 Vehicle-in-the-loop Simulation

The vehicle used for *VIL* simulations was a stock 2019 Toyota RAV4, shown in Fig. 3.2. This vehicle was chosen primarily due to *Openpilot*'s ability to integrate easily with the ADAS system in the vehicle, with no limitations on minimum cruise control speed or complete stop functionality. Communication between the vehicle and *Openpilot* is performed through two vehicle CAN buses: one that interfaces with the on-board diagnostics (OBD II) port found on all modern vehicles in the United States, and another that interfaces with the Advanced driver-assistance systems (ADAS) CAN bus with a connector just above the rear view mirror. The *Openpilot* controller typically works by using a windshield mounted camera combined with the vehicle's built-in radar to perceive the road. In our setup, there is no physical camera facing the road. Instead, *Openpilot* and CARLA are run simultaneously on a laptop, with images from CARLA being fed directly into *Openpilot*, and control information being sent to the car from the computer via a USB-to-CAN connector provided by Comma.AI. Additionally, all radar and sensor-based localization functions within *Openpilot* were disabled, making the controller's perception system purely vision-based.

All of the *VIL* simulations were done on a single-axle dynamometer equipped with an eddy-current brake for applying road load. *VIL* simulation is made possible on the chassis dynamometer by a unique prototype machine, provided by *VIL Solutions, Inc.*, that is designed to deliver Steering-enabled Synchronized wheel-Speed (SeSwS) * and shown in Fig. 3.2. This machine enables the ability to speed match all four wheels by linking the front and rear axles, and thus avoiding the typical challenges caused by vehicle safety system faults detected from wheel speed mismatches. This technology also allows for the actuation of the vehicle's steering

*The intellectual property associated with SeSwS is patent pending, owned exclusively by *VIL Solutions Inc.* Hardware has been provided for research purposes to the University of Tennessee, Knoxville in partnership with *VIL Solutions Inc.*

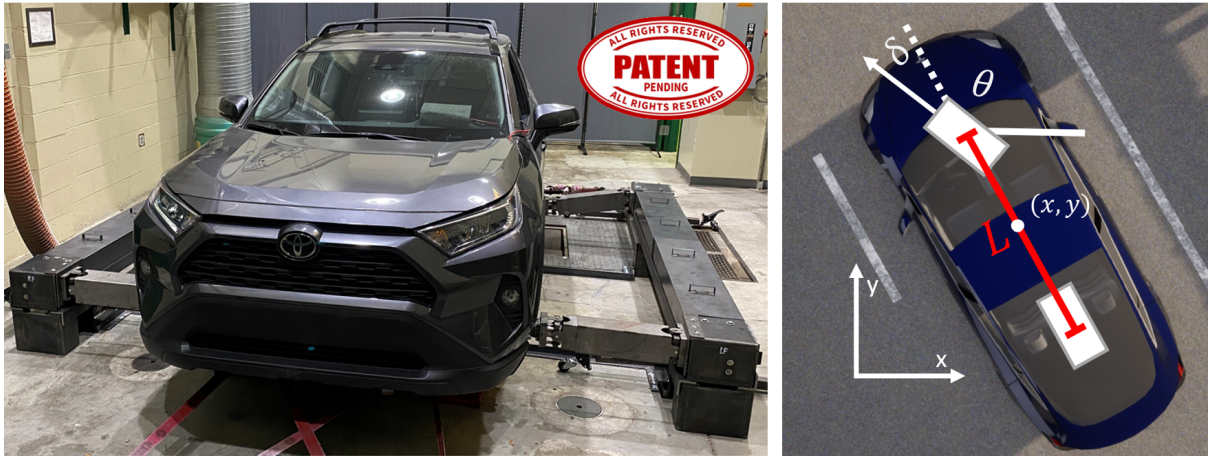


Figure 3.2: Complete *VIL* Simulation setup. (Left) The 2019 Toyota RAV4 is shown on the chassis dynamometer. The device that allows for steering and synchronized wheel speed is attached. (Right) The kinematic model used to update the position of the vehicle in CARLA using the dynamic response of the vehicle is shown. The wheelbase L , the heading θ , and the steering angle δ are illustrated, the heading and the position of the vehicle (x, y, θ) are defined in terms of arbitrary world coordinates defined by the CARLA simulator.

system through an articulated connection to the front axle. The combination of this equipment and the dynamometer allow for safe *VIL* testing in a controlled environment, where a virtual environment can provide stimulus to the vehicle.

For the *VIL* simulations, the dynamic response of the vehicle can be combined with the kinematic bicycle model to produce the next position of the vehicle within the simulation. The kinematic bicycle model is shown visually in Fig. 3.2. The four wheel speeds are measured from the CAN bus, averaged together and fed into a one-dimensional Kalman filter to produce an estimate of the total vehicle speed $\mathbf{v}_t$. The equivalent steering angle of the “bicycle wheel” is computed as $\delta_w = \delta_s / sr$, where the steering ratio is $sr = 14.3$ for the 2019 Toyota RAV4, and the orientation of the steering wheel δ_s is taken off of the CAN bus directly. The time-step Δt is fixed at precisely 10 ms. The wheel base of the vehicle is $L = 2.69\text{m}$ for the 2019 Toyota RAV4. The position and orientation of the ego vehicle in world coordinates are denoted by $\mathbf{e}_t = (x_t, y_t, \theta_t)$, and the slip angle with reference to the center of the vehicle is denoted as β . The full kinematic update at every time-step is produced with:

$$x_{t+1} = (v \cos \theta) \Delta t + x_t \quad (3.1)$$

$$y_{t+1} = (v \sin \theta) \Delta t + y_t \quad (3.2)$$

$$\theta_{t+1} = \left(\frac{v \tan \delta_w \cos \beta}{L} \right) \Delta t + \theta_t \quad (3.3)$$

3.2.4 Software-in-the-loop Simulation

In *SIL* simulation, the parameters of the vehicle within CARLA are kept mostly in line with what has been tested in CARLA via *Openpilot*’s unofficial simulation tool in order to maintain the highest degree of compatibility possible between the *Openpilot* controls and the CARLA physics control models. Namely, the wheelbase used in simulation is taken from the Tesla Model Y at 2.89m. The weight and steering ratio of the vehicle were changed to further bring the model in line with the Toyota RAV4 at 1530kg and 14.3, respectively. Because the physics model within CARLA determines the dynamics of the vehicle in the *SIL* simulations, the velocity $\mathbf{v}_t$, as well as the position and orientation of the vehicle $\mathbf{e}_t$, are determined by simply querying that information from the CARLA simulator itself at every time instance. Images are passed from CARLA to the *Openpilot* controller in the same manner as our *VIL* setup.

Additional steps are also taken in software to spoof certain CAN messages that *Openpilot* needs to read from the vehicle to operate.

3.2.5 Model-in-the-loop Simulation

In *MIL* simulation, we follow a pre-defined trajectory $\mathbf{e}_{0:T}^{(i)}$ for the vehicle. Here, i is the sample identifier from which the pre-defined trajectory is defined, such that $1 \leq i \leq n$ for n samples. This notation is ignored unless explicitly required. We follow this predefined trajectory by updating the vehicle's position and orientation across time with a series of speeds $\mathbf{v}_{0:T}$ and wheel angles $\delta_{0:T}$, and updating the full position and orientation using Equations (3.1) to (3.3). In this case, there is no vehicle controller in operation. Instead, images from CARLA along the fixed path are passed to *Openpilot's Supercombo* in a standalone manner. This form of simulation allows for "trajectory replay", where experiments performed under *VIL* or *SIL* conditions can be repeated in an *MIL* manner by simply replaying the kinematic response across time, and feeding the resulting images to the perception system. When interpreting the dynamic response of a controlled *VIL* or *SIL* experiment, it can be difficult to determine how the perception model contributed to the response at any time t because the trajectory up to that point $\mathbf{e}_{0:t}$ has a causal and cascading effect on the images that produce the perception output $\hat{\mathbf{e}}_t, \hat{d}_t$. Using *MIL* simulation allows for an analysis of the perception output using a variety of weather and lighting conditions, but controlling for the effect of the vehicle's trajectory.

3.3 Experimental Setup

In this work, there are three axes of experimentation. The first is simulation type, as described in the previous section. While keeping all other factors identical, we can compare the dynamic responses of *SIL* and *VIL* simulations to measure the increase in realism when going from *SIL* simulation to *VIL* simulation. The individual components of this difference could be difficult to model separately, with contributions from the response time of vehicle hardware components to the physics of the vehicle itself. Comparing *SIL* simulations to *VIL* simulations allows us to begin to gauge that difference in its totality. The second axis is driving type for the ego vehicle. The two types of driving characterized by our experiments are simple car following, and the ability to slow down and stop for a stopped vehicle in front with sufficient spacing between the ego vehicle and the stopped vehicle. These two types of driving capture the essential functions

needed in an SAE L2 vehicle. The third and primary axis of experimentation are the factors related to perception: lighting, weather, and lead vehicle type. By modifying elements of the environment that affect perception, we can begin to understand how certain driving conditions might impact driving performance and mitigate against them. Visualizations for each driving type, as well as the primary weather and lighting effects, are shown in Fig. 3.3. The sample sizes for each experimental combination using the labels defined in this section are shown in Table 3.1, where "A" and "S" indicate "ambulance lead vehicle" and "sedan lead vehicle" respectively. The metrics used to measure performance and a quantitative analysis of the results are given in this section. Additionally, the predictive metrics from the perception model are described.

3.3.1 Driving Types

Two types of driving were conducted, with the goal of encapsulating the requirements of an SAE L2 vehicle. A bird's eye view of the **Stopping** and car **Following** experiments under clear weather conditions are shown in Fig. 3.3. Due to concerns with preserving experimental hardware, both experiments are conducted at relatively low speeds (< 40 mph). The two types of driving under test are:

- **Stopping:** This is a test of the obstacle detection system in the SAE L2 controller, as well as the ability to bring the car to a smooth stop with a reasonable amount of distance between the ego vehicle and the leading vehicle. Accounting for lane curvature, a stopped vehicle is placed 125 meters in front of the ego vehicle. The ego vehicle accelerates from 0 mph to a set speed of 30 mph. Data collection begins when *Openpilot* control is activated. The run ends when the vehicle either stops for the lead vehicle, or collides with it.
- **Following:** This test combines lead vehicle detection with the ability to provide a consistent throttle and brake signal for car following, which is the core expectation of SAE L2 vehicles. *Openpilot* calculates a target following distance based on the relative speed of the ego vehicle and lead vehicle. For the lead vehicle's set speed of 30 mph, this following distance is roughly 22 meters. The ego vehicle accelerates from 0 mph to a set speed of 35 mph. Data collection begins when the ego vehicle comes within 15 meters of the lead vehicle. At that point, the lead vehicle maintains a constant velocity of 30 mph while keeping its current lane by using CARLA's built in autopilot feature, which is a simple PID controller maintaining minimal

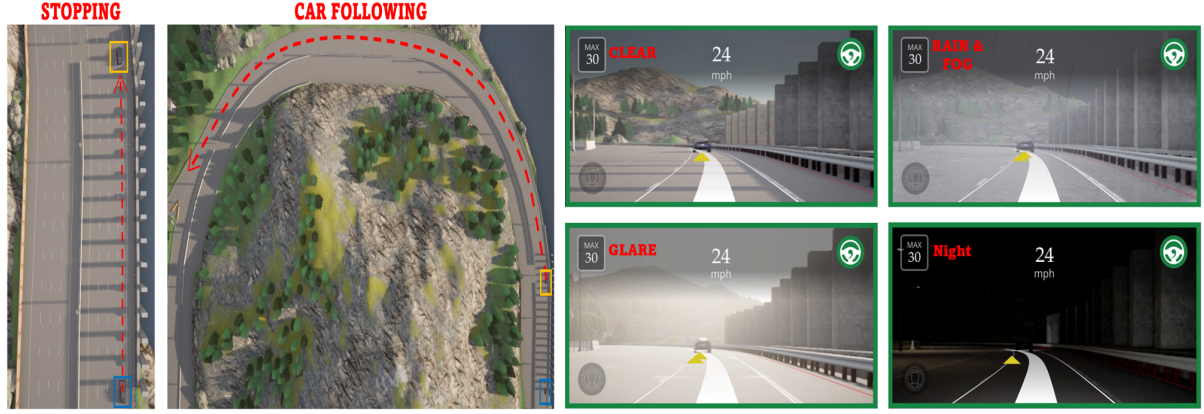


Figure 3.3: Experimental setup, weather, & lighting. Driving types and the primary weather and lighting effects under test are shown. For the two types of driving on the right, the ego vehicle is shown in a blue box, the lead vehicle is shown in an orange box, and the driving path under test is shown as a dashed red line. On the right, an image was captured 8 seconds into the **Stopping** test for clear weather, rain, sun glare, and night type driving with headlights.

Table 3.1: Number of experimental runs n conducted (*VIL*, *SIL*, and *MIL*)

Driving Type	Baseline (<i>VIL</i>)		Night		Rain Fog		Sun Glare		Sunset Clear		Sunset Noon
	S	A	S	A	S	A	S	A	S	A	A
Following	3	3	3	3	3	3	3	3	3	3	0
Stopping	5	5	5	5	5	5	5	5	5	5	5

distance between the vehicle and predefined waypoints in the lane. Accounting for lane curvature, this test lasts for 695m from the starting point of the ego vehicle.

3.3.2 Weather, Lighting, & Lead Vehicle

Weather and lighting effects are applied as a method for applying additional challenges and variance to the *Openpilot*'s perception model. These effects manifest in the lane detection and thus the trajectory prediction, as well as the obstacle detection. Note that all the conditions listed here solely affect perception, and none of these adverse weather conditions effect the dynamic response of the vehicle under test in *SIL* or *VIL* simulations.

- **Baseline** (*VIL*): For both types of driving, baseline experiments were performed, which were simply a single human driver performing each experiment on the *VIL* hardware setup as realistically as possible with no intervention from *Openpilot* or the RAV4's integrated ADAS system. These baseline experiments were performed under clear weather conditions, with the human driver gauging the vehicle response by monitoring the video feed directly from CARLA.
- **SunsetClear** (*VIL, SIL, MIL*): Clear weather conditions in the evening.
- **RainFog** (*VIL, SIL, MIL*): The hardest possible rain setting within CARLA, mid-afternoon. A small amount of fog is also present (15% of maximum in CARLA API).
- **SunGlare** (*VIL, SIL, MIL*): The sun is positioned to be directly overhead of the lead vehicle for the maximum amount of time for the experiment. This corresponds to a 180 degree azimuth angle in CARLA coordinates for the **Stopping** experiment, a 270 degree azimuth angle for the **Following** experiments, and a 10 degree altitude angle for both experiments.
- **Night** (*VIL, SIL, MIL*): Clear weather conditions at night, with headlights on for both vehicles.
- **NoonClear** (*VIL, SIL, MIL*): Clear weather conditions at midday. This condition was done to add more variance to the results for the ambulance, and was solely done for the **Stopping** experiment with the ambulance present.
- **FogLevel** (*MIL Stopping* Only): The **RainFog** setting is repeated, varying fog percentage from [5, 100] in CARLA to determine how various fog levels in rainy conditions affect

perception performance. Experiment consisted of $n = 95$ runs, all conducted with the Ambulance lead vehicle.

- **SunAngle (MIL Stopping Only):** The sun angle is varied for $[0, 90]$ degree altitude and $[0, 360]$ degree azimuth angles in CARLA to determine how time of day and sun angle affect perception performance. Experiment consisted of $n = 333$ runs, all conducted with the black sedan lead vehicle.

The other factor affecting the perception system of *Openpilot* is the choice of the lead vehicle for both types of driving. Headlights were active in night driving experiments for both vehicles. Experiments were conducted with:

- **Black Sedan:** A black Tesla model S. Distance measurement d_t is offset 2.39m to establish the position of the rear of the vehicle, as opposed to the center of the vehicle.
- **Ambulance:** A Ford E-Series Ambulance with full ambulance reflective markings. Distance measurement d_t is offset 3.17m to establish the position of the rear of the vehicle, as opposed to the center of the vehicle.

To fully control against the effect of vehicle trajectory, *MIL* experiments were conducted for all conditions in Table 3.1 using the vehicle trajectories $\mathbf{e}_{0:T}^{(1:n)}$ from the **VIL : Night : Black Sedan** experiments, and thus kept the same number of samples n . The choice of **Night : Black Sedan** conditions was made arbitrarily. Note that these trajectories were used regardless of the choice of lead vehicle. In the experiments specific to *MIL* simulation, only the first sample trajectory $\mathbf{e}_{0:T}^{(1)}$ from the experiment **VIL : Night : Black Sedan** was used.

3.3.3 Safety, Performance, & Predictive Metrics

In both driving types, we focus on safety when measuring performance. The vehicle dynamics should be smooth, the vehicle should stay as close to the center of its own lane as possible, and the vehicle should obviously avoid close calls and collisions while keeping a safe following distance. We also define predictive metrics. These are outputs of our perception model that can be useful to diagnose or predict the vehicle response. The 2D Cartesian position and speed of the ego vehicle are defined as the vector e_p and the scalar e_s respectively. The location and speed of the lead vehicle are defined as l_p and l_s respectively. Notationally, n is reserved for the number of runs for each experiment, shown in Table 3.1. We define n_e as the number of

samples present in each measured trajectory. Our sampling rate for all experiments was 20Hz. Thus, $n_e = 20T$ for all runs, while the length of the run in seconds T varied. For convenience, the distance between the two vehicles at any given time instance t is denoted $d_t = \|e_{pt} - l_{pt}\|$. The metrics used in our analysis are as follows:

- **Average Centerline Distance (Following):** Centerline distance is the smallest distance between the line representing the middle of the current lane, and the midpoint of the ego vehicle. In CARLA, only discrete measurements of the lane position are available. Centerline distance is measured as the distance from the ego vehicle (i.e. e_p) to the line formed by the two nearest lane position measurements denoted by c_1 and c_2 . Mean centerline distance is defined as:

$$\text{cd}_{\text{mean}} = \frac{1}{n_e} \sum_t \left| \frac{c_{1t} - c_{2t} \times e_{pt} - c_{1t}}{\|c_{1t} - c_{2t}\|^2} \right| \quad (3.4)$$

Note that Eq. (3.4) can straightforwardly become a predictive metric as well, when the vehicle position e_{pt} is replaced with the model's predicted trajectory for the vehicle $\hat{e}_{pt}$. In the following section when this predictive form is used we will refer to Eq. (3.4) as $\hat{\text{cd}}_{\text{mean}}$.

- **Minimum time-to-collision (TTC) (Stopping):** The time-to-collision measures how long it would take to impact the lead vehicle if the ego vehicle continues at its current speed indefinitely. In this way, TTC can be measured at every time instance. A value very close to 0 would indicate that a collision occurred. The minimum time to collision is simply the smallest time to collision that occurred in the run, defined as:

$$\text{ttc}_{\text{min}} = \min_t \frac{e_{st}}{d_t} \quad (3.5)$$

- **Mean Following Distance (Following):** *Openpilot* computes a set following distance, given the speed of the ego vehicle and the speed of the lead vehicle. For the **Following** experiments, this is approximately 22 meters. While this may be higher than what could be considered a

safe threshold, we can measure performance relative to how well the vehicle maintained this target distance. Mean following distance is simply defined as:

$$fd_{\text{mean}} = \frac{1}{n_e} \sum_t d_t \quad (3.6)$$

- **Mean Detection Ratio (Stopping & Following):** This is a predictive metric that measures the performance of the lead-vehicle detection feature of the perception model. This is simply a measure of what degree the model is *overestimating* the distance between the ego vehicle and the lead vehicle, where a ratio above 1 indicates an overestimate, and a ratio below 1 indicates an underestimate. Where $\hat{d}_t$ is the distance estimate from the model at any time t , the mean detection ratio is defined as:

$$dr_{\text{mean}} = \frac{1}{n_e} \sum_t \frac{\hat{d}_t}{d_t} \quad (3.7)$$

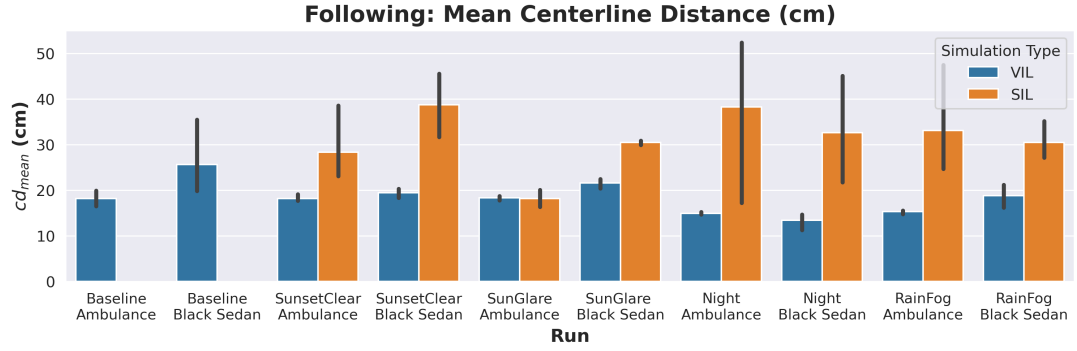
3.4 Results

When analyzing the results of our two experiments through the lens of the metrics outlined in the previous section, we have three main objectives. First, we show that the performance metrics used can be applied to qualitatively illustrate performance. We can see the potential pitfalls of *SIL* simulation against *VIL* simulation, and we can also infer at a high level about the implications of varying environmental and situational conditions. Second, we show that it is possible in the *Openpilot* controller to *diagnose* the vehicle’s dynamic response. Given that the only variables within each type of experiment are factors that solely effect perception, we show that it is possible to correlate the output of the perception system with safety critical metrics found in the eventual vehicle response. Finally, we take this a step further by showing that it is possible to *predict* the *VIL* vehicle response with *Openpilot*’s perception output in the *MIL* experiments. This combination allows us to correlate the perception data from our *MIL* experiments against the highly accurate vehicle response of our *VIL* experiments, effectively combining the best aspects of both modalities.

