Synthetic Data Generation for Lane Detection: Validation and Analysis

PEDRO CLÁUDIO AMARO DA COSTA
(M.Sc.)

Dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science in Informatics and Multimedia Engineering

Supervisor:
Professor Arnaldo Abrantes, Ph.D.

Jury:
Professor Pedro Fazenda, Ph.D. (President)
Professor Pedro Jorge, Ph.D.
Professor Arnaldo Abrantes, Ph.D.

November, 2024

Synthetic Data Generation for Lane Detection: Validation and Analysis

PEDRO CLÁUDIO AMARO DA COSTA
(M.Sc.)

Dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science in Informatics and Multimedia Engineering

Supervisor:
Professor Arnaldo Abrantes, Ph.D.

Jury:
Professor Pedro Fazenda, Ph.D. (President)
Professor Pedro Jorge, Ph.D.
Professor Arnaldo Abrantes, Ph.D.

November, 2024

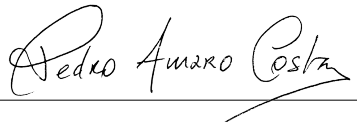
Acknowledgments

The successful completion of this thesis was only made possible by the guidance of Professor Arnaldo Abrantes and Professor Rui Jesus. I am also grateful for their support in submitting the affiliated article ”Harness the Unreal: Evaluating State-of-the-Art Lane Detection with Synthetic Data” at the 7th International Conference on Mathematics and Statistics (ICoMS 2024), available in Annex D. The publication of this article was supported by NOVA LINCS (UIDB/04516/2020), with the financial contribution from FCT.IP. Additionally, I wish to compliment Ms. Jamila Liu and Joana Dâmaso, the Research Director of NOVA LINCS—NOVA Laboratory for Computer Science and Informatics, for their invaluable help.

Statement of integrity

I declare that this dissertation is the result of my personal and independent research. Its content is original, and all sources listed in the bibliographic references were consulted and are duly mentioned in the text. I further declare that all scientific and technical references relevant to the development of the work are duly cited and included in the bibliographic references.

The author



Lisbon, November, 2024

Abstract

Autonomous driving systems rely absolutely on lane detection. Therefore, ensuring its reliability is crucial for road safety. This work proposes validating one of the leading lane detection models in the CULane benchmark with an alternative synthetic dataset, with full automated ground truth labeling, from Epic Games' Unreal Engine 5—a dynamically enriched, photorealistic simulation environment. By providing a range of diverse and challenging conditions (circadian, climatic, and road types), we aim to analyze the algorithm's robustness and, in parallel, collect reference indicators of the domain gap versus real-world datasets. Results reinforce the role of synthetic data in expanding test coverage and minimizing the imbalance of training datasets for safety-critical applications.

Keywords

Synthetic Data, Domain Gap, Lane Detection, Autonomous Driving

Index

List of Figures

List of Tables

Symbology and Abbreviations

ADAS	Advanced Driving Assistance System
ASPICE	Automotive Software Process Improvement and Capability Determination
BLIS	Blind Spot Information System
CARLA	Car Learning to Act
CDCF	Corrective Directional Control Function
CNN	Convolutional Neural Network
CycleGAN	Cycle-Consistent Generative Adversarial Network
DA	Domain Adaptation
EHM	Exact Histogram Matching
ELKS	Emergency Lane Keeping Systems
ESS	Emergency Stop Signal
EU	European Union
FDM	Feature Distribution Matching
FOT	Field Operational Tests
FPN	Feature Pyramid Network
GSR	Global Safety Regulation
GPS	Global Positioning System
HiL	Hardware-in-the-Loop
HUD	Heads-Up Display
ICoMS	International Conference on Mathematics and Statistics
LIoU	Line Intersection over Union
IoU	Intersection over Union
ISA	Intelligent Speed Assistance
KPI	Key Performance Indicators
LDW	Lane Departure Warning
NMS	Non-Maximum Suppression
OEM	Original Equipment Manufacturer
PCW	Pedestrian Collision Warning
POV	Point-of-View
ROI	Region of Interest
SAE	Society of Automotive Engineers

SiL	Software-in-the-Loop
SuT	System-under-Test
UE5	Unreal Engine 5
V2V	Vehicle-To-Vehicle
V2X	Vehicle-To-Everything
V&V	Verification & Validation

Chapter 1

Introduction

Recent advances in self-driving vehicles and driver assistance systems have far-reaching implications for the future of road transportation. While cities are opening up to a broader understanding of mobility, welcoming cyclists and pedestrians into a friendlier, more vibrant, and more familiar scenario, the growing concern for the quality of life in urban environments comes against an age-old problem: traffic accidents and injuries are still an archaic scar in the modern world. Thus, it is vital not only to further the development of self-driving vehicles but also to weigh their impact on multiple heterogeneous agents, whether traditional or professional drivers, cyclists, or pedestrians. Technological progress increases expectations and should yield risk-free outcomes or at least surpass human competency levels.

The development of autonomous driving systems goes hand in hand with efforts to reduce the number of fatalities and serious injuries from traffic accidents, which is seen in programs such as Vision Zero [European Commission, 2020]. The main principle of this safety initiative, launched in Sweden in 1997, is that mobility should never come at the cost of human life. Since then, thanks to various national, regional, and local initiatives, the number of deaths on European roads has halved. European organizations have set out to achieve this ambitious goal of zero traffic deaths by 2050 with a regulatory plan for automated and connected mobility. To support a safe evolution towards higher levels of automated driving, the EU has adopted specifications for cooperative intelligent transport systems (including vehicle-to-vehicle and vehicle-to-infrastructure communication) as well as a code of conduct with requirements and procedures focusing on mixed traffic, interaction with other road users, transition of control, and degradation.