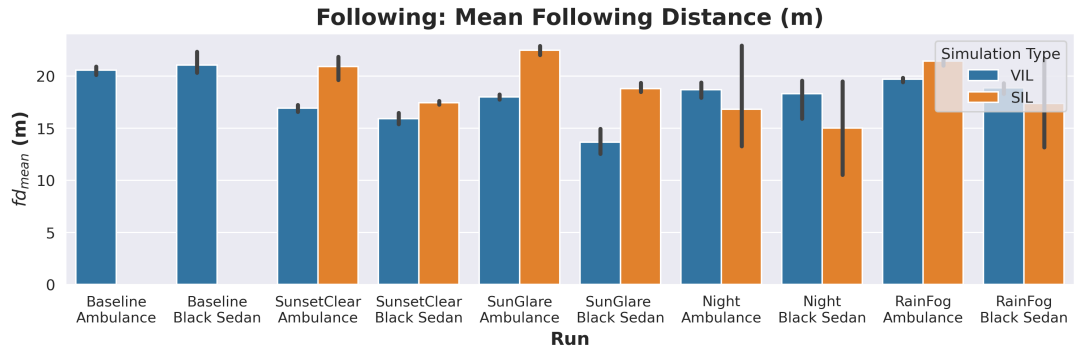
3.4.1 Performance Results

The metrics collected are quantitatively shown in Fig. 3.4. The first comparison to make is the general performance of *Openpilot* against the human driving baseline. In terms of maintaining a low distance from the center of the lane, *Openpilot* running the *VIL* loop generally meets or outperforms the human driver in the following experiments, both in terms of mean distance, as well as the variance of lane deviation across runs. As might be expected, *Openpilot* is able to control the vehicle with fine tuned consistency and precision in this metric that is measured in centimeters. In terms of the **Stopping** experiments, we can also observe that *Openpilot* is significantly less conservative in terms of the minimum time to collision than the baseline human driver. The second observation we may point out is that the vehicle in *SIL* simulation generally had a much more aggressive response to control stimulus than the *VIL* simulation. This had a mixed bag of effects on performance. While **Following**, this resulted in more aggressive steering, which resulted in a higher cd_{mean} than in the *VIL* experiments, with more deviation in response as well. More importantly, we see a major difference in the results of the stopping experiments. We notice that the *VIL* experiments were generally much more dangerous than the *SIL* experiments. While many of the *VIL* experiments resulted in collision or near collision, very little can be parsed from the *SIL* experiments in terms of safety. While the reasoning that explains why some environmental conditions outperformed others is not readily apparent in Fig. 3.4, the hardware and vehicle implementation on the dynamometer clearly allowed us to produce an experiment that represented a safety critical edge case that would not have been apparent if our testing was limited to *SIL* simulation.

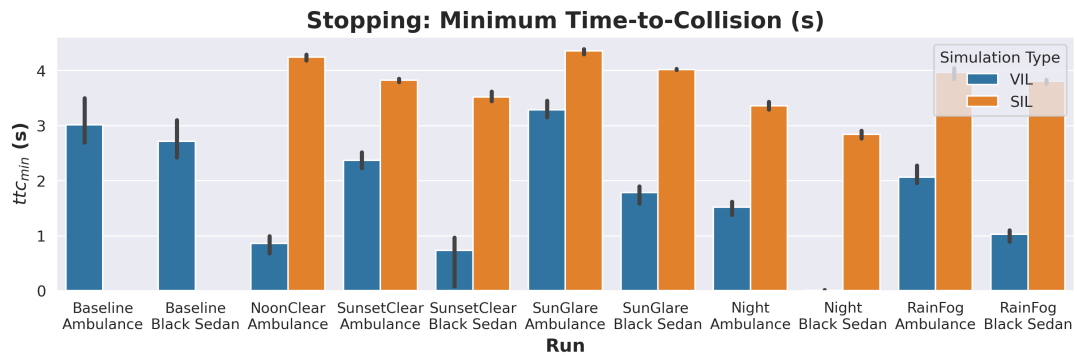
We can also begin to examine the effects of weather and lighting at a high level in the case of the *VIL* simulations. In various instances, the results do not follow intuition. **SunGlare** produced more centerline deviation and a larger deviation from the target following distance of 22 meters in the car **Following** experiments. However, **SunGlare** easily produced the safest ttc_{min} for both the sedan and the ambulance in the **Stopping** experiments, while environmental deterioration like **RainFog** outperformed **SunsetClear** for the black sedan. Indeed, these results both highlight the utility of a *VIL* simulation platform instead of just a *SIL* platform, as well as demonstrate the need to investigate the causal effects of the perception system further in order to attribute explanations for these safety critical edge cases.



(a) cd_{mean}



(b) fd_{mean}



(c) ttc_{min}

Figure 3.4: Performance metrics. Bar length represents the mean score for all runs, with number of runs shown in Table 3.1. Vertical black bars indicate the minimum and maximum measured scores for each metric.

3.4.2 Response Diagnosis

The results from Fig. 3.4 indicate the various weather, lighting, and choice of lead vehicle imposed in our experiments produced significant deviation in terms of the safety critical response of the vehicle. This is particularly true in the **Stopping** experiments, which will draw the most of the focus of our analysis. We can begin to understand this response by working backwards from the control commands to the output of the perception system. *Openpilot* controls the longitudinal motion with a single acceleration command given to the vehicle. This acceleration command for each **Stopping** experiment is shown for $10m < d_t < 50m$ in Fig. 3.5a. We can see that while the requested acceleration stays mostly consistent in the beginning of each run for all experiments, deviation sets in during deceleration. This deviation correlates as expected with the ttc_{min} metric in Fig. 3.4. For example, we can clearly see in Fig. 3.5a that **SunGlare Ambulance** had the most aggressive deceleration, while **Night Black Sedan** had the least aggressive deceleration and resulted in collisions.

Given that all other factors are controlled for in our experiments, we can assume that the causal element affecting the deceleration response time in Fig. 3.5a is the scene variation in the images being given to *Supercombo*. We note that this perception system is a deep learning vision model. With this model, an intuitive understanding of why the model performs the way it does may not be possible without access to the model architecture itself. For instance, an understanding of why the model performs safer under sun glare conditions than under normal midday conditions is not available to us. However, we can still use the output of the model to draw direct correlations between how the perception model failed, and the resulting dynamic response. We use the detection ratio $\hat{d}_t/d_t$ as the baseline performance predictor in the **Stopping** experiments under the assumption that the differing responses in the **Stopping** experiments are the result of the perception model producing different estimates $\hat{d}_t$ of the lead vehicle distance d_t . In Fig. 3.5b, the detection ratio is plotted against lead vehicle distance d_t . We can see that, like the control acceleration signal of Fig. 3.5a, it is a good predictor of ttc_{min} . Generally speaking, the more the model *overestimates* the distance of the lead vehicle, the lower the ttc_{min} of the run. To that end, it is clear from Fig. 3.5b that the choice of lead vehicle had a large impact on this metric, as the position of the black sedan was more consistently overestimated in a comparison against the ambulance.

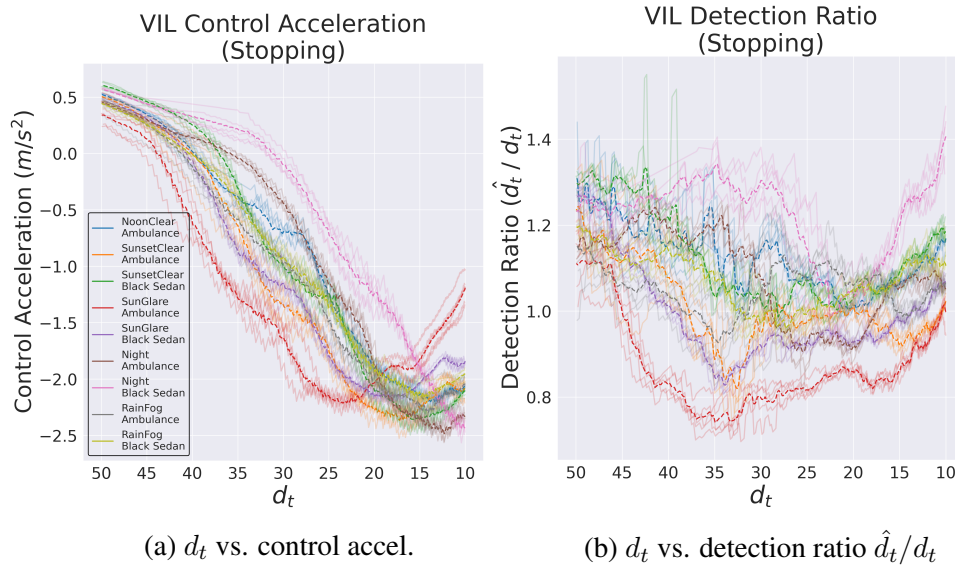


Figure 3.5: Raw data for the stopping experiments. Fig. 3.5a shows the control acceleration signal supplied to the car during the deceleration curve plotted against lead distance d_t . Fig. 3.5b plots the detection ratio for the **Stopping** experiments in the same distance range. Actual trajectory samples are shown as transparent, while the dotted lines indicate the mean value for each experiment.

By simply taking the mean detection ratio of the curves in Fig. 3.5b, we are able to clearly demonstrate the linear relationship between ttc_{min} and dr_{mean} . Similarly, dr_{mean} can be used to predict fd_{mean} in the **Following** experiments, and perhaps most intuitively, the lane deviation of the model $\hat{cd}_{mean}$ predicts the resulting lane deviation of the vehicle cd_{mean} . The linear relationship between these predictive and response metrics for all experiments are shown in Fig. 3.6. The causal relationship is visually apparent in all three cases. In Fig. 3.6c, we can see that beyond a certain value of dr_{mean} , a collision becomes imminent. Qualitatively speaking, this effect is the only source of non-linearity in the correlation shown in Fig. 3.6c. Pearson Correlation Coefficients (PCC) were produced for all three predictive metrics, shown in the *VIL* column of Table 3.2. Correlation is extremely high for all three ($|r| > 0.8$), and all correlations are statistically significant ($p\text{-value} < 0.05$).

3.4.3 *MIL* Simulation for Response Prediction

Given a controller and perception system in which the outputs of these subsystems are transparent, as is the case with *Openpilot*, it becomes possible to gain a rich understanding of the relationship between the perception system and the dynamic response of the vehicle. However, the utility of this understanding is quite limited when searching for new edge-case scenarios if we use *VIL* simulation alone, given that the performance and safety metrics, which are the metrics we are mostly concerned with in edge-case detection, are already available to use if we commit the resources required for *VIL* simulation. In the previous section, we diagnosed vehicle behavior using the output of the perception model. Here, we hypothesize that these same predictive metrics from the perception model, when run under certain environmental conditions will still predict the *VIL* response under those same conditions, even if the trajectory of the vehicle that produced the predictive metrics becomes arbitrary. In other words, we attempt to correlate the predictive metrics of Table 3.2 for *MIL* simulation against the response metrics of Table 3.2 for *VIL* simulation. There is no one-to-one relationship between *MIL* simulation samples and *VIL* response samples, so we are restricted to correlate these metrics using their respective sample means. For any predictive or response metric M , we define the

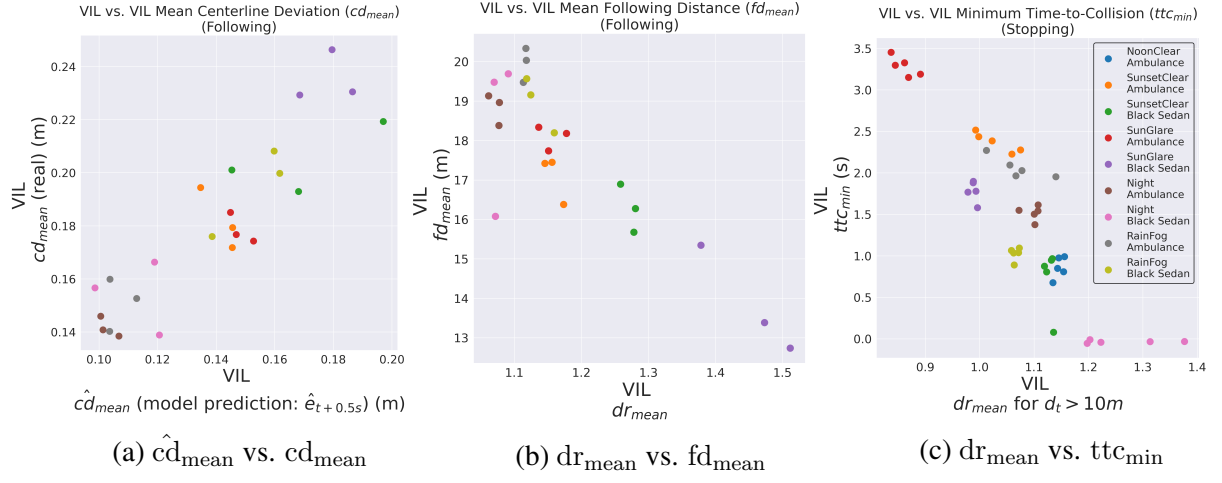


Figure 3.6: *VIL* predictive metrics are plotted against *VIL* response metrics.

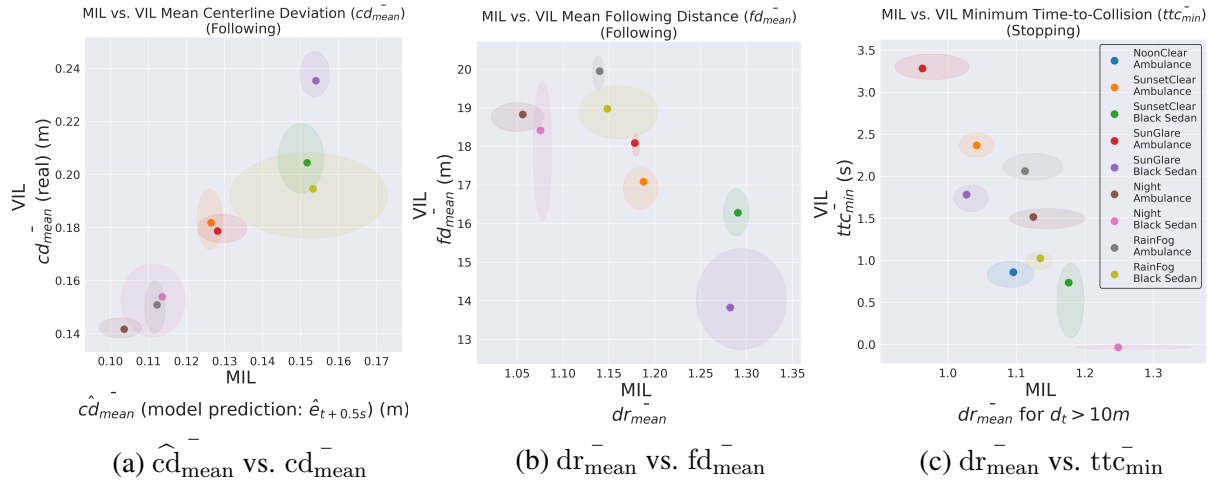


Figure 3.7: *MIL* predictive metrics are plotted against *VIL* response metrics. Transparent ovals indicate an estimate of the possible distribution for each experiment, from Eq. (3.9).

sample mean and the sample min-max pairs respectively as:

$$\bar{M} = \frac{1}{n} \sum_i M^{(i)} \quad (3.8)$$

$$(M^{min}, M^{max}) = (\min_i M^{(i)}, \max_i M^{(i)}) \quad (3.9)$$

For all experiments, the predictive sample means for the *MIL* simulations are plotted against the response sample means of the *VIL* simulations in Fig. 3.7. The sample min-max for these metrics from Eq. (3.9) define the colored ovals around each plotted sample mean. This is done to provide qualitative context around the potential sampling distribution of both the predictive and response metrics. We can see from Table 3.2 that the predictive strength from *MIL* simulation to *VIL* simulation is still very strong ($|r| > 0.8$) for all of the predictive metrics. dr_{mean} remains a strong predictor of both ttc_{min} and fd_{mean} , while the predicted model trajectory $\hat{cd}_{mean}$ is still able to predict the real lane deviation cd_{mean} , even while the model is being run on a fixed trajectory. While there is a large reduction in sample size by being forced to limit ourselves to regressing against the sample means, the results remain statistically significant ($p\text{-value} < 0.05$). These results indicate that even when controlling for the cascading impact of the controller and the dynamic response of the vehicle, we can still leverage the perception model in *MIL* simulation to predict what the response would be under those same environmental conditions in *VIL* simulation.

This combination of using *MIL* simulation to predict the dynamic response of *VIL* simulation is a powerful tool for discovering safety critical edge cases. No longer limited by the hardware and real-time nature of *VIL* simulation, we can perform *MIL* simulations on any machine capable of running CARLA to gain a deeper and higher resolution understanding of how the vehicle perception system can influence performance. To illustrate this, we once again repeated the **Stopping** experiments for *MIL* simulation, but this time with finely tuned changes in environmental conditions and using much larger sample sizes. The results of these experiments are shown in Fig. 3.8a and Fig. 3.8b. We can see that a richer understanding is visually apparent. In the case of fog level in Fig. 3.8a, we can see that under a certain level, fog seems to have little impact on performance. After fog level increases beyond $\approx 40\%$, dr_{mean} degrades linearly. Comparing with Fig. 3.7c, we can estimate that *VIL* crashes would become imminent ($ttc_{min} = 0$) around a fog level of $> 60\%$. In Fig. 3.8b, we observe how the various

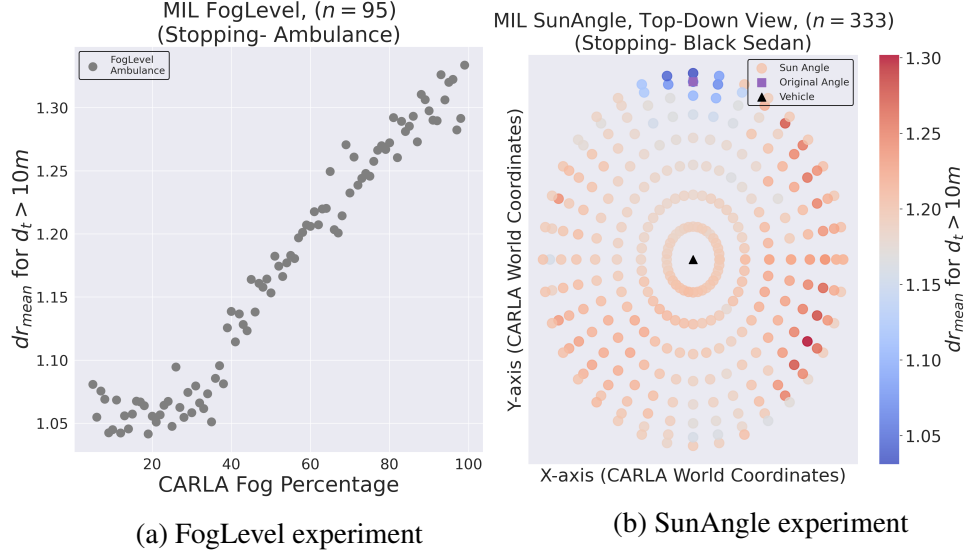


Figure 3.8: Results for the *MIL* simulation experiments.

Table 3.2: Pearson Correlation Coefficient (PCC) statistical test results. Note that the predictive and response metrics for the *MIL*-to-*VIL* statistical tests are sample means, defined by Eq. (3.8).

Driving Type	Predictive Metric	Response Metric	<i>VIL</i> -to- <i>VIL</i> (diagnosis)			<i>MIL</i> -to- <i>VIL</i> (prediction)		
			r	p -value	n	r	p -value	n
Stopping	$dr_{\text{mean}} (d_t > 10\text{m})$	$tt_{\text{C}_{\text{min}}}$	-0.89	0.0	45	-0.9	0.001	9
Following	dr_{mean}	fd_{mean}	-0.87	0.0	24	-0.87	0.02	8
Following	$\hat{cd}_{\text{mean}} (+500\text{ms})$	cd_{mean}	0.9	0.0	24	0.93	0.001	8

sun angles around the vehicle affect performance when in otherwise clear conditions. We notice that glare directly in front of the vehicle lowers dr_{mean} significantly, and is the only sun angle that seems to have this effect. Meanwhile, with the sun on the right-rear of the vehicle, this greatly degrades performance. From the experimental setup, we can deduce that shadows from pillars on the side of the road are having this effect, seen visually in Fig. 3.3. This angle could be considered a safety critical edge case, discovered only with the combination of *VIL* and *MIL* simulation techniques.

3.5 Summary

In this work, we leverage the *Openpilot* perception system and controller for *SIL*, *VIL*, and *MIL* simulation of SAE L2 autonomous vehicles. We propose a novel *VIL* setup, in which images are piped directly from a virtual camera in CARLA to *Openpilot*, which controls the vehicle on a single roller dynamometer with steering enabled synchronized wheel speed (SeSwS). Our *VIL* setup is compared to an *SIL* setup against safety and performance metrics, and the performance of the *VIL* response is diagnosed with respect to the output of the perception model. Additionally, a combination approach to simulation is introduced, where the real dynamic response of *VIL* simulation is predicted from a *MIL* simulation strategy. The real-world accuracy of *VIL* response, paired with the speed and simplicity of *MIL* simulation, allow for a deep exploration of realistic safety-edge cases for autonomous vehicles, with minimized cost. This combinational approach is used to further explore safety edge cases for our system around the effects of fog and sun angle.

Our approach in this chapter applies an extremely realistic *VIL* simulation approach to capture the dynamic response of the vehicle. While the dynamics of the vehicle are updated in simulation with a simplistic bicycle model defined by Equations (3.1) to (3.3), the real wheel speeds and steering angles inform the movement. Additionally, the constraints of CAN bus latency and other hardware related factors are accounted for. However, the perception system underlying this research still uses images directly from simulation, and there is assuredly a noteworthy domain shift gap between CARLA and the physical world. Using the model architecture defined in Chapter 4, we build on this work in Chapter 5 by replacing the *supercombo* perception system with a *perception error model* (PEM).

Chapter 4

Fully Embedded Time-Series Generative Adversarial Networks

4.1 Objective

Generative Adversarial Networks (GANs) should produce synthetic data that fits the underlying distribution of the data being modeled. For real valued time-series data, this implies the need to simultaneously capture the *static* distribution of the data, but also the full *temporal* distribution of the data for any potential time horizon. This temporal element produces a more complex problem that can potentially leave current solutions under-constrained, unstable during training, or prone to varying degrees of mode collapse. In *FETSGAN*, entire sequences are translated directly to the generator’s sampling space using a *seq2seq* style adversarial autoencoder (AAE), where adversarial training is used to match the training distribution in both the feature space and the lower dimensional sampling space. This additional constraint provides a loose assurance that the temporal distribution of the synthetic samples will not collapse. In addition, the First Above Threshold (FAT) operator is introduced to supplement the reconstruction of encoded sequences, which improves training stability and the overall quality of the synthetic data being generated. These novel contributions demonstrate a significant improvement to the current state of the art for adversarial learners in qualitative measures of temporal similarity and quantitative predictive ability of data generated through *FETSGAN*.

4.2 Proposed Architecture

Generative modeling is the field of research concerned with producing new and unique data that is similar to the data that was used to produce the model. More specifically, we can say that this similarity is defined in terms of the ability to model the underlying distribution represented by the training data. In the case of sequential vector $x_{1:T}$ with length T , the data distribution is characterized by the temporal distribution $p(x_1, \dots, x_T)$. Even with relatively simple datasets where the vector x_t is low dimensional, compounding dependencies in the temporal distribution increase with T until it becomes difficult to measure or even visualize the similarity or differences between the temporal distributions of the training data and the generated data. Generative Adversarial Networks (GANs) have demonstrated exceptional ability in modeling complex distributions such as these. However, GANs are notoriously difficult to train, with instability often preventing convergence and the final generative models featuring some degree of mode collapse, where only a portion of the full target distribution is represented in the synthetic samples.

Like much of the work surrounding GANs, the novel process presented here provides additional constraints to the adversarial learning process that regularizes learning, resulting in greater stability during training, higher quality data, and less susceptibility to mode collapse, specifically in the temporal distributions. The architecture presented is a modification of RCGAN[55] or C-RNN-GAN[126] that features a *seq2seq* style AAE as encoder and decoder for data generation [58], [127]. The complete model is visualized in Fig. 4.1. There are three primary benefits to using an adversarial autoencoder. First, there is an additional constraint that matches the posterior distribution of the encodings to the prior distribution of the samples at inference time, further combating mode collapse beyond feature-level adversarial training. Second, the *seq2seq* encoder is forced to summarize the entire sequence at once, allowing it to capture relevant time dependencies of arbitrary length. This can be compared to the approach in the TimeGAN, where the regularizing effect of the teacher-forced supervised losses of the encodings are only applied one time-step into the future [57]. Finally, the posterior encodings of adversarial autoencoders are natively interpretable. This allows fine control over the style of the synthetic data being generated, even in the completely unsupervised learning setting of this work.

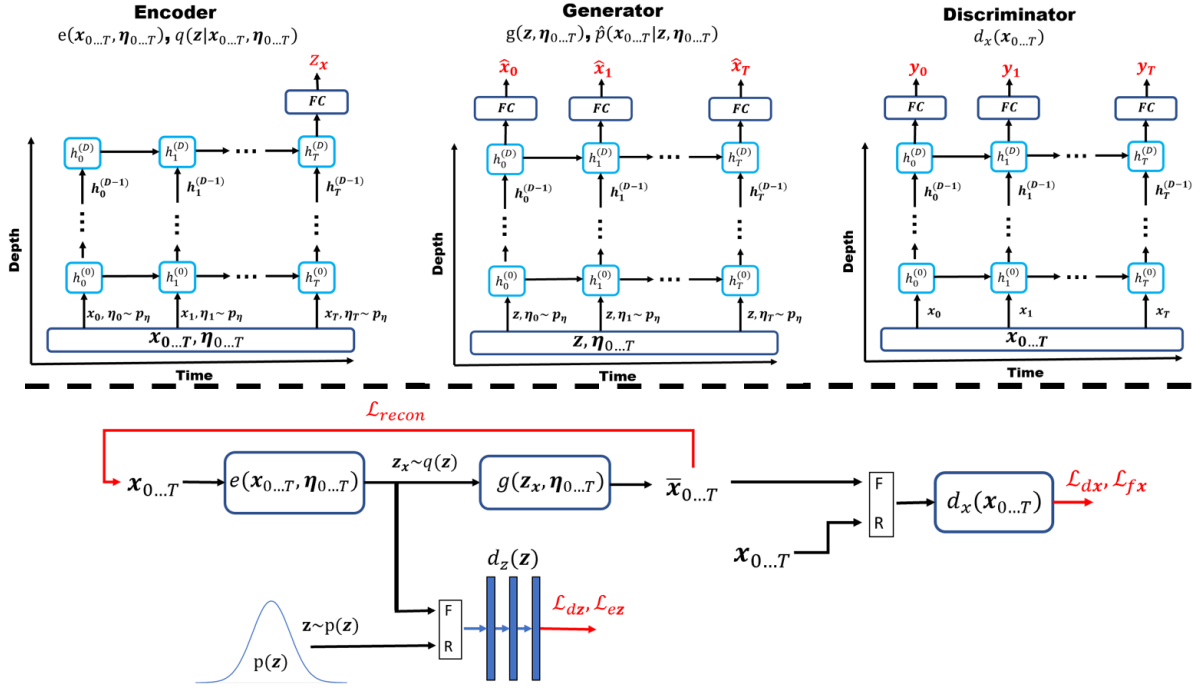


Figure 4.1: An overview of the *FETSGAN* training scheme. Above the dashed line, the RNN style architectures are detailed, showing the model outputs (red) of each network as a function of the inputs. $h_t^{(d)}$ indicates a hidden state of the network, where d represents the weights associated with a specific depth and t represents that state any specific time. FC indicates a fully connected output layer, and these weights are shared for every output across time, i.e. in the case of the generator $\hat{x}_{1:T}$. Below the dashed line, the training flow is visualized. The mechanisms for producing *FETSGAN*'s five objective functions are shown in red.

The decoder in our framework can potentially suffer from the same supervised training problems as any other autoregressive model where compounding errors over time across shared weights can cause slow or unstable training, especially during the reconstruction of long encoded sequences. Typically this is resolved with some variation of teacher-forcing [128], [129], where the network is provided the ground truth from the previous timestep, and learns to predict only one timestep into the future. This method often leaves the network with some degree of exposure bias, where the compounding error of the model’s own predictions are neglected. Additionally in the context of real-valued time-series generation, statistical properties in the data may produce harsh local minimums for regression based optimization. In this work, we do not use an autoregressive decoder at all. Instead, our solution to reconstruction loss is coined First Above Threshold (FAT) loss. In this scheme, stochastic gradient descent is only applied to the model parameters at one time instance per generated sequence. This allows the network to learn progressively longer sequences during training, and can be applied to any element-wise loss function in a supervised or unsupervised manner. The work described here is an extension of RCGAN and a reformulation of TimeGAN that utilizes a supervised loss in the feature space as well as an unsupervised loss on the encodings of the data which produces improved matching of the temporal distribution of the training data. In addition, FAT loss can improve the quality of the data being generated, as well as producing a standardizing effect on the training dynamics. For our experiments, stocks data, energy usage data, metro traffic data, and a synthetic sine wave dataset are used. The sequences produced are qualitatively analysed to show significant improvement in preventing mode collapse along the temporal distribution. In addition, we demonstrate the ability to selectively sample our model at inference time to produce realistic data of a specific style. Finally, we measure the performance of our model against the primary adversarial learners in terms of predictive and discriminative scores. *FETSGAN* shows significant improvement to these methods in all stated dimensions of analysis.

4.3 Problem Formulation

Consider the random vector $X \in \mathcal{X}$, where individual instances are denoted by x . We operate in a discrete time setting where fixed time intervals exist between samples X_t , forming sequences of length T , such that $(X_1, \dots, X_T) := X_{1:T} \in \mathcal{X}^T$. We note that T may be a constant value

or may be a random variable itself, and that the proposed methodology does not differ in either case. The representative dataset of length N is given by $\mathcal{D} = \{x_{n,1:T}\}_{n=1}^N$. For convenience, the subscript n is omitted in further notation.

The data distribution $p(X_{1:T})$ describes the sequences, such that any possible conditional $p(X_t | X_{t-i})$ for $t \leq T$ and $t - i \geq 0$ is absorbed into $p(X_{1:T})$. Thus, the overall learning objective is to produce the model distribution $\hat{p}(X_{1:T})$,

$$\min_{\hat{p}} D(p(X_{1:T}) || \hat{p}(X_{1:T})) \quad (4.1)$$

where D is some measure of divergence between the two distributions. We note here that RCGAN [55] applies adversarial loss that minimizes this divergence directly. In the following works, higher quality data is generated by supplementing this loss function with a supervised loss that is more stable [57], or by avoiding adversarial loss altogether [62]. While the adversarial autoencoder in Section 4.3.1 is intended to supplement and further stabilize training as well, we also highlight the additional ability of the FAT operator to stabilize the minimization of this objective on reconstructions in Section 4.3.2.

4.3.1 Adversarial Autoencoder

In order to represent complex temporal relationships, we would like to leverage the ability to encode the entire time-series as a low dimensional vector in a *seq2seq* style model. We introduce the random variable $Z \in \mathcal{Z}$ as an intermediate encoding for producing $\hat{p}(X_{1:T})$. We also introduce $\eta_{1:T}$ as a random noise vector from a known distribution p_η sampled independent of time. Let $p_z(Z)$ be a known prior distribution, the encoder function $e : (\mathcal{X}^T, \eta^T) \rightarrow \mathcal{Z}$ have the encoding distribution $q(Z | X_{1:T}, \eta_{1:T})$, and the generator (or decoder) function $g : (\mathcal{Z}, \eta^T) \rightarrow \mathcal{X}^T$ have the decoding distribution $\hat{p}(X_{1:T} | Z, \eta_{1:T})$.

Also, we can define an ideal mapping function that maps sequences to vectors $M : (\mathcal{X}^T, \eta^T) \rightarrow \mathcal{M}$, such that $M_X = M(X_{1:T}, \eta_{1:T})$ and $p_M(M_X) = p(X_{1:T})p_\eta(\eta_{1:T})$. The aggregated posterior can now be defined,

$$q(Z_X) = \int_{M_X} q(Z | X_{1:T}, \eta_{1:T}) p_M(M_X) dM_X \quad (4.2)$$

Similar to Eq. (4.1), encoder training occurs by matching this aggregated posterior distribution $q(Z_X)$ to an arbitrary prior distribution $p_Z(Z)$.

$$\min_q D(p_z(Z) \parallel q(Z_X)) \quad (4.3)$$

The posterior distribution $q(Z_X)$ is equivalent to the universal approximator function in [58], where the noise η_t exists to provide stochasticity to the model in the event that not enough exists in the data $\mathcal{D}$ for $q(Z_X)$ to match $p_z(Z)$ deterministically.

The generator is trained through reconstructing the data distribution $p(X_{1:T})$ while being conditioned on the encodings.

$$\min_{\hat{p}} D(p(X_{1:T}) \parallel \hat{p}(X_{1:T} \mid Z_X, \eta_{1:T})) \quad (4.4)$$

If divergence is minimized in Eq. (4.3) and Eq. (4.4), then the encoder is able to perfectly replicate the prior distribution, and the generator is perfectly able to reconstruct $X_{1:T}$, resulting in a complete model of $p(X_{1:T})$ when the prior distribution $p_z(Z)$ is sampled and decoded at inference time.

4.3.2 First Above Threshold (FAT) Operator

In our formulation, the generator model minimizes two measures of divergence. First, there is the direct matching of the target distribution through adversarial training, characterized by Eq. (4.1). Then, there is the secondary reconstruction loss from the intermediate encodings, characterized by Eq. (4.4). Reconstructions of time series data are denoted $\bar{x}_{1:T} = g(e(x_{1:T}))$. The simplest loss function to enforce reconstruction of the original data might be,

$$\mathcal{L} = \mathbb{E}_{x_{1:T} \sim p} \left(\sum_t \|x_t - \bar{x}_t\|^2 \right) \quad (4.5)$$

where this objective is minimized in tandem by the generator and the encoder. This may prove to be challenging as our generator model is not an auto regressive model. It takes the intermediate encodings as input at every timestep, as shown in Fig. 4.1. This reduces the possibility of a vanishing gradient to the encoder itself, and also prevents the generator from forgetting the initial encoding for long sequences. This does, however, make learning through

reconstruction more difficult due to the inability to incorporate teacher forcing methods. In addition, time-series reconstruction with real-valued data faces the optimization problem of large local minimums, where the model may collapse into producing only mean values across any one time-series or the mean of the entire dataset. To alleviate these problems, we propose the solution of only applying reconstruction loss at *one* time instance $t = \tau$ in the sequence, instead of the entire time series at once. Thus, the reconstruction loss,

$$\mathcal{L}_{recon} = \mathbb{E}_{x_{1:T} \sim p}(\|x_\tau - \bar{x}_\tau\|^2) \quad (4.6)$$

trains the generator in a supervised manner. The question remains which time instance τ to choose. We propose a simple solution. Prior to training, we define a real-valued threshold ϵ , such that any error that has compounded to produce the reconstruction $\bar{x}_t$ is *acceptable* so long as $\|x_t - \bar{x}_t\|^2 < \epsilon$. In this way, time-series reconstructions are progressively learned from short-term to long-term, instead of all at once. To this end, the First Above Threshold (FAT) operator is introduced. This operator takes as input a sequence $l_{1:T}$ and a threshold ϵ , such that the minimum value for t is returned where $l_t > \epsilon$. In the event that $l_t < \epsilon \forall t \in T$, $\text{FAT}_t(l_{1:T}, \epsilon) = \arg \max_t(l_{1:T})$. With this newly defined operator,

$$\tau = \text{FAT}_t(\|x_t - \bar{x}_t\|^2, \epsilon) \quad (4.7)$$

defines τ , thus defining the complete form of Eq. (4.6). The benefits are two fold. First, a progressive learning approach stabilizes the early portion of training, as the objective function may be less likely to become stuck at a local minimum while trying to encode and reconstruct long sequences all at once. Second, updating the parameters corresponding to only one time instance at a time has a regularizing effect by providing a more granular gradient that is less likely to interfere with the adversarial training for both the encoder and generator. The combination of both of these effects are demonstrated in Fig. 4.2, as the training dynamics of the model are compared between using the reconstruction loss of Eq. (4.5) and Eq. (4.6). We show that the application of the objective in Eq. (4.6) actually minimizes the loss of Eq. (4.5) more effectively than applying it directly in the sines dataset. The efficacy of the FAT_t operator can be expected to grow with longer, more complex sequences.

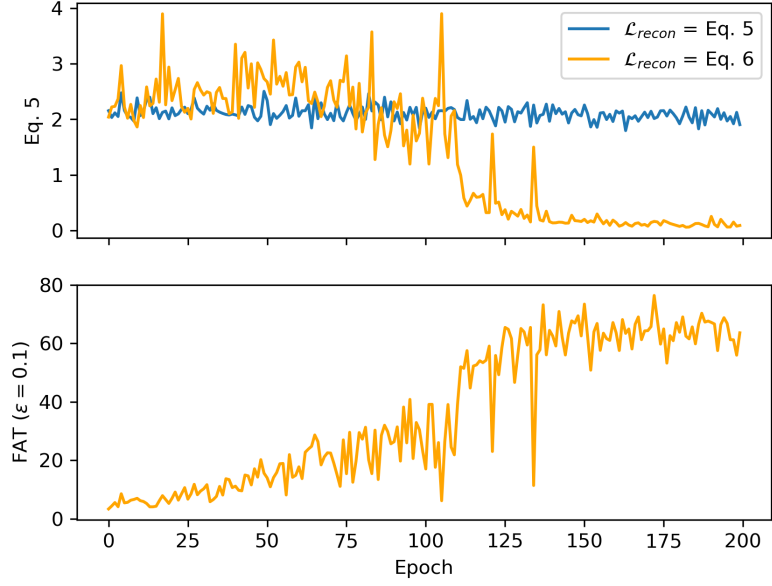


Figure 4.2: A demonstration of the utility of the *FAT* operation is shown for the Sines dataset. For the encoding and reconstruction of $\bar{x}_{1:T} = g(e(x_{1:T}))$, the reconstruction objectives of Eq. (4.5) and Eq. (4.6) are compared for the sines dataset. The average value for each epoch in 200 epochs of training is plotted. Here, training occurs under the complete model objective of Eq. (4.12). In the first row, we can see that the model is unable to apply adversarial learning and simultaneously learn the proper encodings to reconstruct $x_{1:T}$ under Eq. (4.5). This causes the optimization to immediately fall into a local minimum. With τ gradually increasing to progressively learn longer sequences in the second row, the FAT_t operation facilitates minimizing reconstruction loss better than applying Eq. (4.5) directly.

4.3.3 Complete Model

Time series data collected in the physical world such as sensor measurements will have some degree of stochasticity. We cannot replicate this stochasticity using a reconstruction objective alone. To this end, we introduce the feature space discriminator $d_x : \mathcal{X}^T \rightarrow \mathcal{Y}^T$ which maps sequences to classifications, such that $y_{1:T} = d(x_{1:T})$ and $\hat{y}_{1:T} = d(\bar{x}_{1:T})$. This loss is applied to the reconstructions $\bar{x}_{1:T}$, such that the objective can be minimized both through the encoder and the generator. In the Least Squares GAN form, the objective function of the discriminator described by,

$$\mathcal{L}_{dx} = \frac{1}{2} \mathbb{E}_{x_{1:T} \sim p} \left(\sum_t \|1 - y_t\|^2 \right) + \frac{1}{2} \mathbb{E}_{\bar{x}_{1:T} \sim \hat{p}} \left(\sum_t \|\hat{y}_t\|^2 \right) \quad (4.8)$$

$$\mathcal{L}_{fx} = \frac{1}{2} \mathbb{E}_{\bar{x}_{1:T} \sim \hat{p}} \left(\sum_t \|1 - \hat{y}_t\|^2 \right) \quad (4.9)$$

We now introduce the encoding discriminator $d_z : \mathcal{Z} \rightarrow \mathcal{Y}_Z$ such that $y_z = d_z(z)$ and $\hat{y}_z = d_z(z_x)$. Also in the LSGAN form,

$$\mathcal{L}_{dz} = \frac{1}{2} \mathbb{E}_{z \sim p_z} (\|1 - y_z\|^2) + \frac{1}{2} \mathbb{E}_{z_x \sim q} (\|\hat{y}_z\|^2) \quad (4.10)$$

$$\mathcal{L}_{ez} = \frac{1}{2} \mathbb{E}_{z_x \sim q} (\|1 - \hat{y}_z\|^2) \quad (4.11)$$

describe the objective functions for the discriminator and encoder, respectively.