Despite numerous projects that have emerged in the automotive sector in recent years, there are still significant legal and technical challenges in implementing autonomous driving on a larger scale. Numerous studies have shown that Original Equipment Manufacturers (OEMs) need to cover billions of kilometers to validate the reliability and robustness of their systems [Hauer, Schmidt, Holzmüller, & Pretschner, 2019]. One of the biggest challenges in the transition to full autonomous driving is, therefore, certifying this safety premise, given the virtually unlimited range of driving scenarios that can occur in the real world.

1.1 Context

EU Regulation 2019/2144 establishes Emergency Lane Keeping Systems (ELKS) mandatory for vehicle homologation and applicable for any new vehicles starting in July 2024 [European Union, 2019]. The requirements outlined in this new regulation define ELKS as a combination of two distinct systems: a Lane Departure Warning (LDW), which alerts the driver if the vehicle deviates from its lane or steers off-road, and a Corrective Directional Control Function (CDCF), which automatically modifies its trajectory to correct an unintentional lane departure [InterRegs Limited, 2021].

Also known as the EU General Safety Regulation II (GSR), this document intends to set minimum standards for the safety of motor vehicles (see Figure 1.1). It is a historic ruling, now legally binding all manufacturers to the strategic leap that the front-runners of the automotive industry have already initiated in recent decades. Its main objective is to promote the universal adoption of life-saving technologies protecting vehicle occupants, pedestrians, and cyclists and mitigating human error. It aims for greater safety and comfort for drivers and can reduce fatalities by addressing one of its principal scenarios: involuntary lane departures. Studies from Jermakian [2011] and Sternlund [2016] have confirmed that, though not a primary cause, lane departure collisions have the highest fatality rate of all accidents.

As described by Continental Automotive [2021], other automated driving functions are also mandatory for all new vehicles, including, for example, Emergency Stop Signal (ESS), Intelligent Speed Assistance (ISA), Pedestrian Collision Warning (PCW), or Blind Spot Information System (BLIS).

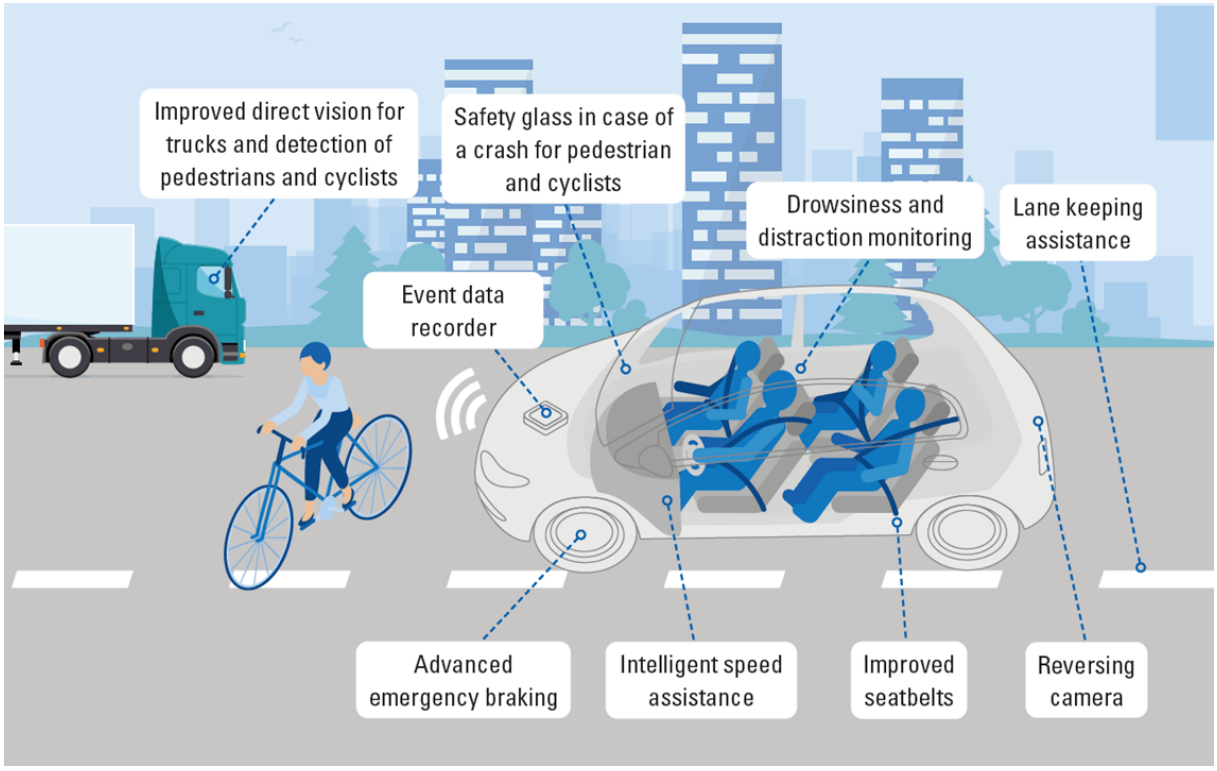


Figure 1.1 Required vehicle safety features in GSR II [TÜV SÜD, 2022].

Overall, GSR II drastically changes how automakers develop, test, produce, and deploy vehicles on the road. The entire landscape of manufacturers and suppliers will inevitably face new challenges and complexity, particularly in software development, sensor technology, and verification and validation.

1.2 Problem

The automotive industry typically sets requirements that imply formal proof of robustness and dependability in safety-critical applications. First, to guarantee that the system continues to operate effectively despite erratic traffic patterns, unforeseen road designs, unfavorable weather conditions, and several other demanding use-case scenarios. Second, to ensure that it does not degrade its performance over time yet upholds a threshold of confidence throughout its life cycle.