Putting everything together, there are three measures of divergence we will minimize with our complete model. The adversarial training between the objective functions described by Eq. (4.8) and Eq. (4.9) minimize Eq. (4.1) directly, where D is the χ^2 -divergence in the LSGAN formulation. The adversarial training described by the objective functions Eq. (4.10) and Eq. (4.11) apply to the divergence of Eq. (4.3), also minimizing χ^2 -divergence of the intermediate encodings. Finally, Eq. (4.6) corresponds to Eq. (4.4), minimizing the Kullback–Leibler (KL) divergence through maximum likelihood (ML) supervised training. In

total, all parameter optimization occurs through,

$$\begin{aligned}
& \min_e \min_g (\lambda_1 \mathcal{L}_{recon} + \mathcal{L}_{ez} + \lambda_2 \mathcal{L}_{fx}) \\
& \min_{d_z} (\mathcal{L}_{dz}) \\
& \min_{d_x} (\mathcal{L}_{dx})
\end{aligned} \tag{4.12}$$

thus describing the complete objective of the *FETSGAN* architecture.

4.4 Implementation

The hyperparameters of the model are λ_1 , λ_2 , and ϵ . Additionally, while not described in notation, the dimensionality of p_z and p_η are also parameters of the model. In terms of tuning these values, p_z should generally correspond to the estimated complexity of the input signals while p_η should correspond to the complexity of the noise. Overestimating these values may simply lead to more complex encodings than are required, and potentially more unstable learning in the embedding space described by Eqs. (4.10) and (4.11). λ_1 and λ_2 depend primarily on the scaling of the data. In our experiments, all data is normalized between $(-1, 1)$. Finally, ϵ should be chosen based on the model’s use-case regarding what amount of error is acceptable in reconstruction. In other words, this value represents the amount of error tolerable for the signal, where any error under this value can be considered stochastic noise subject to adversarial feature space learning. All hyperparameters are fairly robust. To demonstrate this, the parameters are fixed for all experiments. The primary parameters are $\lambda_1 = 10$, $\lambda_2 = 1$, and $\epsilon = 0.1$. The prior distribution p_z and the noise distribution p_η contain four dimensions, and both are sampled from $p_z, p_\eta \sim \mathcal{U}(-1, 1)$. All models are trained with the Adam optimization strategy [130], where the learning rate for the generator, encoder and both discriminators is 0.001. These learning rates decay exponentially in the last 10% of epochs. The full model implementation in Pytorch and instructions for experimental reproduction are provided in the link. [*]

*<https://github.com/jbeck9/FETSGAN>

4.4.1 Experimental Setup & Datasets

For our experiments, we compare the performance of our model primarily against TimeGAN [57] and RCGAN [55]. As a baseline comparison, we have also included a purely autoregressive method that was trained using only teacher forcing. The methods of [61, 62, 63] are omitted at the time of writing due to a lack of available implementations for reproduction. Due to those limitations, we limit the scope of our conclusions to time series models with adversarial learning applied directly to the feature space, or low dimensional encodings of the feature space.

The efficacy of our model is shown along three dimensions. First, we demonstrate the qualitative similarity between the original data and our generated data in Section 4.4.2. Then, we demonstrate the unique ability of our method to interpret the prior sampling distribution as a way of providing selective samples that are similar in style to specific samples from the original dataset in Section 4.4.3. Finally, we demonstrate that under strenuous classification and prediction tasks, our method holds state of the art performance among adversarial learners that apply directly on the feature space for time series data in Section 4.4.4.

We use three primary datasets for analysis. First, we generate a one dimensional sines dataset of length $T = 100$, where the amplitude, frequency, and phase for each sequence are sampled from a uniform distribution. We also use the six-dimensional historical Google stocks dataset, containing stock price information from 2004 to 2019 of various lengths T . This dataset is provided directly in the code repository. Finally, we use 6 dimensions of the UCI Appliances energy dataset [131] and real-valued traffic and weather data from the UCI Metro Interstate dataset [132].

4.4.2 Distribution Matching

Visualizing synthetic data generated from an adversarial learning process is important to analyze the extent of mode collapse that may have occurred. This task is tricky for time series data, as it is possible that temporal mode collapse could exist, but be obscured if the temporal dimension is flattened for analysis. In the case of the sines dataset, we can reduce the sequences to a single dimension by simply capturing the *dominant frequency* in the sequence, using the Discrete Fourier Transform (DFT). Here, the dominant frequency is given by $\arg \max_f DFT(x_{1:T})$. Since the original data consists of a sine wave with

a single frequency, this provides a valid analysis of the entire sequence. The amplitude and phase of the corresponding dominant frequency are taken as well. A histogram can then be produced, comparing the distribution of frequencies, amplitudes, and phases in each dataset. For the stocks and energy datasets, we settled for a visual comparison in the flattened temporal dimension using TSNE visualization [133], repeating the procedure reported in [57]. The results of these visualizations are shown in Fig. 4.3. While the results for TimeGAN generally match what is shown in [57], *FETSGAN* demonstrates a substantial improvement over TimeGAN, RCGAN, and the baseline autoregressive models in matching the underlying distribution for all datasets used.

4.4.3 Selective Sampling

There is an obvious use case that at inference time, perhaps there is a need to produce a specific *style* of data, or to sample from a specific portion of the data distribution $p(X_{1:T})$. Because our model forces dimensionality reduction through an encoder that is forced to match q to p_z , we can leverage spatial relationships in the latent space z to selectively sample from the prior distribution p_z at inference time. This allows us to produce synthetic data that retains a specific style. To demonstrate this, three sine waves were taken from the data, corresponding to sequences $x = \sin(2\pi ft)$ with $f = 2, 5, 8$. These sequences were then encoded as $z_x = e(x)$. Finally, new sequences were generated by adding noise to these encodings, such that $x_\eta = g(z_x + \eta)$, where $\eta \sim \mathcal{N}(0, 0.1)$. The results in Fig. 4.4 show that, as expected, spatial relationships are maintained in the latent space z . We are able to produce synthetic sine waves that maintain the close relationship to the anchor point x they were sampled near, allowing the ability to produce synthetic sequences within an expected range of style.

4.4.4 Performance Results

To compare quantitative performance between models, we apply two testing metrics, *discriminative score*, and *predictive score*. Discriminative score is measured by training an ad hoc RNN classifier to discriminate between real dataset and a static synthetic dataset generated by each model. The best model will have the lowest score $0.5 - pred$, corresponding to how far the predictions were below the decision boundary, where a score of 0 corresponds to indistinguishable data. In the case of prediction, the “Train on Synthetic, Test on Real” (TSTR) approach is used [55]. A simple RNN is trained as a forecasting model, predicting 1,

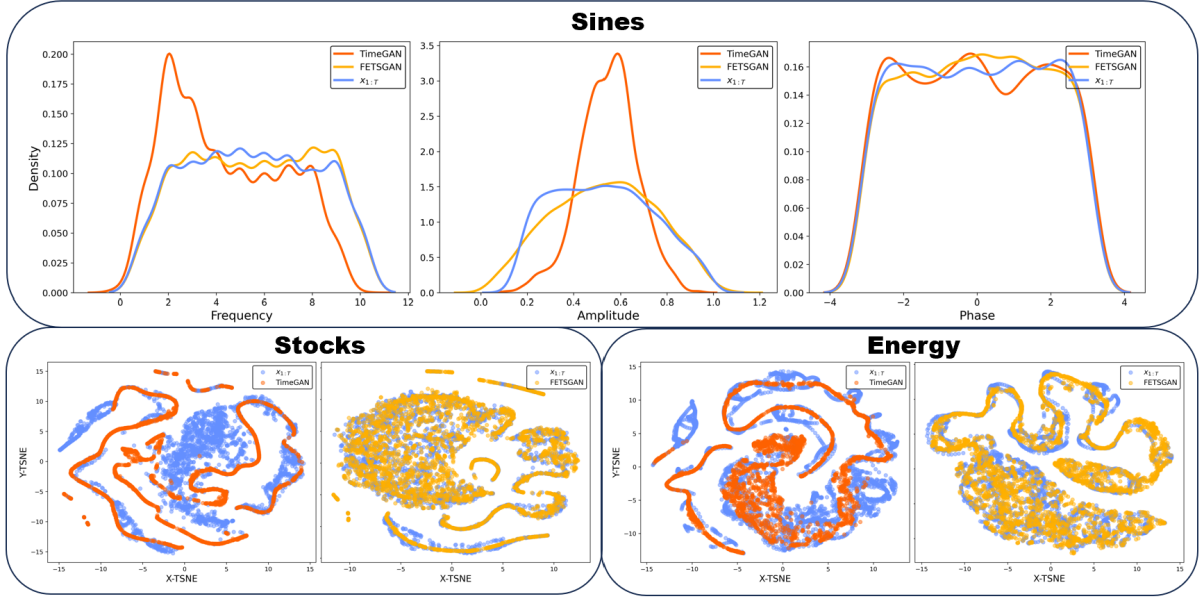


Figure 4.3: Qualitative visualization results. Sines data is shown on the top row, with the dominant component of the DFT for each sequence’s frequency, amplitude, and phase (left to right) is shown as both a histogram and kernel density estimate (KDE). On the bottom row, we show a TSNE visualization for the stock dataset (left) and energy data (right), following the same procedure from [57]. In all graphs, real data $x_{1:T}$ is shown in blue, synthetic data $\hat{x}_{1:T}$ (FETSGAN) is in orange, and the synthetic data generated by TimeGAN is shown in red. FETSGAN produces data that more closely matches the original data by a substantial margin.

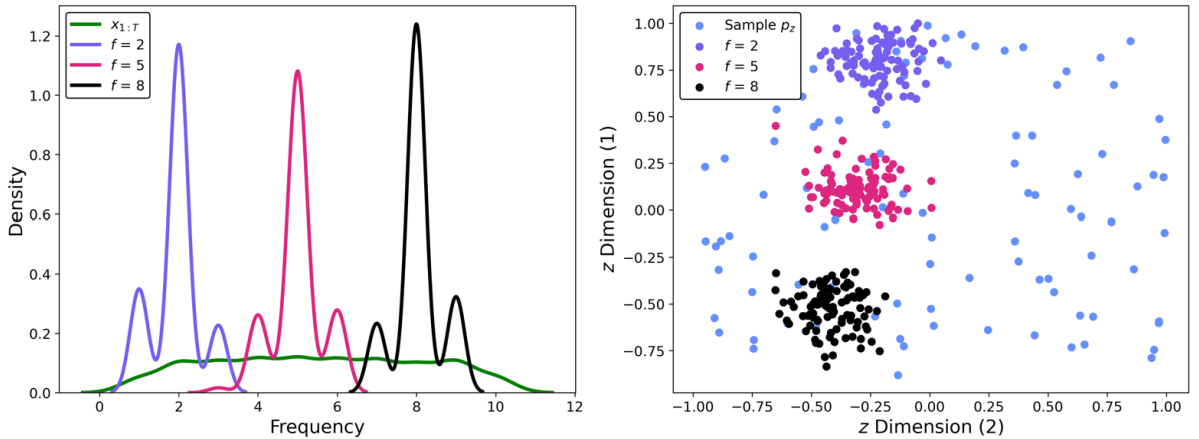


Figure 4.4: A demonstration of selective sampling of the prior distribution p_z . Sines data is shown, where 100 random samples z_s were taken near the encodings of three sine waves with frequencies $f = 2, 5, 8$ and then fed to the generator. Similar to 4.3, the corresponding histogram and KDE plot for $DFT(g(z_s))$ is shown on the left. Projections of the intermediate encodings are shown on the right, demonstrating spatial interpretability.

3, and 5 steps into the future, given the sequence $x_{1:t}$ for any valid step where $t + \text{step} \leq T$ on synthetic data. Then, MAE prediction error from the trained model is measured on the real dataset. The best model is the one which produces the lowest prediction error on real data. Whenever the model under test calls for an RNN style network, a Gated Recurrent Unit (GRU) of 64 cells and 3 layers was used. For each architecture, 3 models were trained, and sampled 5 times. Thus, 15 tests were conducted for each value in Table 4.1. Training for all models occurred under 1000 epochs using the Adam optimizer with a learning rate of 0.001, including the classification and prediction models. Variations of *FETSGAN* are included to analyze sources of gain. *FETSGAN without FAT* removes the FAT_t operation by replacing Eq. (4.6) with Eq. (4.5). *FETSGAN without FD* trains without the feature space discriminator d_x . The complete model scores best in totality, and the variations of the *FETSGAN* show statistically significant improvement over RCGAN and TimeGAN in all cases. We note that in the case of the noisy and lengthy Energy dataset, only the complete *FETSGAN* model was able to produce data realistic enough to completely fool the ad-hoc discriminator with a score under 0.1 for all experiments. The number of trainable parameters for each model is also shown in Table 4.1, which was fixed for all experiments.

4.5 Limitations & Ethical Considerations

Fully embedding time-series data using an adversarial autoencoder ensures the synthetic data produced more closely matches the full distribution of the target data, and this auto-encoding reconstruction has a regularizing effect on the adversarial training. However, this also limits the potential applications of this methodology. Data such as video or language tokens are too highly dimensional to be reliably reduced to vector embeddings. As such, the realistic use cases are limited to real-valued time-series signals with relatively low dimensionality. The architecture chosen in this work was motivated in part by the fact that it is not auto-regressive in nature, and thus does not suffer from exposure bias. Limiting exposure bias would allow for more robust use of auto-regressive methods such as attention-based transformers, and constitutes a promising pathway for future work.

While the applications of our work may not reach the scope and magnitude of other synthetic data generation models such as large language models (LLMs), they are similar with

Table 4.1: Predictive & Discriminative Scores. Best scores shown in **bold**.

Model	Metric	Sines	Energy	Stocks	Metro
T-Forcing (64,134 parameters)	+1 Step	.011 $\pm$.004	.018 $\pm$.002	.047 $\pm$.003	.093 $\pm$.002
	+3 Step	.083 $\pm$.010	.028 $\pm$.002	.064 $\pm$.006	.168 $\pm$.004
	+5 Step	.421 $\pm$.220	.034 $\pm$.003	.084 $\pm$.014	.223 $\pm$.013
	Dis. Score	.494 $\pm$.008	.246 $\pm$.053	.243 $\pm$.031	.264 $\pm$.021
RCGAN (93,717 parameters)	+1 Step	.011 $\pm$.002	.035 $\pm$.007	.026 $\pm$.001	.065 $\pm$.002
	+3 Step	.038 $\pm$.010	.045 $\pm$.007	.073 $\pm$.020	.134 $\pm$.003
	+5 Step	.073 $\pm$.020	.054 $\pm$.008	.094 $\pm$.026	.180 $\pm$.003
	Dis. Score	.467 $\pm$.040	.490 $\pm$.011	.459 $\pm$.018	.114 $\pm$.031
TimeGAN (337,858 parameters)	+1 Step	.046 $\pm$.043	.021 $\pm$.002	.039 $\pm$.001	.146 $\pm$.038
	+3 Step	.133 $\pm$.121	.026 $\pm$.001	.048 $\pm$.002	.271 $\pm$.047
	+5 Step	.167 $\pm$.132	.031 $\pm$.001	.053 $\pm$.002	.330 $\pm$.054
	Dis. Score	.317 $\pm$.105	.164 $\pm$.066	.268 $\pm$.054	.374 $\pm$.104
FETSGAN w/o FAT (262,686 parameters)	+1 Step	.016 $\pm$.004	.025 $\pm$.003	.030 $\pm$.001	.063 $\pm$.001
	+3 Step	.041 $\pm$.006	.029 $\pm$.001	.039 $\pm$.002	.124 $\pm$.002
	+5 Step	.058 $\pm$.008	.032 $\pm$.001	.042 $\pm$.001	.165 $\pm$.002
	Dis. Score	.249 $\pm$.125	.237 $\pm$.114	.055 $\pm$.032	.056 $\pm$.031
FETSGAN w/o FD (182,827 parameters)	+1 Step	.008 $\pm$.005	.018 $\pm$.002	.031 $\pm$.002	.077 $\pm$.003
	+3 Step	.019 $\pm$.004	.026 $\pm$.001	.037 $\pm$.001	.155 $\pm$.008
	+5 Step	.028 $\pm$.003	.031 $\pm$.002	.041 $\pm$.002	.192 $\pm$.006
	Dis. Score	.002 $\pm$.007	.189 $\pm$.041	.086 $\pm$.031	.122 $\pm$.066
FETSGAN (262,686 parameters)	+1 Step	.007 $\pm$.006	.016 $\pm$.001	.026 $\pm$.001	.062 $\pm$.001
	+3 Step	.017 $\pm$.005	.023 $\pm$.001	.036 $\pm$.001	.128 $\pm$.002
	+5 Step	.026 $\pm$.003	.028 $\pm$.001	.040 $\pm$.001	.169 $\pm$.001
	Dis. Score	.001 $\pm$.004	.030 $\pm$.012	.005 $\pm$.007	.025 $\pm$.014

respect to the ethical considerations highlighted in [70]. Our methodology can straightforwardly be used to produce synthetic datasets where privacy is a concern within the original data. The primary responsibility for using this methodology and those like it is to ensure it is made explicitly clear that the data produced is synthetic. Additionally, the creator of the model should thoroughly ensure it sufficiently matches the distributions of the target data before applying it to problems with real-world impact.

4.6 Summary

In this chapter we introduce *FETSGAN*, a novel approach to real-valued time series generation that combines feature space adversarial learning with the adversarial autoencoder framework. In addition, we introduce the FAT operator, which provides a regularizing effect on training complex temporal sequences that are produced from an intermediate encoding. Finally, the method shown here provides an interpretable latent space, allowing higher flexibility for selective sampling at inference time. We demonstrate significant improvement over current adversarial methods applied directly to the feature space or encodings thereof. In future work, we intend to leverage the accuracy and interpretability of this model on a variety of datasets to demonstrate real world utility for synthetic data to aid in various applied machine learning models in forecasting and classification.

Chapter 5

Improving Realism Through Perception Error Models

5.1 Objective

In this chapter, we introduce a perception error model (PEM) based on the *FETSGAN* architecture from Chapter 4. In our formulation, the perception error model can be considered a "probabilistic model of a perception model". Here, the perception model under scrutiny is the pre-trained *supercombo* model [90] provided by the open-source *openpilot* SAE Level 2 self-driving platform. Our *FETSGAN* based PEM is trained using the *Comma2k19* dataset [134], which provides video and radar information for a wide range of conditions best fitted for an adaptive cruise control driving setting. Once trained, we compare the output of our PEM model against the performance of *Supercombo* on the real dataset, making some relevant assertions for the realism of our approach. We then use our PEM in a simulated environment, borrowing much of the methodology from Chapter 3. Ultimately, this allows for two major points of understanding. First, we are able to use the PEM to get a sense of the domain shift error within CARLA whenever the vision-based system is used natively, highlighting the utility of our approach. Second, with a higher amount of sampling than what is present in Chapter 3, we are able to leverage the probabilistic nature of our model to provide stronger assertions to the safety performance of the vehicle across a broad spectrum of conditions.

5.2 Problem Formulation

5.2.1 Modified *FETSGAN*

In Section 4.3, we outline the problem formulation for fully embedded time-series generative adversarial networks. For our implementation of perception error models using *FETSGAN*, we make some minor adjustments to the formulation of the problem. We still consider real-valued time-series signals, denoted as $(X_1, \dots, X_T) := X_{1:T} \in \mathcal{X}^T$. In our context, these time series signals $X_{1:T}$ are the perception error terms that define the output of our modified perception system, formally defined in the next section. We also introduce in notation $S \in \mathcal{S}$, where individual instances are denoted as s . Here, S denotes supplementary information that may be useful in producing the model distribution $\hat{p}(X_{1:T})$. In the context of our PEM, this is generally information easily available to us both in simulation as well as the real-world dataset being modeled, such as *ground truth* labels. These signals are still denoted as time-series signals that have pair-wise mappings to X , denoted by $[(X_1, S_1), \dots, (X_T, S_T)] := (X, S)_{1:T} \in (\mathcal{X}, \mathcal{S})^T$. It is possible that some supplementary information is categorized as a descriptor of the entire sequence, and is not time-series in nature, such as what is described in [57]. For the sake of simplicity, we still encode this information as a constant-valued time-series, such that $s_t = s_0$ for all $t \in \mathbb{N}$. Just as in Section 4.3, The data distribution $p(X_{1:T})$ describes the distribution of the target sequence, which is perception errors in this context. Now, we define this distribution such that any possible conditional $p(X_t \mid S_{t-i}, X_{t-i})$ for $t \leq T$ and $t - i \geq 0$ is absorbed into $p(X_{1:T})$. At this point, the divergence measure described in Eq. (4.1) applies identically, as well as the rest of the formulation that culminates in the total objective function of Eq. (4.12).