Regarding ELKS, the efficacy of the underlying lane detection functionality is pivotal. As part of the perception stack of Advanced Driver Assistance Systems (ADAS), lane detection is essential for guaranteeing that its environmental model correctly registers the traffic layout of the road along with its surrounding geometry and dynamic elements. The goal is to precisely extract the driving lane boundaries and track these markings continuously in real-time. It supports the segmentation of the appropriate navigation path and, thus, factors into trajectory planning and motion control in autonomous vehicles.

As noted by Zakaria et al. [2023], lane detection methods have mainly employed deep learning models in recent years, either in standalone end-to-end architectures or with classical computer vision approaches supported by geometric modeling in conventional pipelines. Even though classic methods provide pixel-preciseness, transparency, and predictability (non-stochastic training), they may be too exhaustive or sluggish for real-time applications and often rely on complex heuristics or handcrafted configurations that struggle to adapt to eventful scenarios or unexpected road layouts. In contrast, results have overwhelmingly shown the high generalization capabilities of convolutional neural networks and encoder-decoder structures. However, adding deep learning techniques imposes a new challenge: safeguarding the system's performance if the underlying components possess inherently opaque, emergent characteristics. Borg et al. [2018] defend that OEMs should necessarily augment requirements specifications and software testing procedures in response to the increasing dependency on neural networks. One common strategy has been to expand testing from exclusive on-road into simulation environments, matching this growth of formal verification scenarios and the corresponding coverage needed. Stress or endurance testing and exhaustive KPI evaluation now acquire paramount importance.

In conclusion, the growing complexity of autonomous driving functions with the widespread adoption of deep learning models has significantly increased the demand for rigorous testing and validation. Lane detection as a fundamental component of autonomous driving systems requires meticulous evaluation to ensure reliability and safety in real-world traffic.

1.3 State-of-the-Art

Autonomous driving systems heavily rely on perception modules to accurately interpret the vehicle's surroundings through sensor data. A critical perception task is the aforementioned lane detection, which involves identifying road markings to facilitate longitudinal and lateral control. To ensure functional safety across diverse driving conditions, rigorous validation is paramount.

Despite the inherent challenges, the industry has made significant strides in developing Level 2 and Level 2+ systems, with a gradual progression towards Level 3 automation (see Figure 1.2). These advancements are driven by the need for premium features and customer confidence in partially automated functions. Traditionally, validation has primarily relied on real-world data collected from vehicles equipped with sensors in various traffic scenarios. To complement real-world data and accelerate development, the automotive industry is exploring virtual environments and synthetic data generation techniques.

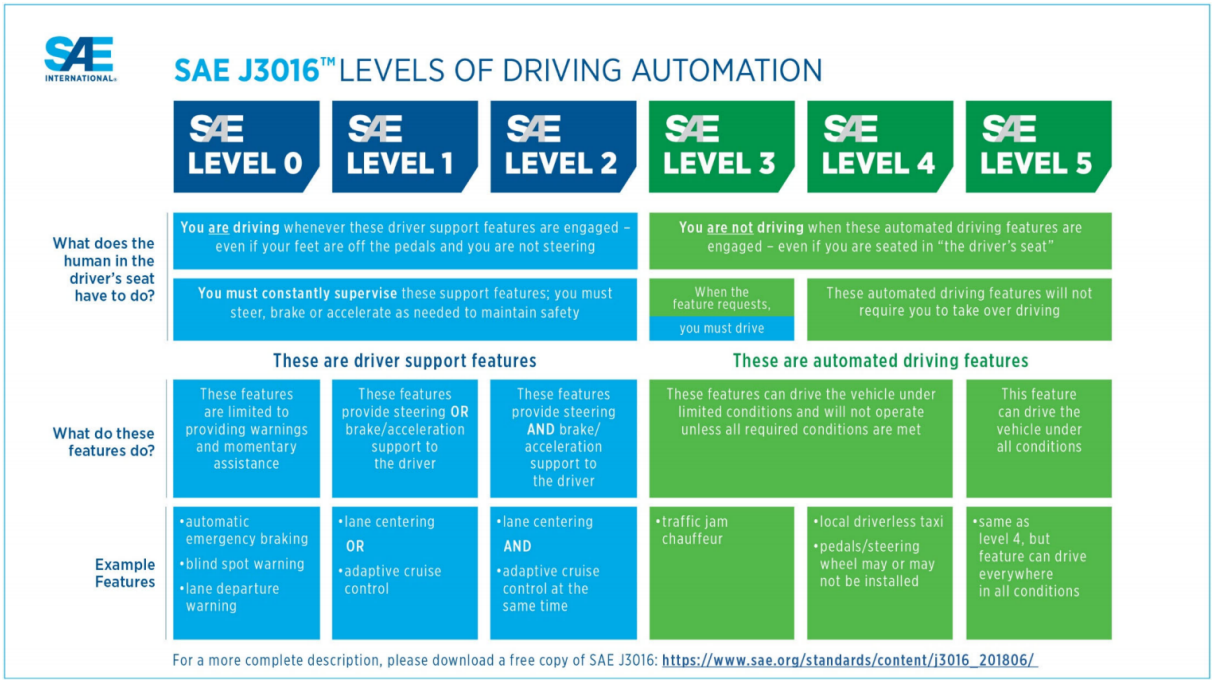


Figure 1.2 SAE J3016 standards for automated driving systems [SAE International, 2021].

1.3.1 Real-World Data

These validation efforts have yielded numerous proprietary and publicly accessible datasets. The latter, often prevalent through widespread scientific adoption, provide high-quality data, accurate ground truth labels, and standardized validation metrics. As such, they serve as a reference point for researchers to measure progress against other state-of-the-art methods and for the industry to assess the performance of proprietary or production algorithms (see Table 1.1).

CULane [Pan et al., 2018], TuSimple [TuSimple, 2017], LLAMAS [Behrendt & Soussan,

Popularity	Dataset	Best Model
1	CULane	CLRerNet-DLA34
2	TuSimple	SCNN_UNet_Attention_PL*
3	CurveLanes	CondLSTR (ResNet-101)
4	LLAMAS	CLRNet (DLA-34)
5	BDD100K val	YOLOPv2
6	nuScenes	DSLP
7	OpenLane	CondLSTR (ResNet-18)
8	Caltech Lanes Washington	VPGNet
9	Caltech Lanes Cordova	VPGNet
10	DET	LDNet

Table 1.1 Lane detection benchmarks and optimal models in PapersWithCode [2024].

2019], BDD100K [Yu et al., 2020], and CurveLanes [Xu et al., 2020] are some of the prominent benchmarks in lane detection research. These datasets, which originate from extensive real-world driving data captured by camera-equipped vehicles, cover a variety of traffic situations, geographical regions, meteorological variables, and lighting settings. Ground truth annotations typically consist of pixel-wise markers for lane-fitting polynomial lines and, in some cases, lane-type information (solid, dashed, or double).

1.3.2 Virtual Environments

To complement real-world data and accelerate development, the automotive industry has long been exploring virtual environments and synthetic data generation techniques. These methods present a chance to generate extensive, heterogeneous datasets and facilitate thorough testing and validation of autonomous driving systems. They have become indispensable in recent years because of the ever-increasing miles required to meet testing standards, equipment or driver costs, and the inherent risks associated with deploying prototype systems onto actual roads. Following the latest surveys [Y. Li et al., 2024; Tang et al., 2023] and market research [Mordor Intelligence, 2024], these are some of the most comprehensive and reputable solutions in the market:

- **IPG CarMaker** (see Figure 1.3) is a highly reputable simulation tool in the automotive industry for validating autonomous driving functions, vehicle dynamics, powertrain, and control systems [IPG Automotive, 2024b]. It often serves as a core simulation platform for X-in-the-Loop setups. Its most distinctive features are the accurate production-level vehicle models and the integration of signal visualization, environment modeling, and testing management tools. It also enables loading 3D models from external real-world maps for road layout and traffic topology.

Figure 1.3 IPG CarMaker [IPG Automotive, 2024b].

- **dSPACE AURELION** (see Figure 1.4) is a robust simulation environment for developing and validating autonomous driving systems [dSPACE, 2024]. With ray tracing support, it excels at realistic sensor simulation, especially for radar and lidar, and produces high-fidelity data under a variety of driving circumstances, allowing for rigorous testing of perception algorithms. When combined with other dSPACE tools, AURELION creates a complete development ecosystem that may be used, for example, to set up virtual scenarios with procedural generation or reconstruct real-world data. While not as extensively focused on vehicle dynamics as IPG CarMaker, its emphasis is on sensor simulation. Like other platforms, it has Unreal Engine as its rendering source [Unreal Engine, 2024].

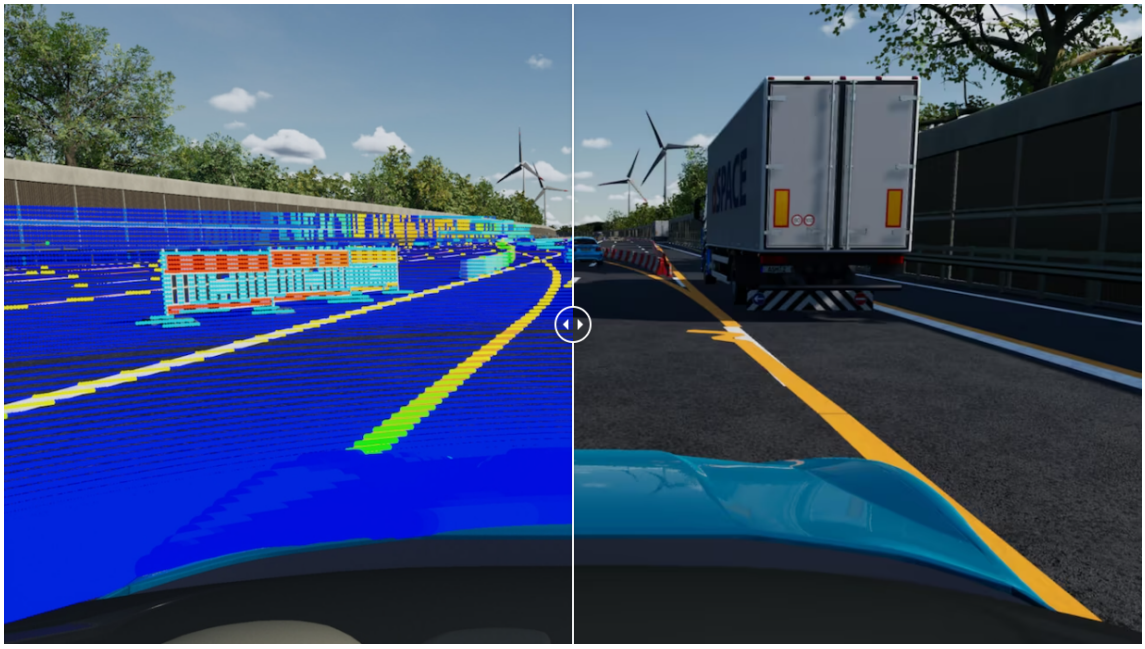


Figure 1.4 dSPACE AURELION [Unreal Engine, 2024].

- **NVIDIA DRIVE Sim** (see Figure 1.5) is an end-to-end simulation platform for testing autonomous driving systems [NVIDIA, 2024]. Also based on Unreal Engine, it offers a highly realistic range of driving scenarios, from urban streets to complex rural roads. As part of the NVIDIA Omniverse platform, it is highly scalable, allowing it to simulate large-scale testing series. NVIDIA also includes physical sensor modeling to properly replicate signal complexity obtained with real sensors (radar, lidar, cameras, and GPS). It has recreated actual locations with high levels of detail and realism.