In addition to the conditional vector S_t , we also introduce a binary detection variable to our model formulation that is not present in the real-valued signals of Chapter 4. We denote the binary perception model output $b_t^{(p)}$ as the presence of a detected lead vehicle. Meanwhile, $\hat{b}_t^{(p)}$ is the output of our PEM, and is real-valued between 0 and 1. As is common in literature, we also introduce the threshold θ to discretize $\hat{b}_t^{(p)}$, where $\hat{b}_t^{(p)} \geq \theta$ is considered a detection, and $\hat{b}_t^{(p)} < \theta$ is considered a missed detection. This model output is trained on mean binary cross entropy loss [135], and this loss can be added as $\lambda_1 \mathcal{L}_b$ to the objective Eq. (4.12). For simplicity, the hyper-parameter λ_1 is borrowed from the reconstruction loss term.

5.2.2 Perception Error Models

In this work we consider two different perception models. First, is the vision-based perception system. We define this model as the subsystem that takes in high dimensional sensor data from the environment, and produces low dimensional labels.

$$\hat{Y} = Per(Sen(Env)) \quad (5.1)$$

These labels can take the form of bounding box coordinates, depth estimations, or any other value that is informative for vehicle control. Here, $\hat{Y}$ is a random variable that represents an approximation of the "ground truth" label Y . The distribution of $\hat{Y}$ can be described in notation as $\hat{p}(Y)$. When the perception system was formed, some measure of divergence between $p(Y)$ and $\hat{p}(Y)$ was reduced. However, there is inevitably some amount of error X in our perception labels $\hat{Y}$.

$$\hat{Y} = Y - X \quad (5.2)$$

The error term, denoted once again by the random variable X from Section 5.2.1, provides an alternative method for producing perception labels $\hat{Y}$. With this method, we can replicate the behavior of the perception system within a virtual simulation, while also circumventing the domain shift error from real world sensors Sen , and real environments Env from Eq. (5.1). Instead, we are left only with the ground-truth labels Y , which have little to no domain shift error between the real world and virtual simulation. We note that the error term X is almost certainly going to be probabilistic in nature, due to the perception model $\hat{p}(Y)$ being trained on principles of regression that produce Gaussian residuals. In virtual simulation, the ground-truth label Y is readily available for any type of label. Given this fact, we can simply approximate the perception system in Eq. (5.1) by producing the distribution $\hat{p}(X)$, restated as $\hat{p}(X_{1:T})$ for real-valued time-series error signals, and applying Eq. (5.2). Conceptually, we coin this concept as a "model of a model", where a probabilistic error model $\hat{p}(X)$ is used to model the perception system $\hat{p}(Y)$.

5.3 Methodology

5.3.1 Perception Model & Dataset

In this section, we define the specifics of our perception model under test, and the methodology for producing the dataset required to train the perception error model outlined in Section 5.2.2. We start by analyzing the *supercombo* perception model. This model takes in as input consecutive RGB images that are captured at a framerate of 20 Hz. The *openpilot* perception stack also features a calibration system, such that the model is robust against slight changes in extrinsic orientation of the camera, as well as intrinsic changes to field-of-view and resolution. This calibration procedure was followed for all data using this perception model. There are two outputs of note from the *supercombo* model. First, a two-dimensional spline is produced that represents a prediction of the upcoming lane trajectory. This is the primary input provided to the lateral control subsystem that affects the steering of the vehicle. Second, the model produces *lead* predictions. This lead vehicle information provides an estimate of the speed of the lead vehicle, as well as the distance between the ego vehicle and the lead vehicle. This is the primary input of the longitudinal control subsystem of *openpilot* that affects throttle and braking. In our implementation, we will allow the lateral perception information to be produced visually in simulation, as described in Eq. (5.1). We will instead focus solely on the longitudinal lead information when forming our perception error model. The *openpilot* controller considers lead vehicle information by combining the ground truth distance to the lead vehicle $d_t^{(gt)}$ with a stopping distance for the lead vehicle, derived from $v_t^{(gt)}$, shown in Eq. (5.3). The estimated lead vehicle stopping distance $\hat{d}_t$ from the perception model is shown in Eq. (5.4).

$$d_t = d_t^{(gt)} + v_t^{(gt)^2} / a_{\max} \quad (5.3)$$

$$\hat{d}_t = d_t^{(p)} + v_t^{(p)^2} / a_{\max} \quad (5.4)$$

Here, the expected maximum deceleration rate of the lead vehicle $a_{\max}$ is set by the controller at a constant of 5m/s^2 for both Eq. (5.3) and Eq. (5.4). To produce $\hat{d}_t$, our goal

is to create a model of the two perception outputs $d_t^{(p)}$ and $v_t^{(p)}$, combine them using Eq. (5.4), and pass this value to the longitudinal *openpilot* controller.

To produce the model distribution of our PEM $\hat{p}(X_{1:T})$ described in Section 5.2.2, we must form training data that describe $p(\hat{Y})$ and $p(Y)$, from which the distribution $p(X)$ is formed from $X = Y - \hat{Y}$. For this purpose, we use the *comma2k19* dataset [134]. This dataset features video feed collected at 20 Hz for over two-thousand one minute long segments. The dataset also features baseline vehicle information such as speed and steering angle, as well as post-processed front-facing radar data. Ground-truth labels Y are produced using the radar sensor data. The radar provides both a distance measurement, and a measurement of the relative speed between the sensor and the lead vehicle. We also apply additional post-processing in the form of a one-dimensional discrete Kalman Filter to the radar signal in order to reduce small amounts of sensor noise and make the signal more similar to the distance and speed measurements that will be taken in simulation. We describe the ground-truth labels $Y_t = [b_t^{(gt)}, \text{KF}_t(d_t^{(gt)}), \text{KF}_t(v_t^{(gt)})]$. Here, $b_t^{(gt)} \in \{0, 1\}$ denotes whether or not a lead vehicle is present within ~ 200 meters in front of the ego vehicle at time t . $d_t^{(gt)}$ (m) and $v_t^{(gt,rel)}$ (m/s) denote the radar output at time t for distance and lead-vehicle speed respectively, while KF_t denotes the 1D Kalman filter operation along the time axis. To produce the absolute speed of the lead vehicle in our representation, we add the known speed of the vehicle $v_t^{(ego)}$ with the relative speed captured from radar $v_t^{(gt,rel)}$ at every time instance $v_t^{(gt)} = v_t^{(ego)} + v_t^{(gt,rel)}$.

Perception output is collected by simply passing the processed images from the *comma2k19* dataset and recording the output. A visual overlay of the model onto an example image from the dataset is shown in Fig. 5.1. Considering the longitudinal lead information from the model, we define $\hat{Y}_t = [b_t^{(p)}, d_t^{(p)}, v_t^{(p)}]$. In this case, $b_t^{(p)} \in \{0, 1\}$ denotes whether or not a lead vehicle was detected by the perception system at time t . $d_t^{(p)}$ (m) and $v_t^{(p)}$ (m/s) denote the model's estimation of the distance to the lead vehicle, and the model estimation of the lead vehicle speed, respectively. With Y and $\hat{Y}$ fully defined, we can now define the error term $X_t = [b_t^{(p)}, \Delta d_t, \Delta v_t]$. In our case, the presence of a lead vehicle detection $b_t^{(p)}$ is simply copied from the perception output $\hat{P}$ directly. Also, we define $\Delta d_t = \text{KF}_t(d_t^{(gt)}) - d_t^{(p)}$ and $\Delta v_t = \text{KF}_t(v_t^{(gt)}) - v_t^{(p)}$ as the error terms for the distance to the lead vehicle and the absolute velocity of the lead vehicle, respectively. Making use of additional information, we define the supplementary information $S_t = [c, v_t^{(ego)}, \text{KF}_t(d_t^{(gt)}), \text{KF}_t(v_t^{(gt)})]$. We denote c as the 24-hour clock time corresponding to $t = 0$. $c \in [0, 1]$ such that 0 corresponds to the local time 00 : 00,

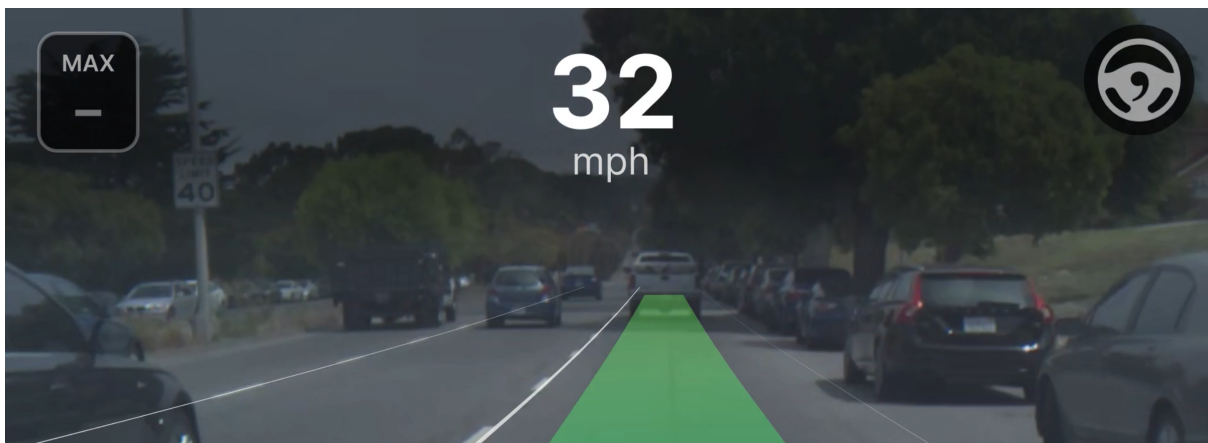


Figure 5.1: An image from the *Comma2k19* dataset. A graphical interpretation of the model's output is shown as an overlay, demonstrating the procedure for producing our dataset.

and 1 corresponds to the local time of 23 : 59. The target distribution $p(X_t \mid S_t, X_{t-i})$ is now represented by our dataset $\mathcal{D} = \{(x, s)_{n,1:T}\}_{n=1}^N$. With all variables fully defined, we can now refer back to our objective. The PEM model described in Section 5.4 produces the approximate distribution $\hat{p}(X_{1:T})$, such that $p(X_t \mid S_t, X_{t-i})$ for $t \leq T$ and $t - i \geq 0$ is absorbed into $p(X_{1:T})$. Synthetically produced time-series error signals from our PEM are denoted as $\hat{X}_{0:T}$.

5.3.2 Simulation Strategies

The simulation strategy in this work follows the simulation strategies of Section 3.2 closely. Once again, we propose virtual simulation using a vehicle-in-the-loop (VIL) strategy that is supplemented with software-in-the-loop (SIL) and model-in-the-loop (MIL) approaches. In our SIL strategy, the vehicle dynamics are modeled in CARLA, and the CAN bus communication required for our implementation of *openpilot* to function is simulated in software. In VIL simulation, virtual simulation is applied on a real vehicle. The dynamic response of the vehicle is measured through the CAN bus communication, and the position of the vehicle in simulation is updated using the bicycle model of Eqs. (3.1) to (3.3). VIL simulation is performed using a steering-enabled dynamometer, shown in Fig. 3.2. To measure the output of the PEM model in MIL simulation we do not need simulated image input, as is described in Section 3.2. Instead, we only need to provide a predefined conditional vector $S_{0:T}$, and a random sample from the encoding space p_z . These requirements are straightforward to produce, and allow for a much higher number of MIL experiments to be performed in comparison to both of the closed-loop VIL and SIL simulation strategies.

In all of our simulations, lateral control is always handled from visual information using the paradigm described in Eq. (5.1). Camera images are taken at each time instance, using the detected lanes to inform the *openpilot* controller. For perception related to longitudinal control, we consider three types perception strategies, applicable to both VIL and SIL simulation approaches:

- **Visual VIS:** Lead information from the model is produced visually, as intended in the Vision Only Adaptive Cruise Control (VOACC) functionality within *openpilot*. Perception is conducted strictly according to Eq. (5.1), and visual domain shift error is present. In this case, we denote the error term $X_t^{(sim)}$. This additional notation is done to highlight the possibility that domain shift error may exist in $p(X_{1:T}^{(sim)})$, such that there is substantial deviation from $p(X_{1:T})$. All vision-based experimental runs were conducted with the **Clear**

conditions, described in Section 3.3. We also note that the intrinsic values and calibrations of the camera were modified in this work. While we used default camera properties for the controller in Chapter 3, we modified these properties in this work to more closely match the camera intrinsics used to produce the *Comma2k19* dataset [134].

- **Ground Truth GT:** Here, $X_t = [1, 0, 0]$ for all of t . This represents a best-case scenario with respect to the perception system, where there is no error present at all. With this approach, any dangerous behavior or variance in vehicle response can be confidently attributed to the control strategy or hardware related latency.
- **Perception Error Model PEM:** Most noteworthy, our PEM strategy provides the probabilistic model of our perception system under test. During experimental inference, the error term $\hat{X}_t = [\hat{b}_t^{(p)}, \hat{\Delta d}_t, \hat{\Delta v}_t]$ is the output from our trained model. We also provide as input to our PEM model the supplementary vector $S_t = [c, v_t^{(gt)}, d_t^{(gt)}, v_t^{(ego)}]$. Here, c is defined as part of the experimental setup, while $v_t^{(gt)}$, $d_t^{(gt)}$, and $v_t^{(ego)}$ are measured directly within the CARLA simulator. The error term is applied to the ground-truth label according to Eq. (5.2), and the resulting detection information is passed to the controller. In the case of our model predicting a missed detection, $\hat{\Delta d}_t$ and $\hat{\Delta v}_t$ are disregarded, and a flag is passed to the longitudinal controller indicating that no lead vehicle is present.

5.3.3 Experimental Setup

The purpose of the experiments in this work is to test the adaptive cruise control behavior of *openpilot* while highlighting the benefits of our PEM methodology. Because lateral perception and lane-keeping are accomplished through visual cues, we don't feature experiments that test these systems for failure. Instead, we focus on longitudinal scenarios that will give us a more complete understanding of how the perception system affects throttle, braking, and dynamic response. For this purpose, all experiments are conducted on a straight road leading up to a stop, shown in Fig. 5.2. For the VIL experiments, we once again used the Stopping experiments defined Section 3.3, where a lead vehicle is stationary in the road, while the ego vehicle approaches from behind. This testing procedure is simple to implement, and provides a straightforward experimental setup for comparing the perception and response of our three different types of perception systems, defined in Section 5.3.2.

For the SIL experiments, we broadened the scope of the experiments to analyze the behavior of our PEM and the *openpilot* response to stopping conditions with a moving lead vehicle. The first testing parameter in these experiments is the initial speed of the lead vehicle, which is equivalent to the ego vehicle $v_{t=0}^{(ego)} = v_{t=0}$. The second parameter defining our experimental setup is the following time FT. The initial lead distance between the vehicles is inferred from $d_{t=0}^{(gt)} = v_{t=0}^{(ego)} * FT_{t=0}$. We test the behavior of the ego vehicle with PEM-based perception under the conditions $v_{t=0} = \{20\text{mph}, 30\text{mph}, 40\text{mph}\}$. Additionally, we cover three different following times, ranging from very aggressive to conservative $FT_{t=0} = \{1.5\text{s}, 2.5\text{s}, 3.5\text{s}\}$. In all SIL experiments, we decelerate the car at a constant rate of $a_{\text{stopping}} = -3\text{m/s}^2$. The full experimental setup for the SIL experiments is visualized in Fig. 5.2.

While vision-based experiments were only done with the VIL Stopping experiments, ground truth runs were conducted for all experiments for both VIL and SIL strategies. To produce the PEM perception output for the MIL experiments, the MIL conditional vector $S_{0:T}^{(MIL)}$ was formed from simply measuring the ego and lead vehicle response from the first run $n = 1$ of each experiment using ground truth perception. For all MIL experiments, time of day was set to a random value, chosen uniformly between $c = [0, 1]$. Combining the dynamic response measurements from the VIL and SIL experiments with the rich sampling of the encoding space defined by p_z in the MIL experiments provides a thorough platform for understanding the ego vehicle’s longitudinal safety performance across a representative spectrum of conditions.

5.4 Results

In this section, we highlight the results and impact of our trained perception error model. To do this, we use two distinct approaches. First, the *accuracy* and *realism* of the model is demonstrated by comparing our synthetic error data $\hat{X}_{0:T}$ directly against the real error dataset $\mathcal{D}$, where we holistically compare the datasets against qualitative visualizations and quantitative metrics. In this way, we demonstrate that the model is at least realistic enough that useful conclusions can be drawn from its implementation. Second, we utilize our PEM model within *openpilot* to conduct vehicle simulation experiments, making note of key performance and safety implications of the vehicle’s response.

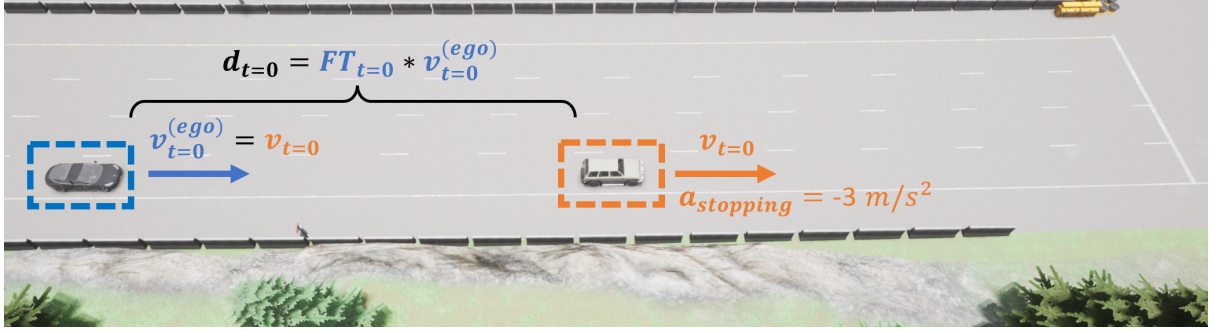


Figure 5.2: The CARLA setting for the SIL experiments is shown. The ego vehicle is shown in the blue bounding box. The lead vehicle is shown in the orange bounding box. While controlling for $a_{\text{stopping}} = -3 \text{ m/s}^2$, the initial speed of the stopping lead vehicle $v_{t=0}$ and the starting lead distance $d_{t=0} = FT_{t=0} * v_{t=0}^{(ego)}$ are varied.

5.4.1 Training Dynamics

As defined in Chapter 4, the hyperparameters of the *FETSGAN* model are λ_1 , λ_2 , and ϵ , as well as the dimensionality of p_z and p_η . In our trained PEM model, we used $\lambda_1 = 0.05$, $\lambda_2 = 1$, and the error limit for the first above threshold (FAT) operation was chosen to be $\epsilon = 50$. Note that in Chapter 4, all values in each time-series dataset were normalized in the range $[-1, 1]$. Here, we did not scale the error terms Δd_t and Δv_t . This primarily affects reconstruction loss and the FAT operation, where λ_1 and ϵ were adjusted accordingly. The dimensionality of the intermediate encodings p_z was 20 dimensions, and the uniform random noise vector p_η was 5 dimensions. All sub-components of the *FETSGAN* model were trained with the Adam optimization strategy [130], where the learning rate for the generator, encoder and both discriminators is 0.0001. Training occurred for 2000 epochs, where final model tuning occurred with learning rates decaying exponentially in the last 10% of epochs. The dataset $\mathcal{D}$ contained 2200, ten second sequences. Due to some concern about the model over-fitting this relatively small dataset, a 5% dropout was added to the generator's hidden layers during training. Additionally, a small amount of Gaussian noise $\mathcal{N}(0, 0.01)$ was added to all dimensions of the conditional vector s_t during training.

5.4.2 Model Validation

Before analyzing our PEM in the context of closed-loop vehicle control, we first must verify its accuracy. To do this, we examine the structure of the temporal signals themselves and visually ensure the synthetic PEM data matches the training dataset, while discussing the implications of the signals themselves. Once we believe the temporal profile is captured, we flatten the temporal dimension of the PEM and perform statistical analysis on the dataset as a whole, further ensuring it is a good fit of the real error data $\mathcal{D}$.