Figure 1.5 NVIDIA DRIVE Sim [NVIDIA, 2020].

- **CARLA** (see Figure 1.6) is an open-source simulator built upon Unreal Engine. Unlike CarMaker, AURELION, and NVIDIA DRIVE Sim, CARLA grants finer control over the simulation environment through its C++/Blueprint source, enabling advanced customization and modeling. On the other hand, it also offers an intuitive Python client interface for easier access to advanced driving behaviors in realistic scenarios. In contrast to other commercial solutions, CARLA fosters an open-source community for collaborative development and knowledge sharing. Despite its limitations in terms of vehicle dynamics and signal fidelity, it incorporates a comprehensive sensor suite with configurable parameters and ground truth data for accurate perception algorithm development. The platform's preset virtual urban environments, populated with dynamic pedestrians and vehicles, establish an acceptable level of complexity [CARLA, 2024].

Figure 1.6 CARLA [CARLA, 2024].

- **VI-WorldSim** (see Figure 1.7) is a proprietary simulation tool that introduces advanced traffic models and sensor fusion algorithms. Based on Unreal Engine for highly realistic virtual environments, it is tailored for specific use cases, such as Vehicle-to-Vehicle (V2V) and Vehicle-to-Everything (V2X) communication [VI-GRADE, 2024].

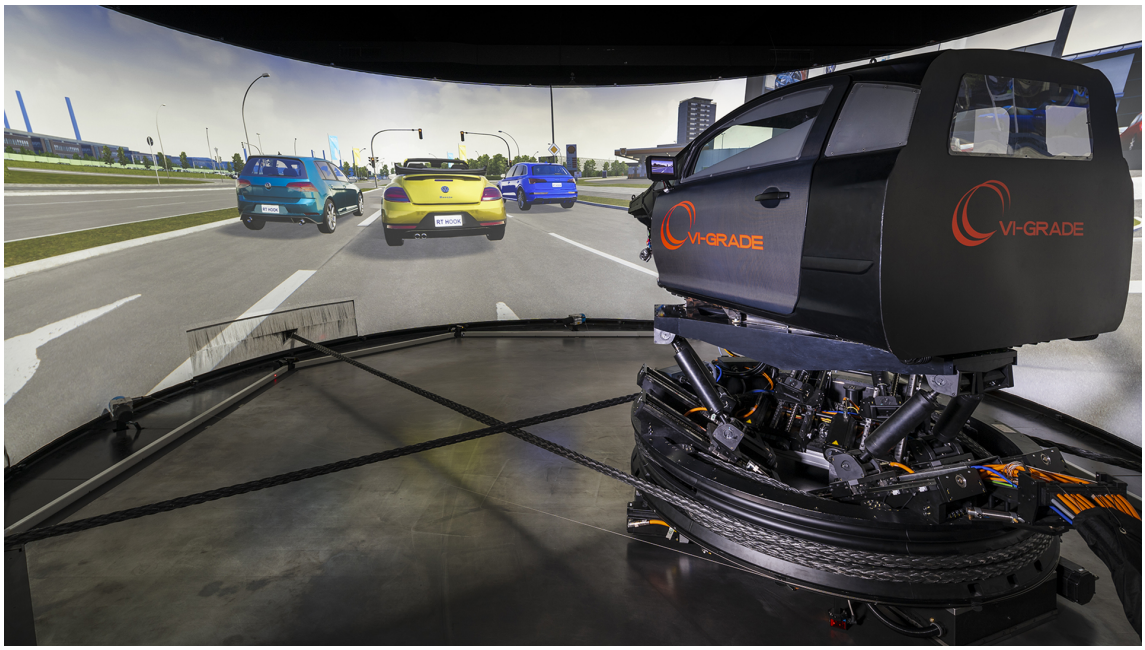


Figure 1.7 VI-WorldSim [VI-GRADE, 2024].

- **AVSimulation SCANeR** (see Figure 1.8) stands as a flexible simulation platform that integrates a set of complete and editable models for roads, vehicle dynamics, traffic, sensors, and 3D assets, from generic ideal modeling to high-level physics-based models. Similar to AURELION, SCANeR prioritizes high-fidelity sensors to generate realistic synthetic data and relies on Unreal Engine for visual rendering [AVSimulation, 2024].

Figure 1.8 AVSimulation SCANeR [AVSimulation, 2024].

1.3.3 Synthetic Techniques

Regarding synthetic techniques, many works have used the open-source simulator CARLA [CARLA, 2024] for data generation. Tran and Le [2019] trained and validated a U-Net segmentation network for extracting lane marking features with a dataset of 4000 images from the simulator. *Simulanes* is a CARLA-based dataset generator that supports a domain adaptation technique with adversarial generative and feature discriminators (see Figure 1.9). The goal is to train the learning model on a synthetic domain with simulation-based labeled scenes and predict lanes on a given real-data domain [C. Hu et al., 2022]. *CarlaScenes* is a benchmark dataset specifically designed to analyze the performance of odometry models on a collection of specific test cases with camera and LIDAR-labeled data [Kloukiniotis et al., 2022].

More recent examples of synthetic data generation already incorporate the 5^{th} version of Unreal Engine. Damian et al. [2023] trained object recognition algorithms with data from the platform. Accurate 3D replicas of real-world objects are scanned using photogrammetry, imported into various simulation scenarios, and compiled as a training set for YOLOv8 to increase robustness. Y. Hu, Datta, Beerel, and Beerel [2023] developed a synthetic dataset using UE5 to evaluate the impact of camera shutter types on object detection's accuracy, specifically for low-speed traffic participants (pedestrians).

Garnett et al. [2020] trained lane detection models with a reduced part of the TuSimple and LLAMAS datasets (10%)—the remainder consisted of synthetically generated data from 3D modeling (Blender), with parameterized variability in road and lane topology, topography, and curvature (see Figure 1.10). Nie et al. [2022] generated artificially foggy scenes with an atmospheric scattering model and applied them as a data augmentation method to the CULane dataset, improving lane detection on both fog and open sky images (see Figure 1.11).

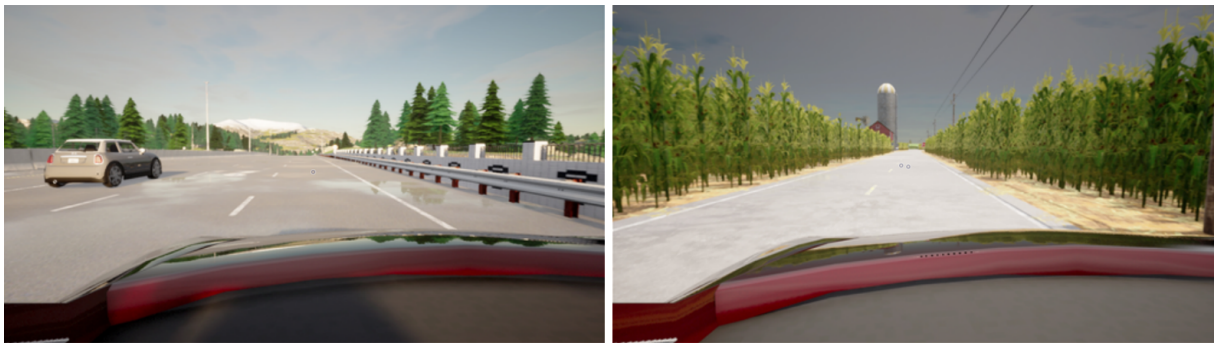


Figure 1.9 Synthetic images from *Simulanes* [C. Hu et al., 2022].

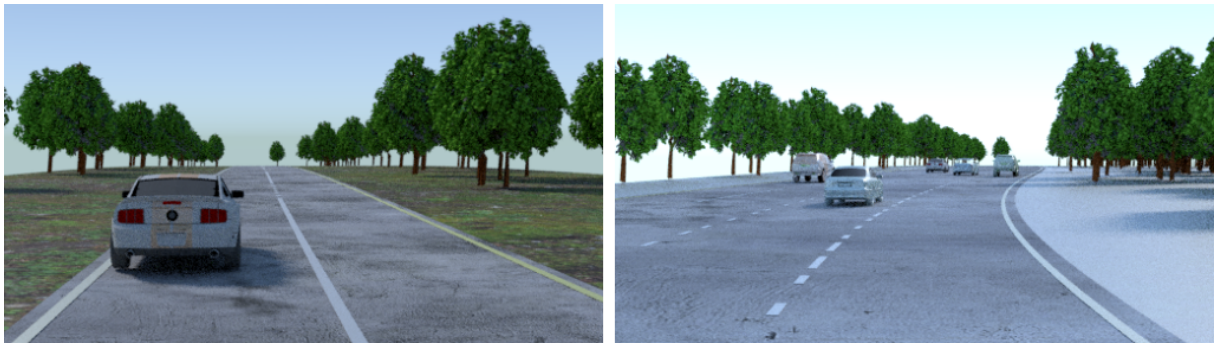


Figure 1.10 Synthetic images from Garnett et al. [2020].

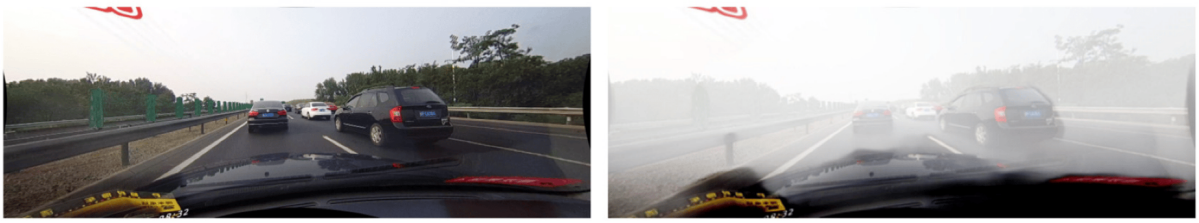


Figure 1.11 Synthetic fog samples from Nie et al. [2022].

1.4 Objectives & Scope

The goal of this project is to contribute to the development of a visual perception validation framework. Specifically, this framework shall support the performance evaluation of a defined range of autonomous driving functions. Given its importance as a fundamental component of environment modeling in ADAS, lane markings' detection is the function under test within the scope of this work. A synthetic dataset with fully automated ground truth labeling for lane markings is subsequently its main result. The relevant performance results of the selected algorithms form the secondary outcome. The lane detection algorithm selected as System-under-Test (SuT) is:

- *Cross Layer Refinement Network for Lane Detection* (CLRNet)—an open-source model with the highest and third-highest performance as of 2024 on LLAMAS and CULane datasets, respectively (see Table 1.1).

The environment chosen as the source for generating synthetic data is Unreal Engine 5 (UE5)—a dynamically enhanced, photorealistic simulation engine. The choice of this engine stems from its prominent position in most modern driving simulation platforms, as previously showcased. The planned tasks for the environment require the implementation of a camera sensor for the ego-vehicle, procedural control over road users, accurate ground truth labeling, and real-time recording of synthetic image data. In parallel, the validation process includes the definition of traditional performance metrics (precision, recall, and F1 score). Based on the results obtained, we can evaluate the performance of CLRNet, especially in unfavorable weather and lighting conditions (fog, glare, and night).