Temporal Perception Profile

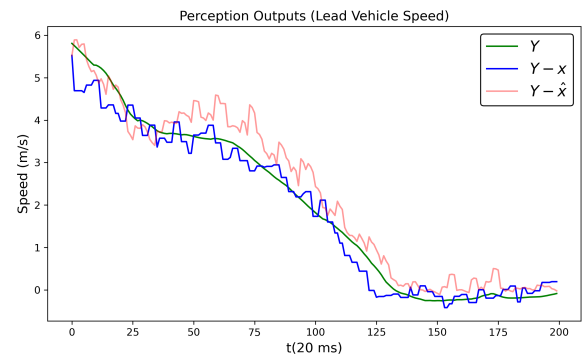
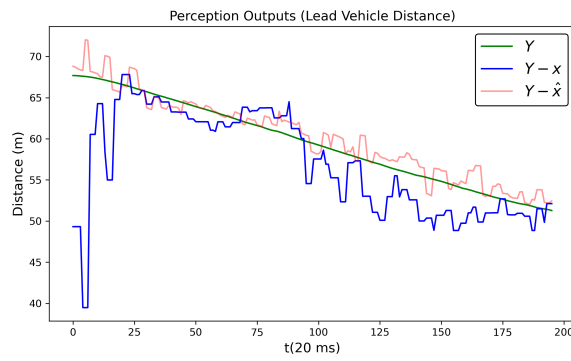
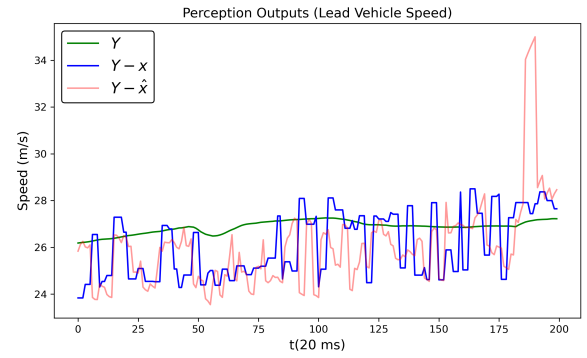
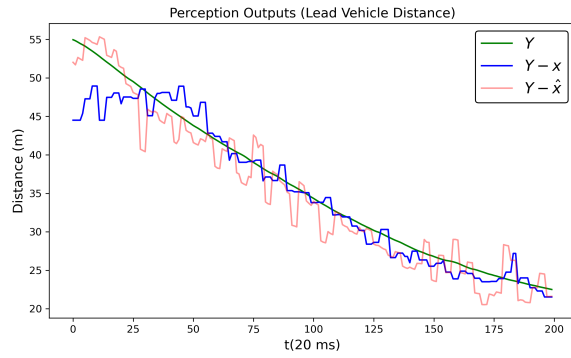
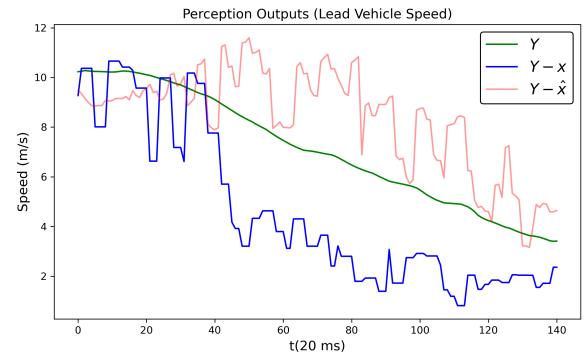
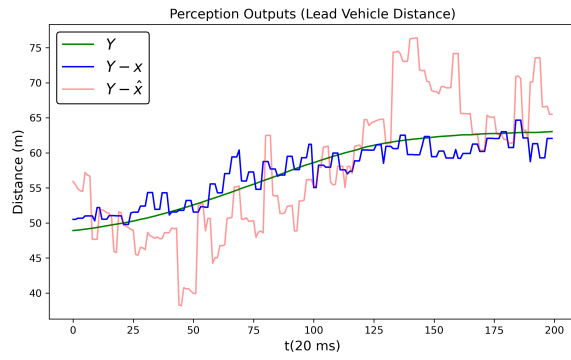
Post-training, we produced a synthetic dataset $\hat{\mathcal{D}}$. This dataset was formed by using the same conditional vectors S_t from the training set $\mathcal{D}$, but encoding vectors randomly sampled from p_z , following the inference process of Chapter 4. In this way, we can straightforwardly compare the output of our model with the real perception output, with the size of both datasets being identical, containing 2200 time-series signals. In Fig. 5.3, we show the raw output of our PEM

based perception system, denoted by $y - \hat{x}$ on real-data series from our *comma2k19* dataset $\mathcal{D}$. We compare this output to the actual perception output $y - x$, as well as the ground-truth radar data shown as y . Denoting raw model output for just a handful of data samples is not a quantitative exercise, but it can provide a valuable understanding of what the target distribution $p(X_{1:T})$ looks like, and a baseline gauge for how well $\hat{p}(X_{1:T})$ fits that distribution. We also provide this visual to demonstrate that the temporal noise profile of $p(X_{1:T})$ is quite complex. To verify that the sequences in $\mathcal{D}$ were sufficiently distinct from Gaussian noise, auto-regressive temporal analysis was performed. Pearson correlation was conducted between $X_{0+0.25s:T}$ and a signal lagging by 250ms $X_{0:T-0.25s}$. 95.4% of the signals in $\mathcal{D}$ were determined to show statistically significant correlation, with $p < 0.05$. Similarly, 93.8% of the samples in $\hat{\mathcal{D}}$ showed statistically significant temporal correlation. Less accurate temporal models described in Section 1.2.4, as well as PEM models that simply try to approximate $\hat{p}(X_{1:T})$ with Gaussian sampling, will clearly fail to fully capture this temporal profile. It is plainly obvious in Fig. 5.3 that error can be quite significant with respect to the ground truth, and properly modeling this error is key to providing realistic vehicle response with a PEM model.

Quantitative Analysis

Because the temporal distribution of $X_{1:T}$ is complex and undefined, it is not possible to provide conclusive metrics indicating that the PEM producing $\hat{X}_{0:T}$ matches it. However, We can apply some statistical exercises to the data to gain further confidence beyond pure visualization that the datasets sufficiently match. Toward that effort, we can also gain some insights into the characteristics of the dataset $\mathcal{D}$, beyond the temporal characteristics highlighted in Fig. 5.3. Here, the temporal dimensions of $\mathcal{D}$ and $\hat{\mathcal{D}}$ were flattened, and we performed statistical analysis only on the feature dimensions.

The quantitative metrics used to compare the real and synthetic datasets are shown in Table 5.1. First, we used Pearson Correlation to determine the linear relationship between the absolute error terms of $|x_{1:T}|$, and the terms of the conditional vector s_t . Instead of using clock-time, we modified c to reflect the lighting conditions. $c_{\text{light}} = \sin(\pi c)$, where 12 : 00 would correspond to a value of 1, while 00 : 00 and 24 : 00 would correspond to 0. Analyzing the relationships in our real dataset $\mathcal{D}$, we found that there were a handful of weak to moderate correlations between these variables. Most notably, lead vehicle distance d predicts the absolute



(a) Lead Distance (raw output).

(b) Lead Velocity (raw output).

Figure 5.3: Comparison of Lead Distance and Lead Velocity (raw model outputs) across time

error terms $|\Delta d|$ and $|\Delta v|$ at a higher margin than any other relationship. Meanwhile, it is also noteworthy that lighting appears to have little to no influence on our perception model output at all. While the correlation values of r may be moderate to low, we can see that our synthetic dataset captures the relationships, or lack thereof, that exist between these variables. We also looked at model dropout rate as a predicted response to lighting conditions. While this did seem to be significant at large distances ($d_t^{(gt)} > 100\text{m}$), at the operational range of our experimental setup ($d_t^{(gt)} < 60\text{m}$), dropout was not a large factor with light conditions playing a minimal role. Nonetheless, the synthetic data reflected the model’s performance with respect to dropout accurately.

In Fig. 5.3, we can clearly see visually that the error profile that defines the perception system according to Eq. (5.2) is not Gaussian in nature on the temporal dimension. However, other work in perception error modeling such as [96] make a Gaussian error assumption by producing error terms using Hidden Markov Models (HMM). While this assumption may not always be valid, it does provide guarantees that the output of their PEM fits the real distribution, given a Gaussian definition. The GAN framework used in this work provides no such guarantees, so it is a useful exercise to analyze the datasets $\mathcal{D}$ and $\hat{\mathcal{D}}$ under an assumption of Gaussian error, and observe how well our PEM model fits the real distribution. This analysis is shown under the **Gaussian Error** section of Table 5.1. Given that error correlates most strongly with distance between the ego vehicle and the lead vehicle, we defined error windows for both model outputs Δd and Δv with respect to $d_t^{(gt)}$. The windows were kept in small 10m increments, given the experimental setup of the next section. The magnitude of the expected errors, indicated by the standard deviation SD, generally increases with distance as expected. We can see that these Gaussian relationships are mostly maintained in the *PEM* model, with little deviation.

In this work, we are not concerned with the statistical difference between the real and synthetic datasets, given that we know it is highly likely for some difference to exist between them that is beyond random chance. We are more interested in the *practical difference* between them, which can be measured by finding the *effect size* between the two distributions. The effect size is indicated most commonly by Cohen’s distance [136]:

$$d_{\text{cohen}} = \frac{\mu_{\text{real}} - \mu_{\text{PEM}}}{\sqrt{\frac{1}{2}(\text{SD}_{\text{real}}^2 + \text{SD}_{\text{PEM}}^2)}} \quad (5.5)$$

where μ is the mean and SD is the standard deviation. Cohen’s distance can be interpreted as a measurement of similarity between two distributions that is inversely proportional to the amount of overlap those distributions have. In our case, μ_{real} , μ_{PEM} , SD_{real} , and SD_{PEM} were taken for all $d_t^{(gt)} < 50\text{m}$. We found $d_{\text{cohen}} = 0.008$ for detected distance error Δd , and $d_{\text{cohen}} = 0.004$ for detected lead speed error Δv , which corresponds to 99.55% and 99.77% overlap between the real and synthetic distributions for detected distance error and detected lead speed error, respectively. These are extremely small effect sizes. While interpreting Cohen’s distance is application dependent, most values under 0.2 are considered minimal [136]. This indicates that even when placing Gaussian assumptions on our PEM model, we still match the original distribution very accurately.

5.4.3 Vehicle Response

Through VIL simulation in Section 5.4.3, we can get a baseline understanding of how perception error relates to the physical vehicle’s response. This allows us to view a more complete picture of the variance in vehicle response provided by the PEM. Additionally, we are able to see how this response relates to the straightforward implementation of the visual perception system in simulation, as well as the ground truth perception approach. Ultimately, we are able to quantify how the vehicle responds with respect to the minimum time to collision ttc_{min} , and correlate this value against the detection ratio dr_{mean} , as described in Section 3.4. However, the formulation of our PEM conditional vectors $S_{0:T}$ feature the relative distance $d_{0:T}^{(gt)}$, speed of the ego vehicle $v_{0:T}^{(ego)}$, the lead vehicle speed $v_{0:T}^{(gt)}$. While the clock time c was not highlighted for these experiments due to the lack of influence at short distances described in Section 5.4.2, sufficiently testing the other parameters of $S_{0:T}$ is critical to getting a complete understanding of vehicle behavior under similar stopping conditions. The experiments conducted under SIL conditions in Section 5.4.3 feature a similar adaptive cruise control stopping maneuver, with varying lead and ego speeds between 20-40 mph, as well as following distances ranging from a conservative 3.5s to an aggressive 1.5s. While direct

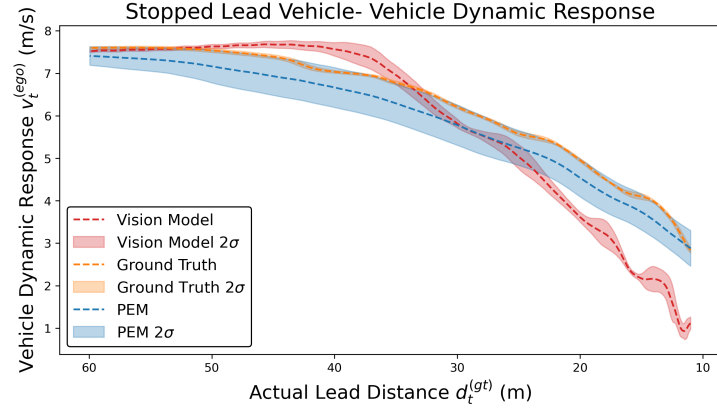
Table 5.1: Quantitative Comparison: Real $\mathcal{D}$ vs. PEM $\hat{\mathcal{D}}$

Pearson Correlation (r)					
Model Output x_t	Conditional s_t	Real		PEM	
Distance Error Δd	Lighting c_{light}	-0.020		0.036	
	Ego Speed $v_t^{(\text{ego})}$	0.163		0.159	
	Lead Distance $d_t^{(gt)}$	0.415		0.426	
	Lead Speed $v_t^{(gt)}$	0.210		0.207	
Lead Speed Error Δv	Lighting c_{light}	-0.083		-0.062	
	Ego Speed $v_t^{(\text{ego})}$	0.169		0.188	
	Lead Distance $d_t^{(gt)}$	0.349		0.343	
	Lead Speed $v_t^{(gt)}$	0.328		0.358	
Lighting-Dropout Percentage					
Lighting Cond.		Real		PEM	
Night		1.49%		1.68%	
Dawn/Dusk		1.07%		1.08%	
Midday		1.21%		1.14%	
Gaussian Error					
Model Output x_t	Lead Distance $d_t^{(gt)}$	Real		PEM	
		μ	SD	μ	SD
Distance Error Δd $d_{\text{cohen}} = 0.008$	$d_t^{(gt)} < 10$ m	-0.41	2.09	-0.38	1.87
	$10\text{m} \leq d_t^{(gt)} \leq 20\text{m}$	-0.75	3.07	-0.78	3.24
	$20\text{m} \leq d_t^{(gt)} \leq 30\text{m}$	-0.67	3.36	-0.54	3.62
	$30\text{m} \leq d_t^{(gt)} \leq 40\text{m}$	-0.16	3.59	-0.1	3.39
	$d_t^{(gt)} > 40\text{m}$	1.36	6.6	1.33	6.65
Lead Speed Error Δv $d_{\text{cohen}} = 0.004$	$d_t^{(gt)} < 10$ m	-0.02	0.53	-0.04	0.52
	$10\text{m} \leq d_t^{(gt)} \leq 20\text{m}$	0.43	1.04	0.4	0.98
	$20\text{m} \leq d_t^{(gt)} \leq 30\text{m}$	0.62	1.43	0.72	1.43
	$30\text{m} \leq d_t^{(gt)} \leq 40\text{m}$	0.75	1.77	0.74	1.68
	$d_t^{(gt)} > 40\text{m}$	1.44	2.47	1.42	2.32

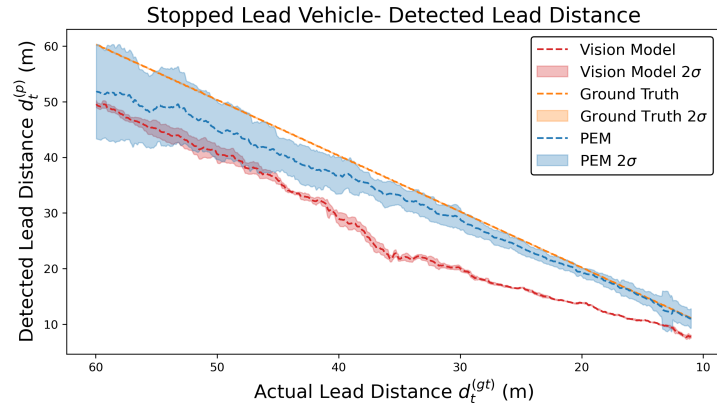
inferences on physical vehicle behavior cannot be made from the time to collision values of the SIL experiments, we can still get some insight on the vehicle behavior of each experimental condition relative to the others, and the SIL response of the ground truth perception output. For both experimental types, MIL simulations were performed. One of the major axes of our measured results is the percentage of runs that feature potentially unsafe behavior, or perception error that could induce unsafe behavior. Given that these are relatively uncommon events, we deploy MIL simulation in order to produce a large number of samples to shed light on this information. Using the strong negative correlation between the minimum time to collision and the mean detection ratio that was discussed in Section 3.4, we use the MIL simulations of our PEM model to make statistically founded predictions regarding what kind of safety implications may exist for similar types of driving in a real-world setting.

VII Analysis- Stopped Lead

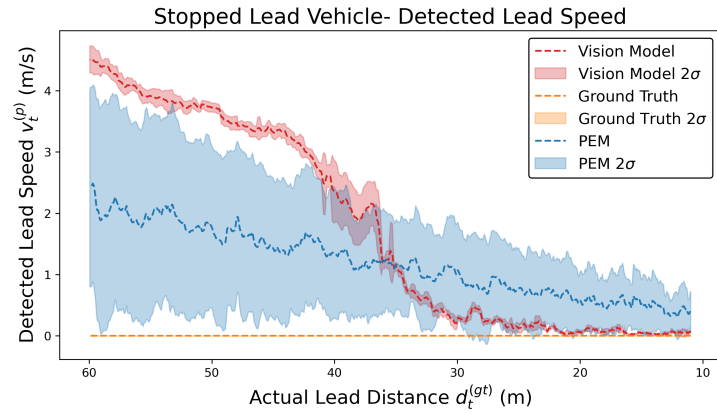
In Fig. 5.4, we show response and perception output for the vehicle-in-the-loop setup, reusing the experimental setup from the **Stopping** experiments of Section 3.3.1. Experiments were run using the PEM model, corresponding to $y - \hat{x}$, as well as the native visual model, corresponding to $y - x^{(sim)}$, where $x^{(sim)}$ is the error term of the *supercombo* visual perception system accounting for potential domain shift error. We also include the dynamic response to ground truth information, corresponding to $X_t = [1, 0, 0]$. In Fig. 5.4a, twenty samples $N = 20$ are taken for our PEM model, while $N = 5$ for our Vision approach, and $N = 5$ with our ground truth approach. Two standard deviations ($\mu - \sigma, \mu + \sigma$) are plotted along with the mean to demonstrate the distribution of responses. While the ground-truth perception is virtually deterministic, with variation only coming from the introduction of vehicle hardware, we can see from Fig. 5.4a that the PEM produces significantly more deviation from than repeated runs using the visual perception system. The mean temporal standard deviation of ego speed $v_t^{(ego)}$ across all samples of the PEM experiments is roughly 3.5 times larger than that of the vision experiments. We could theoretically bring these distributions in line by testing a more complete set of visual conditions, but this highlights the purpose of the PEM as it is formulated: to produce the full distribution of perception conditions that can lead to the full distribution of possible dynamic responses. We can also see that the visual response is quite different from the PEM response, even within two standard deviations. The visual perception system causes the vehicle to reach higher speeds, but also brake more sharply. For diagnosis, we can



(a)



(b)



(c)

Figure 5.4: Response and perception are shown for the **Stopping** experiments of Section 3.3.1

look to Fig. 5.4b and Fig. 5.4c. There, we see that the visual perception system generally underestimates the distance to the lead vehicle, while overestimating the speed of the lead vehicle, which is 0 for the entire run. This suggests that running experiments visually may provide poor data for extrapolation. Even within two standard deviations, the domain shift error of $X_t^{(sim)}$ could be great enough that the dynamic response is no longer informative of the performance of the same perception system on real-world data, further highlighting the utility of the PEM approach.

In Fig. 5.5, we examine the performance metrics for the VIL experiments. We use kernel density estimates to get an approximated visualization for the distribution of each metric [137]. For ground truth runs, the mean value for each metric is shown with a vertical line. For ttc_{min} in Fig. 5.5a, we see that once again the distribution for the visual experimental runs is much more collapsed than the PEM. The average ttc_{min} is also much more conservative than the PEM generally, while the PEM is mostly centered around the ground truth response, with an average ttc_{min} of about 2.5 seconds. We do note that one of the PEM runs resulted in a collision with the lead vehicle, representing a 5% collision rate in our somewhat limited sample size. To gain further insight into the vehicle response, we can make use of the procedure defined in Section 3.4, by examining a *predictive* metric for our vehicle response. In this case, we examine the detection ratio dr_{mean} . As expected, the mean detection ratio is generally higher than the visual response in Fig. 5.5b. Plotting dr_{mean} against ttc_{min} in Fig. 5.5c, we see that the linear relationship between detection ratio and time-to-collision generally holds, as it did in Section 3.4. Defining the Pearson correlation between the detection ratio dr_{mean} and the time-to-collision ttc_{min} of each run as r_{dr} , we measure that $r_{dr} = -0.93$ for the PEM experiments. The one collision reported in our VIL experimental runs had $dr_{mean} > 1.2$, affirming that threshold as a reasonable predictor for dangerous vehicle response and imminent collisions. We can further leverage this relationship to infer the VIL response from the detection ratio in our MIL experiments. In 100,000 MIL experiments using the first ground truth vehicle response $n = 1$ to form $S_{0:T}^{(MIL)}$, we saw 7.3% of runs end with $dr_{mean} > 1.1$, indicating performance that could produce borderline dangerous behavior. Meanwhile 2.1% of runs featured $dr_{mean} > 1.2$, indicating a high possibility that the perception system could produce imminent danger. Both the VIL and MIL experimental metrics for the Stopping experiments are shown in Table 5.2.