Finally, this thesis analyzes the domain gap between synthetic and real data by examining the algorithms' performance in our synthetic dataset versus its performance in CULane, one of the most referenced dataset benchmarks in the literature [Pan et al., 2018]. To this end, we implement a synthetic discriminator based on a Convolutional Neural Network (CNN) architecture to obtain reference indicators for the verisimilitude and similarity of the synthetic dataset towards real driving data. In parallel, we investigate relevant measures for domain adaptation as further improvements and future directions.

1.5 Document Overview

This work contains four chapters. The introduction describes the current context in the automotive industry and highlights the key motivations and challenges for the progress of autonomous driving. It also examines related work, namely some of the most prominent projects that have generated synthetic data for testing and validation purposes.

The 2nd Chapter explains the underlying methodology. This includes an overview of the concepts, requirements, and practices in ADAS development, with a particular focus on testing automated driving functions. This will include an examination of lane detection functions to illustrate the specifics of the validation process. In parallel, we will analyze the advantages and disadvantages of generating and integrating synthetic data as part of an improved strategy to achieve extended test coverage.

The 3rd Chapter explains the implementation tasks in connection with the selected simulation environment, the pipeline set up for the generation of synthetic data, and the prepared test case variants.

The 4th Chapter elaborates on the model proposed as SuT, describes the metrics, and analyzes the results, in particular the variance between the different test case scenarios. It addresses the discrepancy between the simulation and the real domain by comparing the performance of CLRNet on synthetic data with its original results on real data. Finally, it presents relevant measures to identify this discrepancy using a synthetic discriminator and to reduce the domain gap.

Chapter 2

Methodology

2.1 Verification & Validation

To be certain that an autonomous vehicle operates safely in an unpredictable traffic environment, the automotive industry has a thorough Verification and Validation (V&V) methodology. As described by Wishart et al. [2021]:

- **Verification** ensures the system is constructed correctly, adhering to rules and specifications from traffic and transportation entities, automakers, suppliers, and customers.
    1. Requirements—Ensuring clarity, completeness, and consistency.
    2. Design—Confirming the architecture meets requirements.
    3. Code—Ensuring the code implements the architecture.
    4. Process—Checking compliance with standards.
- **Validation** guarantees the system delivers its intended functionality and meets safety standards. Relies mostly on scenario-based testing on proving grounds, real-world conditions, and simulation to collect exhaustive indicators on the system's response.
    1. System—Verifying the system meets customer specifications.
    2. Performance—Evaluating performance under various conditions.
    3. Durability—Assessing resistance to environmental/operational stress.
    4. Safety—Confirming compliance with safety standards.

This organized approach is necessary to ensure robustness and reliability as automated driving systems become more complex and regulations increase their requirements. Briefly, V&V is a key method that formalizes a set of activities to assert the quality of a product during its lifecycle. In this work, it constitutes the conceptual backdrop in which all testing procedures for an autonomous driving system shall fit.

While V&V provides the overarching concept, ASPICE offers the standard framework for implementing V&V processes within the automotive industry. Essentially, ASPICE (Automotive

Software Process Improvement and Capability Determination) provides a roadmap for companies to implement their V&V processes and ensure that software products meet the required quality standards and customer expectations [Visure Solutions, 2024]. One of the most important guidelines of ASPICE is the V-model (see Figure 2.1). It dictates constant evaluation to ensure continuous improvement—by setting a testing phase for each development stage of the project. Developers benefit from being able to eliminate potential problems in the early stages, while customers benefit from a thorough scrutiny of the development process.

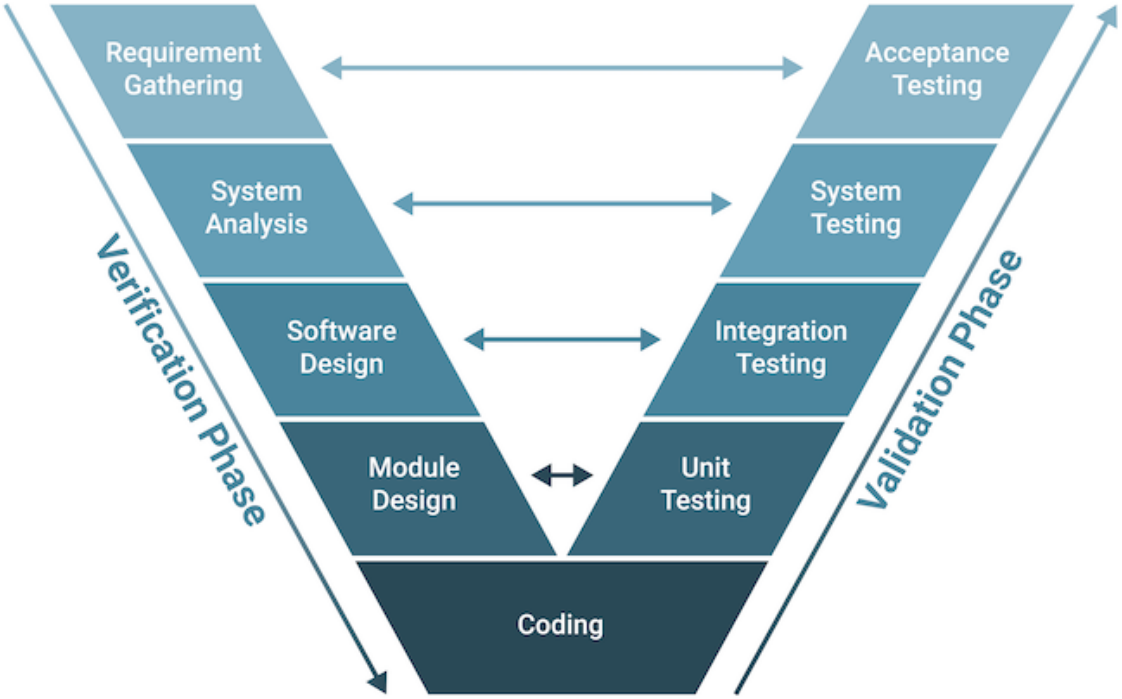


Figure 2.1 V-model in ASPICE [Built In, 2024].