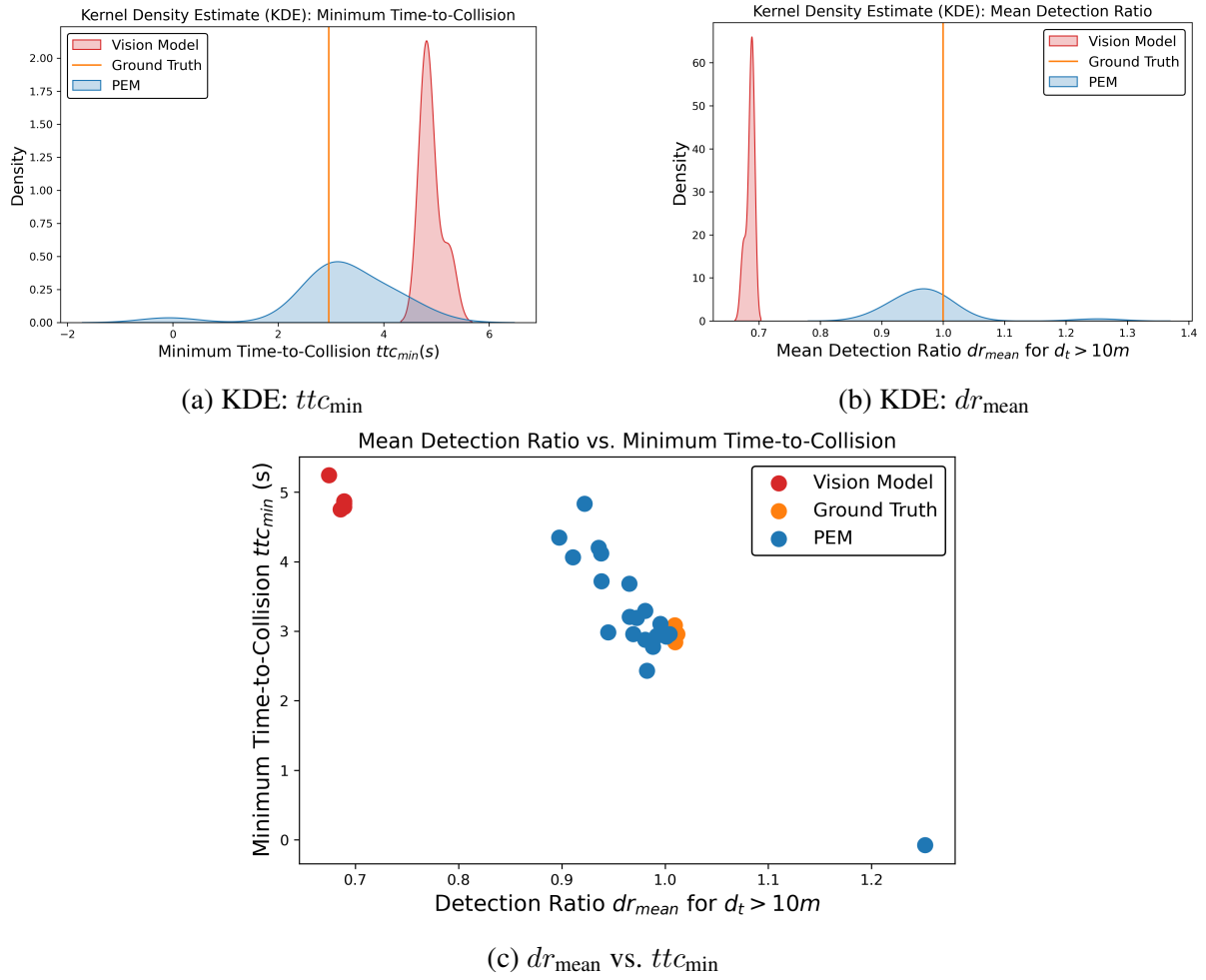


Figure 5.5: Predictive and Performance metrics are analyzed for VIL experiments for all three experiment types.

Table 5.2: Predictive & Performance metrics for VIL, SIL, and MIL experiments

Stopping (VIL and MIL)									
Perception Type		ttc _{min}				dr _{mean}			
		μ	SD	r _{dr}		μ	SD	% > 1.1	% > 1.2
VIS (N=5)		4.9	0.18	-0.91		0.69	0.01	0.0%	0.0%
GT (N=5)		2.96	0.08	-0.04		1.00	N/A	0.0%	0.0%
PEM (N=20)		3.23	0.97	-0.93		0.98	0.07	5.0%	5.0%
(MIL) (N=100,000)						(1.01)	(0.08)	(7.3%)	(2.1%)
Moving Lead (SIL and MIL)									
Driving Type		SIL				MIL			
		GT (N=5)	PEM (N=50)			PEM (N= 100,000)			
		ttc _{min}	ttc _{min}			dr _{mean}			
Lead Speed	FT	μ	μ	SD	r _{dr}	μ	SD	% > 1.1	% > 1.2
20 mph	1.5s	2.37	2.64	0.72	-0.61	0.98	0.12	9.0%	5.8%
	2.5s	3.19	3.42	0.99	-0.85	1.00	0.15	12.7%	7.8%
	3.5s	3.68	3.99	0.48	-0.91	1.01	0.11	8.2%	3.4%
30 mph	1.5s	2.61	2.80	1.21	-0.89	1.00	0.12	11.2%	7.5%
	2.5s	3.44	3.82	0.54	-0.92	0.99	0.08	8.2%	2.7%
	3.5s	3.86	4.11	0.43	-0.84	0.99	0.06	3.3%	0.3%
40 mph	1.5s	2.36	2.10	1.03	-0.79	1.01	0.13	13.4%	8.9%
	2.5s	3.26	3.29	0.51	-0.93	1.00	0.07	8.2%	0.6%
	3.5s	3.66	3.17	0.30	-0.91	1.00	0.05	2.4%	0.1%

SIL Analysis- Moving Lead

The performance metrics for our SIL experiments are shown along side the predictive metrics of our MIL experiments in Table 5.2. The first thing we can highlight is the ttc_{min} values across the spectrum of conditions. As expected, minimum time to collision generally decreases with the following time. Additionally, ground truth perception is generally lower for the lower speed scenarios of 20mph and 30mph. However, this relationship reversed at the highest speed scenario of 40mph. *Openpilot* uses detected stopping distance $\hat{d}_t$ to make all longitudinal control decisions while in adaptive cruise control. We can see from Eq. (5.4) that there is a non-linear relationship between detected stopping distance $\hat{d}_t$ and detected lead speed $v_t^{(p)}$. We can see from Fig. 5.4 that there appears to be a tendency for the perception model's to overestimate the lead vehicle speed in stopping conditions, which may lead to more aggressive stopping behavior at higher speeds. This speaks to the need to test at a variety of conditions for more complete insight into the implications of perception model error. In terms of vehicle response, we also once again see that the correlation between dr_{mean} and ttc_{min} is generally strong as well. The SIL controller has a tendency to be more aggressive than the VIL controller, with more ability to make sudden braking late in a maneuver that could explain why the correlation r_{dr} is slightly weaker in the SIL experiments.

Vehicle response in SIL simulation may give some context to expected safety performance relative to other conditions in the experimental setup, but the performance metrics garnered from CARLA-based physics may be too unrealistic to make strong statements regarding what the performance of a physical vehicle on the road would be under similar circumstances. To make further inferences for the behavior of a physical vehicle, we examine the MIL predictive metric under the assumption that the strong negative correlation of r_{dr} will continue to hold. For all MIL simulations under SIL experimental conditions, 8.2% of runs had $dr_{mean} > 1.1$, indicating behavior that could be approaching danger. Meanwhile, 4.1% of runs had $dr_{mean} > 1.2$, which could indicate imminent collision. Breaking down the MIL results by experiment, we see that at low speeds, the number of potentially dangerous runs is generally high across the board. These experiments with lower speeds correspond to late detections, with shorter initial lead distances $d_{t=0}^{(gt)}$. This could indicate that in the fairly rare scenario of detecting the lead vehicle late ($d_{t=0}^{(gt)} < 60m$), perception becomes more erratic. For the other experiments, we see that the percentage of dangerous runs is low for moderate to conservative following times. However, as following time becomes aggressive, we see a spike in the percentage of

MIL simulations overestimating the stopping distance. Putting all of the available information together, this could indicate a dangerous combination. Time to collision is naturally lower at high speeds with a shorter following time. That fact is generally independent of the observation that our perception model has a tendency to overestimate stopping distance at a higher rate for these speeds. Combining these two observations represents a strong indicator that we have found a potentially hazard set of scenarios for the *openpilot* controller, further highlighting the utility in our methodology.

5.5 Summary & Limitations

In this work, we outline a GAN approach to perception error modeling. We use the deep learning architecture defined in Chapter 4 to produce a model of perception error for the open source SAE level 2 *openpilot* controller. Our PEM is demonstrated to effectively match the distribution of the original error data both temporally and as a temporally flattened static dataset. We applied this PEM model in both VIL and SIL simulation modalities in order to measure the downstream effect of the modeled perception system under closed-loop vehicle control. Additionally, we were able to leverage the strong correlation between vehicle response and perception model output to utilize MIL simulation as well. This allowed us to measure the presence of fairly uncommon failure scenarios at an extremely high number of samples, while still being able to make statistically justified inferences regarding how common those scenarios would be for a real-world driving.

There are limitations in this work that prohibit our ability to make broader assertions regarding the conclusions provided in our results. As it is formulated in this work, the perception error model can only provide a broad view at the performance of the perception system under test. To use our methodology for more rigorous edge-case detection, a richer dataset with more dimensions to the conditional vector S_t would provide additional insight into what causes the perception system to produce error. These perception conditions could then be delineated from each other in an experimental approach. Similarly, we are limited to testing the adaptive cruise control subsystem of *openpilot*. Producing a more thorough dataset with a more complex perception model and controller would allow this approach to be applied to more complex automated driving scenarios. Addressing these limitations constitutes a goal for future data collection and research.

Chapter 6

Conclusion

6.1 Original Contributions

This dissertation makes several important contributions toward the field of safety validation and testing of autonomous vehicles by leveraging advanced simulation techniques. The key contributions can be summarized as follows:

1. A comprehensive framework for analyzing autonomous vehicle sensor data and reconstructing accident scenarios was developed. This work fills a critical research gap by providing accident investigators with richer data and context to understand autonomous vehicle crashes, going beyond basic vehicle information or witness testimony. The proposed data pipeline and visualization techniques enable a more thorough analysis of pre-crash conditions and vehicle responses.
2. A novel vehicle-in-the-loop (VIL) simulation platform using a custom dynamometer setup that allows for highly realistic testing of autonomous vehicle systems was developed. This platform provides a safe environment to explore edge cases and failure modes that would be difficult or dangerous to test on public roads. A quantitative comparison between VIL, software-in-the-loop (SIL), and model-in-the-loop (MIL) simulations demonstrated the increased fidelity and value of a multi-simulation approach for edge-case discovery.
3. A new methodology for diagnosing and predicting autonomous vehicle behavior by analyzing the outputs of the *openpilot* perception system in relation to vehicle response was formed. This work established clear causal links between perception errors

and safety-critical vehicle behaviors, providing valuable insights and an approach for improving autonomous driving systems.

4. The *Fully Embedded Time-Series Generative Adversarial Network (FETSGAN)* architecture was introduced. This novel architecture can be used for generating high-quality synthetic time-series data. The approach outperformed existing methods in capturing complex temporal relationships and preventing mode collapse, with applications beyond autonomous vehicles to fields like medicine, energy management, and sensor simulation.
5. The *FETSGAN* architecture was used to create a *Perception Error Model (PEMs)* for simulating autonomous vehicle perception systems. This approach allows for more realistic simulation of perception errors without the need to fully replicate complex sensor data, bridging the gap between simulated and real-world testing.

6.2 Impact

As vehicles with autonomous capabilities continue to be developed, the work described here can further impact the testing and validation procedures for these vehicles. Complex methodologies are required to provide thorough safety assurances, and we have outlined approaches that can utilize virtual simulation to address this difficult problem. Additionally, the original model architecture of *FETSGAN* and the specific formulation of our perception error model have the potential to be further utilized in other fields beyond autonomous vehicle validation.

6.3 Limitations & Future Work

To expand on this work in the future, more effort should be spent on the realism of the approaches outlined, and the specifics of how these methodologies can be applied to real vehicles on the road. With realism gaps, it is difficult to make definitive statements regarding the results of edge-case testing. While we can speak to the methodology that produced results, it is imperative that the results from simulation are realistic enough that we can begin to speak on what real-world edge-cases actually are, and not just the methodologies that can discover them. Another opportunity for future work is to further expand this approach to vehicle systems being implemented on real-vehicles. While we focused primarily on the *openpilot* controller, there is

much work to be done to create a standardized approach for testing the self-driving capabilities of many different vehicle make's and model's. This will further allow researchers to take these ideas from the academic space and put practical information into the hands of consumers, allowing for a higher level of trust as we continue to develop these exciting technologies.

Bibliography

- [1] “The Grand Challenge.” <https://www.darpa.mil/about-us/timeline/-grand-challenge-for-autonomous-vehicles>. 1
- [2] C. B. J.D, “93% Have Concerns About Self-Driving Cars According to New Forbes Legal Survey.” <https://www.forbes.com/advisor/legal/auto-accident/perception-of-self-driving-cars/>, Feb. 2024. 1
- [3] A. Shariff, J.-F. Bonnefon, and I. Rahwan, “How safe is safe enough? Psychological mechanisms underlying extreme safety demands for self-driving cars,” *Transportation Research Part C: Emerging Technologies*, vol. 126, p. 103069, May 2021. 1
- [4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks,” *Commun. ACM*, vol. 63, pp. 139–144, Oct. 2020. 2, 8
- [5] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, “A Comprehensive Survey on Transfer Learning,” *Proceedings of the IEEE*, vol. 109, pp. 43–76, Jan. 2021. 2
- [6] N. Kalra and S. M. Paddock, “Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?,” *Transportation Research Part A: Policy and Practice*, vol. 94, pp. 182–193, Dec. 2016. 2
- [7] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, “CARLA: An Open Urban Driving Simulator,” *arXiv:1711.03938 [cs]*, Nov. 2017. 3
- [8] G. Rong, B. H. Shin, H. Tabatabaee, Q. Lu, S. Lemke, M. Možeiko, E. Boise, G. Uhm, M. Gerow, S. Mehta, E. Agafonov, T. H. Kim, E. Sterner, K. Ushiroda, M. Reyes, D. Zelenkovsky, and S. Kim, “LGSVL Simulator: A High Fidelity Simulator for

- Autonomous Driving,” in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1–6, Sept. 2020. 3, 10
- [9] “ApolloAuto/apollo.” Apollo Auto, July 2024. 3, 10
- [10] IPG Automotive GmbH, “IPG Carmaker.” <https://ipg-automotive.com/products-services/simulation-software/carmaker/>, 2024. 3
- [11] dSpace GmbH, “dSpace ASM (Advanced Simulation Models) Software.” <https://www.dspace.com/>, 2024. 3
- [12] Baidu, Inc., “Apollo Open Platform.” <https://apollo.auto/>, 2024. 3
- [13] The Autoware Foundation, “Autoware.” <https://www.autoware.org/>, 2024. 3, 4
- [14] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, “ROS: An open-source Robot Operating System,” 3
- [15] comma.ai, “Openpilot — open source advanced driver assistance system.” <https://www.comma.ai/openpilot>. 3, 4
- [16] “Model-in-the-Loop Testing - A. R. Plummer, 2006.” <https://journals-sagepub-com.utk.idm.oclc.org/doi/abs/10.1243/09596518JSCE207>. 3
- [17] S. Demers, P. Gopalakrishnan, and L. Kant, “A Generic Solution to Software-in-the-Loop,” in *MILCOM 2007 - IEEE Military Communications Conference*, pp. 1–6, Oct. 2007. 3
- [18] M. Basic, “On hardware-in-the-loop simulation,” in *Proceedings of the 44th IEEE Conference on Decision and Control*, pp. 3194–3198, Dec. 2005. 3
- [19] LD. Chen, Y. Papelis, G. Waston, and D. Solis, “NADS at the University of IOWA: A tool for driving safety research,” in *Proceedings of the 1st Human-Centered Transportation Simulation Conference*, 2001. 4
- [20] X. Cao, H. Chen, S. Y. Gelbal, B. Aksun-Guvenc, and L. Guvenc, “Vehicle-in-Virtual-Environment (VVE) Method for Autonomous Driving System Development, Evaluation and Demonstration,” *Sensors*, vol. 23, p. 5088, Jan. 2023. 4
- [21] “MCity.” <https://mcity.umich.edu/>. 4

- [22] N. Haghighi, X. C. Liu, G. Zhang, and R. J. Porter, “Impact of roadway geometric features on crash severity on rural two-lane highways,” *Accident Analysis & Prevention*, vol. 111, pp. 34–42, 2018. [4](#)
- [23] A. M. Hezaveh, R. Arvin, and C. R. Cherry, “A geographically weighted regression to estimate the comprehensive cost of traffic crashes at a zonal level,” *Accident Analysis & Prevention*, vol. 131, pp. 15–24, 2019. [4](#)
- [24] D. S. Shinstine, S. S. Wulff, and K. Ksaibati, “Factors associated with crash severity on rural roadways in Wyoming,” *Journal of traffic and transportation engineering (English edition)*, vol. 3, no. 4, pp. 308–323, 2016. [4](#)
- [25] J. Augenstein, K. Digges, E. Perdeck, J. Stratton, G. Bahouth, P. Baur, and G. Messner, “Application of ACN data to improve vehicle safety and occupant care,” in *Proceedings of the 17th International Technical Conference on the Enhanced Safety of Vehicles, Lyon, France*, 2007. [4](#)
- [26] D. W. Kononen, C. A. Flannagan, and S. C. Wang, “Identification and validation of a logistic regression model for predicting serious injuries associated with motor vehicle crashes,” *Accident Analysis & Prevention*, vol. 43, no. 1, pp. 112–122, 2011. [4](#)
- [27] K. D. Kusano and H. C. Gabler, “Automated crash notification: Evaluation of in-vehicle principal direction of force estimations,” *Transportation Research Part C: Emerging Technologies*, vol. 32, pp. 116–128, 2013. [4](#)
- [28] J. M. Scanlon, K. D. Kusano, and H. C. Gabler, “A preliminary model of driver acceleration behavior prior to real-world straight crossing path intersection crashes using EDRs,” in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pp. 938–943, IEEE, 2015. [4](#)
- [29] J. M. Scanlon, K. D. Kusano, and H. C. Gabler, “Analysis of driver evasive maneuvering prior to intersection crashes using event data recorders,” *Traffic injury prevention*, vol. 16, no. sup2, pp. S182–S189, 2015. [4](#)
- [30] K. Kikuchi, H. Hashimoto, T. Hosokawa, K. Nawata, and A. Hirao, “Relationship between pedestrian detection specifications of parking sensor and potential safety benefits,” *Accident Analysis & Prevention*, vol. 151, p. 105951, 2021/03/01/. [5](#)

- [31] R. Yu and S. Li, “Exploring the associations between driving volatility and autonomous vehicle hazardous scenarios: Insights from field operational test data,” *Accident Analysis & Prevention*, vol. 166, p. 106537, 2022/03/01/. [5](#)
- [32] P. Lindner and G. Wanielik, “3D LIDAR processing for vehicle safety and environment recognition,” in *2009 IEEE Workshop on Computational Intelligence in Vehicles and Vehicular Systems*, pp. 66–71, IEEE, 2009. [5](#)
- [33] J. Zhao, H. Xu, H. Liu, J. Wu, Y. Zheng, and D. Wu, “Detection and tracking of pedestrians and vehicles using roadside LiDAR sensors,” *Transportation Research Part C: Emerging Technologies*, vol. 100, pp. 68–87, 2019. [5](#)
- [34] J. Wu, H. Xu, Y. Zheng, and Z. Tian, “A novel method of vehicle-pedestrian near-crash identification with roadside LiDAR data,” *Accident Analysis & Prevention*, vol. 121, pp. 238–249, 2018. [5](#)
- [35] J. Wu, Y. Zhang, and H. Xu, “A novel skateboarder-related near-crash identification method with roadside LiDAR data,” *Accident Analysis & Prevention*, vol. 137, p. 105438, 2020. [5](#)
- [36] G. J. A. Sequeira, S. Afraj, R. Lugner, and T. Brandmeier, “LiDAR based prediction and contact based validation of crash parameters for a preemptive restraint strategy,” in *2019 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, pp. 1–7, IEEE, 2019. [5](#)
- [37] N. Carlini and D. Wagner, “Towards evaluating the robustness of neural networks,” in *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 39–57, IEEE Computer Society, 2017. [6](#)
- [38] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards deep learning models resistant to adversarial attacks,” in *International Conference on Learning Representations*, 2022. [6](#)
- [39] J. P. Miller, R. Taori, A. Raghunathan, S. Sagawa, P. W. Koh, V. Shankar, P. Liang, Y. Carmon, and L. Schmidt, “Accuracy on the line: On the strong correlation between out-of-distribution and in-distribution generalization,” in *International Conference on Machine Learning*, pp. 7721–7735, 2021. [6](#)

- [40] B. Feuer, A. Joshi, and C. Hegde, “A meta-analysis of distributionally-robust models,” *arXiv preprint arXiv:2206.07565*, 2022. [6](#)
- [41] S. Amani, C. Thrampoulidis, and L. Yang, “Safe reinforcement learning with linear function approximation,” in *Proceedings of the 38th International Conference on Machine Learning* (M. Meila and T. Zhang, eds.), vol. 139 of *Proceedings of Machine Learning Research*, pp. 243–253, PMLR, 2021-07-18/2021-07-24. [6](#)
- [42] G. Dalal, K. Dvijotham, M. Vecerik, T. Hester, C. Paduraru, and Y. Tassa, “Safe exploration in continuous action spaces,” *arXiv preprint arXiv:1801.08757*, 2018. [6](#)
- [43] K. Srinivasan, B. Eysenbach, S. Ha, J. Tan, and C. Finn, “Learning to be safe: Deep RL with a safety critic,” *arXiv preprint arXiv:2010.14603*, 2020. [6](#)
- [44] Q. Yang, T. D. Simão, S. H. Tindemans, and M. T. J. Spaan, “Safety-constrained reinforcement learning with a distributional safety critic,” *Machine Learning*, vol. 112, pp. 859–887, Mar. 2023. [6](#)
- [45] B. Thananjeyan, A. Balakrishna, S. Nair, M. Luo, K. Srinivasan, M. Hwang, J. E. Gonzalez, J. Ibarz, C. Finn, and K. Goldberg, “Recovery RL: Safe reinforcement learning with learned recovery zones,” *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4915–4922, 2021. [6](#)
- [46] Y. Emam, G. Notomista, P. Glotfelter, Z. Kira, and M. Egerstedt, “Safe reinforcement learning using robust control barrier functions,” *IEEE Robotics and Automation Letters*, pp. 1–8, 2022. [6](#)
- [47] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, “End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 3387–3395, July 2019. [6](#)
- [48] P. S. N. Mindom, A. Nikanjam, F. Khomh, and J. Mullins, “On assessing the safety of reinforcement learning algorithms using formal methods,” in *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*, pp. 260–269, IEEE, 2021. [6](#)

- [49] N. Fulton and A. Platzer, “Safe reinforcement learning via formal methods: Toward safe control through proof and learning,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, Apr. 2018. [6](#)
- [50] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, “Safe reinforcement learning via shielding,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, Apr. 2018. [6](#)
- [51] X. Li, *A Formal Methods Approach to Interpretability, Safety and Composability for Reinforcement Learning*. PhD thesis, Boston University, 2020. [6](#)
- [52] S. Ramakrishna, B. Luo, C. B. Kuhn, G. Karsai, and A. Dubey, “ANTI-CARLA: An Adversarial Testing Framework for Autonomous Vehicles in CARLA,” in *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 2620–2627, 2022-10, 2020. [6](#)
- [53] D. J. Fremont, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, and S. A. Seshia, “Scenic: A language for scenario specification and scene generation,” in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, Pldi 2019*, (New York, NY, USA), pp. 63–78, Association for Computing Machinery, 2019. [6](#)
- [54] P. Eigenschink, T. Reutterer, S. Vamosi, R. Vamosi, C. Sun, and K. Kalcher, “Deep Generative Models for Synthetic Data: A Survey,” *IEEE Access*, vol. 11, pp. 47304–47320, 2023. [7](#)
- [55] C. Esteban, S. L. Hyland, and G. Rätsch, “Real-valued (Medical) Time Series Generation with Recurrent Conditional GANs,” Dec. 2017. [7](#), [61](#), [64](#), [70](#), [71](#)
- [56] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling,” Dec. 2014. [7](#)
- [57] J. Yoon, D. Jarrett, and M. van der Schaar, “Time-series Generative Adversarial Networks,” in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019. [7](#), [61](#), [64](#), [70](#), [71](#), [72](#), [77](#)
- [58] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, “Adversarial Autoencoders,” May 2016. [7](#), [8](#), [61](#), [65](#)

- [59] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein GAN,” Dec. 2017. [7](#), [8](#)
- [60] M. N. Fekri, A. M. Ghosh, and K. Grolinger, “Generating Energy Data for Machine Learning with Recurrent Generative Adversarial Networks,” *Energies*, vol. 13, p. 130, Jan. 2020. [7](#)
- [61] T. Xu, L. K. Wenliang, M. Munn, and B. Acciaio, “COT-GAN: Generating Sequential Data via Causal Optimal Transport,” in *Advances in Neural Information Processing Systems*, vol. 33, pp. 8798–8809, Curran Associates, Inc., 2020. [7](#), [70](#)
- [62] D. Jarrett, I. Bica, and M. van der Schaar, “Time-series Generation by Contrastive Imitation,” in *Advances in Neural Information Processing Systems*, vol. 34, pp. 28968–28982, Curran Associates, Inc., 2021. [7](#), [64](#), [70](#)
- [63] A. Coletta, S. Gopalakrishnan, D. Borrajo, and S. Vyetrenko, “On the Constrained Time-Series Generation Problem,” *Advances in Neural Information Processing Systems*, vol. 36, pp. 61048–61059, Dec. 2023. [7](#), [70](#)
- [64] S. Dash, A. Yale, I. Guyon, and K. P. Bennett, “Medical Time-Series Data Generation Using Generative Adversarial Networks,” in *Artificial Intelligence in Medicine* (M. Michalowski and R. Moskovitch, eds.), Lecture Notes in Computer Science, (Cham), pp. 382–391, Springer International Publishing, 2020. [7](#)
- [65] H. Li, S. Yu, and J. Principe, “Causal Recurrent Variational Autoencoder for Medical Time Series Generation,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, pp. 8562–8570, June 2023. [7](#)
- [66] S. Chatteraj, S. Pratiher, S. Pratiher, and H. Konik, “Improving Stability of Adversarial Li-ion Cell Usage Data Generation using Generative Latent Space Modelling,” in *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8047–8051, June 2021. [7](#)
- [67] M. Fochesato, F. Khayatian, D. F. Lima, and Z. Nagy, “On the use of conditional TimeGAN to enhance the robustness of a reinforcement learning agent in the building domain,” in *Proceedings of the 9th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, BuildSys ’22, (New York, NY, USA), pp. 208–217, Association for Computing Machinery, Dec. 2022. [7](#)

- [68] E. Adib, A. S. Fernandez, F. Afghah, and J. J. Prevost, “Synthetic ECG Signal Generation Using Probabilistic Diffusion Models,” *IEEE access : practical innovations, open solutions*, vol. 11, pp. 75818–75828, 2023. [7](#)
- [69] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. ukasz Kaiser, and I. Polosukhin, “Attention is All you Need,” in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017. [7](#)
- [70] P. P. Ray, “ChatGPT: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope,” *Internet of Things and Cyber-Physical Systems*, vol. 3, pp. 121–154, Jan. 2023. [7](#), [75](#)
- [71] X. Lyu, M. Hueser, S. L. Hyland, G. Zerveas, and G. Raetsch, “Improving Clinical Predictions through Unsupervised Time Series Representation Learning,” Dec. 2018. [8](#)
- [72] A. M. Dai and Q. V. Le, “Semi-supervised Sequence Learning,” Nov. 2015. [8](#)
- [73] F. M. Bianchi, L. Livi, K. Ø. Mikalsen, M. Kampffmeyer, and R. Jenssen, “Learning representations for multivariate time series with missing data using Temporal Kernelized Autoencoders,” July 2019. [8](#)
- [74] D. P. Kingma and M. Welling, “Auto-Encoding Variational Bayes,” Dec. 2013. [8](#)
- [75] I. Tolstikhin, O. Bousquet, S. Gelly, and B. Schoelkopf, “Wasserstein Auto-Encoders,” Dec. 2019. [8](#)
- [76] M.-Y. Liu, T. Breuel, and J. Kautz, “Unsupervised Image-to-Image Translation Networks,” in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017. [8](#)
- [77] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, “Unpaired image-to-image translation using cycle-consistent adversarial networks,” in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2223–2232, 2017. [8](#), [10](#)
- [78] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang, and S. P. Smolley, “Least Squares Generative Adversarial Networks,” Apr. 2017. [8](#)
- [79] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, “Spectral Normalization for Generative Adversarial Networks,” Feb. 2018. [8](#)

- [80] S. Xiang and H. Li, “On the Effects of Batch and Weight Normalization in Generative Adversarial Networks,” Dec. 2017. [8](#)
- [81] A. Tampuu, T. Matiisen, M. Semikin, D. Fishman, and N. Muhammad, “A Survey of End-to-End Driving: Architectures and Training Methods,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, pp. 1364–1384, Apr. 2022. [9](#)
- [82] A. Pandharipande, C.-H. Cheng, J. Dauwels, S. Z. Gurbuz, J. Ibanez-Guzman, G. Li, A. Piazzoni, P. Wang, and A. Santra, “Sensing and Machine Learning for Automotive Perception: A Review,” *IEEE Sensors Journal*, vol. 23, pp. 11097–11115, June 2023. [9](#)
- [83] P. Adarsh, P. Rathi, and M. Kumar, “YOLO v3-Tiny: Object Detection and Recognition using one stage improved model,” in *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pp. 687–694, Mar. 2020. [9](#)
- [84] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” *arXiv:2004.10934 [cs, eess]*, Apr. 2020. [9](#)
- [85] “Ultralytics/yolov5.” Ultralytics, Nov. 2022. [9](#)
- [86] S. F. Bhat, I. Alhashim, and P. Wonka, “AdaBins: Depth Estimation Using Adaptive Bins,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4009–4018, 2021. [9](#)
- [87] C. Zhao, Q. Sun, C. Zhang, Y. Tang, and F. Qian, “Monocular depth estimation based on deep learning: An overview,” *Science China Technological Sciences*, vol. 63, pp. 1612–1627, Sept. 2020. [9](#)
- [88] D. Gragnaniello, A. Greco, A. Saggese, M. Vento, and A. Vicinanza, “Benchmarking 2D Multi-Object Detection and Tracking Algorithms in Autonomous Vehicle Driving Scenarios,” *Sensors*, vol. 23, p. 4024, Jan. 2023. [9](#)
- [89] L. Zhao, M. Wang, S. Su, T. Liu, and Y. Yang, “Dynamic Object Tracking for Self-Driving Cars Using Monocular Camera and LIDAR,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 10865–10872, Oct. 2020. [9](#)
- [90] comma.ai, “Openpilot-SuperCombo Vision Model.” <https://github.com/commaai/openpilot/tree/master/selfdrive/modeld/models>. [9](#), [76](#)

- [91] N. J. Zakaria, M. I. Shapiai, R. A. Ghani, M. N. M. Yassin, M. Z. Ibrahim, and N. Wahid, “Lane Detection in Autonomous Vehicles: A Systematic Review,” *IEEE Access*, vol. 11, pp. 3729–3765, 2023. [9](#)
- [92] M. Amodio and S. Krishnaswamy, “TraVeLGAN: Image-To-Image Translation by Transformation Vector Learning,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8983–8992, 2019. [10](#)
- [93] J. Hoffman, E. Tzeng, T. Park, J.-Y. Zhu, P. Isola, K. Saenko, A. Efros, and T. Darrell, “CyCADA: Cycle-Consistent Adversarial Domain Adaptation,” in *Proceedings of the 35th International Conference on Machine Learning*, pp. 1989–1998, PMLR, July 2018. [10](#)
- [94] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, “Learning transferable visual models from natural language supervision,” in *International Conference on Machine Learning*, pp. 8748–8763, PMLR, 2021. [10](#)
- [95] A. Piazzoni, J. Cherian, M. Slavik, and J. Dauwels, “Modeling Sensing and Perception Errors towards Robust Decision Making in Autonomous Vehicles,” *arXiv:2001.11695 [cs]*, Jan. 2020. [10](#)
- [96] A. Piazzoni, J. Cherian, J. Dauwels, and L.-P. Chau, “PEM: Perception Error Model for Virtual Testing of Autonomous Vehicles,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 25, pp. 670–681, Jan. 2024. [10](#), [89](#)
- [97] P. Mitra, A. Choudhury, V. R. Aparow, G. Kulandaivelu, and J. Dauwels, “Towards Modeling of Perception Errors in Autonomous Vehicles,” in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 3024–3029, Nov. 2018. [10](#)
- [98] R. Krajewski, M. Hoss, A. Meister, F. Thomsen, J. Bock, and L. Eckstein, “Using drones as reference sensors for neural-networks-based modeling of automotive perception errors,” in *2020 IEEE Intelligent Vehicles Symposium (IV)*, pp. 708–715, Oct. 2020. [10](#)

- [99] J. Zhao, Y. Li, B. Zhu, W. Deng, and B. Sun, “Method and Applications of Lidar Modeling for Virtual Testing of Intelligent Vehicles,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, pp. 2990–3000, May 2021. [10](#)
- [100] V. J. Expósito Jiménez, G. Macher, D. Watzenig, and E. Brenner, “Safety of the Intended Functionality Validation for Automated Driving Systems by Using Perception Performance Insufficiencies Injection,” *Vehicles*, vol. 6, pp. 1164–1184, Sept. 2024. [10](#)
- [101] C. Innes and S. Ramamoorthy, “Testing Rare Downstream Safety Violations via Upstream Adaptive Sampling of Perception Error Models,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 12744–12750, May 2023. [10](#)
- [102] H. Arnelid, E. L. Zec, and N. Mohammadiha, “Recurrent Conditional Generative Adversarial Networks for Autonomous Driving Sensor Modelling,” in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pp. 1613–1618, Oct. 2019. [10](#)
- [103] C. Esteban, S. L. Hyland, and G. Rätsch, “Real-valued (Medical) Time Series Generation with Recurrent Conditional GANs,” Dec. 2017. [10](#)
- [104] F. H. A. (FHWA), “Highway Statistics, 2016,” tech. rep., 2018. [13](#)
- [105] N. T. S. B. (NTSB), “Collision Between a Car Operating With Automated Vehicle Control Systems and a Tractor-Semitrailer Truck Near Williston, Florida, May 7, 2016,” 2017. [13](#)
- [106] N. T. S. Board, “Preliminary Report: HWY18FH011,” tech. rep., NTSB, 2018. [13](#)
- [107] N. T. S. B. (NTSB), “Preliminary Report: HWY18MH010,” tech. rep., NTSB, 2018. [13](#)
- [108] N. T. S. B. (NTSB), “Preliminary Report: HWY19FH008,” tech. rep., NTSB, 2018. [13](#)
- [109] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, “CARLA: An open urban driving simulator,” in *Conference on Robot Learning*, pp. 1–16, PMLR, 2017. [14](#), [28](#)
- [110] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets robotics: The KITTI dataset,” *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013/09/01/. [14](#), [23](#), [24](#)

- [111] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 779–788, 2016. [14](#), [23](#)
- [112] S. F. Bhat, I. Alhashim, and P. Wonka, “AdaBins: Depth Estimation Using Adaptive Bins,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4009–4018, 2021. [14](#), [24](#)
- [113] State of California Department of Motor Vehicles, “Autonomous vehicle collision reports,” 2020. [15](#)
- [114] N. Akai, L. Y. Morales, T. Yamaguchi, E. Takeuchi, Y. Yoshihara, H. Okuda, T. Suzuki, and Y. Ninomiya, “Autonomous driving based on accurate localization using multilayer LiDAR and dead reckoning,” in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1–6, IEEE, 2017. [22](#)
- [115] Y. Zhou and O. Tuzel, “Voxelnet: End-to-end learning for point cloud based 3d object detection,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4490–4499, 2018. [22](#)
- [116] S. Shi, X. Wang, and H. Li, “Pointnet: 3d object proposal generation and detection from point cloud,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 770–779, 2019. [22](#)
- [117] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, “Frustum pointnets for 3d object detection from rgb-d data,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 918–927, 2018. [22](#)
- [118] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. L. Waslander, “Joint 3d proposal generation and object detection from view aggregation,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–8, IEEE, 2018. [22](#)
- [119] L. Fridman, D. E. Brown, W. Angell, I. Abdić, B. Reimer, and H. Y. Noh, “Automated synchronization of driving data using vibration and steering events,” *Pattern Recognition Letters*, vol. C, no. 75, pp. 9–15, 2016. [22](#)

- [120] E. Olson, “A passive solution to the sensor synchronization problem,” in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1059–1064, 2010/10//. [22](#)
- [121] “Ultralytics/yolov5.” Ultralytics, Nov. 2022. [23](#)
- [122] “Papers with Code - KITTI Eigen split Benchmark (Monocular Depth Estimation),” [24](#)
- [123] S. Wang and Z. Li, “Exploring causes and effects of automated vehicle disengagement using statistical modeling and classification tree based on field test data,” *Accident Analysis & Prevention*, vol. 129, pp. 44–54, 2019/08/01/. [29](#)
- [124] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, “CARLA: An Open Urban Driving Simulator,” in *Proceedings of the 1st Annual Conference on Robot Learning*, pp. 1–16, PMLR, Oct. 2017. [39](#)
- [125] comma.ai, “Openpilot — open source advanced driver assistance system,” 2023. [39](#)
- [126] O. Mogren, “C-RNN-GAN: Continuous recurrent neural networks with adversarial training,” Nov. 2016. [61](#)
- [127] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to Sequence Learning with Neural Networks,” Dec. 2014. [61](#)
- [128] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer, “Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks,” Sept. 2015. [63](#)
- [129] A. Lamb, A. Goyal, Y. Zhang, S. Zhang, A. Courville, and Y. Bengio, “Professor Forcing: A New Algorithm for Training Recurrent Networks,” Oct. 2016. [63](#)
- [130] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014. [69](#), [86](#)
- [131] L. M. Candanedo, V. Feldheim, and D. Deramaix, “Data driven prediction models of energy use of appliances in a low-energy house,” *Energy and Buildings*, vol. 140, pp. 81–97, Apr. 2017. [70](#)
- [132] J. Hogue, “Traffic data from MN department of transportation, weather data from OpenWeatherMap,” 2018. [70](#)

- [133] L. van der Maaten and G. Hinton, “Visualizing Data using t-SNE,” *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008. [71](#)
- [134] H. Schafer, E. Santana, A. Haden, and R. Biasini, “A Commute in Data: The comma2k19 Dataset,” Dec. 2018. [76](#), [80](#), [83](#)
- [135] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, “A tutorial on binary cross-entropy loss for classification tasks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 10, pp. 1710–1722, 2005. [77](#)
- [136] V. B. Kampenes, T. Dybå, J. E. Hannay, and D. I. K. Sjøberg, “A systematic review of effect size in software engineering experiments,” *Information and Software Technology*, vol. 49, no. 11-12, pp. 1073–1086, 2007. [89](#), [90](#)
- [137] B. W. Silverman, *Density Estimation for Statistics and Data Analysis*. No. 26 in Monographs on Statistics and Applied Probability, London: Chapman and Hall/CRC, 1986. [94](#)

Vita

Born and raised in Murfreesboro Tennessee, Joe Beck graduated from Oakland High School in 2013 with an interest in pursuing a career in engineering. Joe attended Middle Tennessee State University and studied Mechatronics Engineering. There, he gained a passion for a multidisciplinary approach to solving tough problems. After graduating at the top of his class in 2017, Joe moved to Charlottesville, VA and worked as an electrical engineer, where he developed on a variety of robotics-based solutions for food-related packaging. In 2019, Joe joined Dr. Chakraborty's lab as a graduate student and began his career pursuing transportation research. Upon graduation, Joe hopes to continue applying his broad range of expertise in robotics and vehicle-related research environments.