This work proposes to integrate synthetic data generation into what is referred to as the system testing phase of the V-model, specifically for algorithmic testing. By establishing a virtual environment capable of producing realistic camera images, it provides a data source that can be injected into the lane detection algorithm. Since the iterative nature of the V-model validation phase demands continuous testing at all stages and for every specific component, the role of a high-volume and readily available data pipeline stands as quite crucial for performance maturity.

2.1.1 Scenario-Based & Endurance Testing

Validation, as mentioned previously, primarily utilizes scenario-based testing, conducted either in test tracks, public roads, or virtual environments. According to Hauer et al. [2019], the scenario-based testing methodology implies that, before deployment, one must validate the system on all reasonably expected traffic scenarios. Test engineers design and implement

target challenges, also known as test cases, having the system requirements as the reference. The resulting spatial state, to which the automated driving system must confirm compliance, represents the entire testing coverage. These states then fit into broad categories of common, generic traffic maneuvers, such as ”overtaking maneuver” and ”emergency braking,” among others.

In terms of scale, these categories branch out to a multitude of particular scenario variations by altering a road feature (curvature, lane type, markings), maneuver parameter (speed, acceleration, start offset), traffic element (pedestrian, vehicle, or road object), or any other testing element. To ensure that the system operates as intended for all edge situations and crucial maneuvers, scenario-based testing may then naturally encompass hundreds, thousands, or tens of thousands of test cases. The goal of this method is to track, as profusely and exhaustively as possible, the system's decision-making in high-risk or high-relevance scenarios that serve as demonstrations of automated driving in complex, everyday traffic. The functional safety of the system derives subsequently from each result, and the complete testing space state indicates the system's systemic proficiency.

On the one hand, the advantages of scenario-based testing compared to traditional test drives are remarkable since it allows to respond to each system requirement with an unambiguous formal verdict. In parallel, it also permits the objective and efficient estimation/coordination of the global validation efforts. However, this methodology constitutes only one facet of a comprehensive validation strategy and is not without limitations. Primarily, the lack of a final criterion for test coverage. Hauer et al. [2019] explain that if any new system requirement is brought into the equation, it will add numerous test specifications and new variation branches. Simultaneously, it does not provide any guarantee that, in absolute terms, all truly relevant test cases are known and established. As with any sampling strategy, implicit discretization of test cases can lack granularity and extension.

Following Hakuli and Krug [2014], the traditional approach—endurance testing—is to subject the system to an extended real-world drive in an open variety of conditions, accumulating enough kilometers to capture the full complexity and unpredictability of daily traffic. This is typically conducted by professional drivers with prototype or testing vehicles on public roads, and it is particularly suitable for attaining extensive KPIs of the system or component. The concept of endurance contrasts with the implicit subsampling or discretization of test cases in scenario-based testing: it attempts to reach those yet unknown edge cases or rare events that cannot be foreseen in any formal specification strategy. One common example of its unique usefulness is to help detect and minimize false positives or false alarm rates [Hakuli & Krug, 2014]. Endurance testing thus comprises the so-called Field Operational Tests (FOT), or tests with equipped vehicles on public roads.

However, as the variety is achieved through sheer volume, the efficiency and objectivity of test cases are lost. Accordingly, the time and cost increase immeasurably, especially when considering the targets of millions (or billions) of kilometers required for fully testing an autonomous vehicle. Yet, it is fundamental for any development process since it complements

scenario-based testing: by opposing its analytical reduction, the probability of encountering unpredictable problems and thus going beyond the known spatial state is much higher.

As standard practice, endurance testing results constitute the final approval of the SuT, as it offers the most realistic validation stage before production and deployment [Hakuli & Krug, 2014]. In any case, having these tests as a viable option ultimately relies upon the level of maturity of the project. The system needs to be fully integrated into the vehicle with an adequate level of proven performance, and the prototype vehicle itself in turn also needs to be ready for long-range driving. Since most functionalities, such as lane detection in this work, have a structural role in the automated driving system, there is usually a need to execute specific endurance tests on components. This is hardly feasible before the latter stages of the project.

2.1.2 Virtual Testing

In recent years, the exponential increase in technical specifications and legal requirements has pushed the minimum number of test cases far beyond road testing capacity. For instance, according to Wachenfeld and Winner [2016], 6.61 billion kilometers must be driven to encounter at least one critical incident, based on the average number of kilometers driven between two fatal incidents on German roadways. Therefore, they conclude that executing verification and validation of autonomous vehicles for public deployment solely through real test drives is likely infeasible.

Thus, the strategy has gradually expanded to include virtual testing in simulation environments, where typically the effectiveness of safety systems can be evaluated first in driving simulators and later on with actual real-world data [Hakuli & Krug, 2014]. Scenario-based testing facilitates this transition by providing clear and quantifiable parameters for test configuration and metrics for test coverage. But it also sheds new horizons for endurance testing since it enables the control and expansion of diversity while limiting costs for previously unattainable or unreasonable targets of driving kilometers.

In virtual test approaches, engineers typically configure an X-in-the-Loop architecture. This is part of a testing methodology that aims to integrate real-world components into a simulated environment. The element under assessment (”X”) may be a software or hardware component, a vehicle, or a model. In parallel, the simulation platform (”loop”) comprises virtual vehicle prototypes and their critical sub-systems capable of executing realistic vehicle behavior, the virtual environment, sensors, a communication stack, and a real-time co-simulation platform for deterministic computation [Moten, Celiberti, Grottoli, & van der Heide, 2018]. Connecting these elements to a simulation platform allows precise monitoring of the outputs, i.e., vehicle behavior, corresponding to a given synthetic input. Therefore, system performance can be insulated and assessed unequivocally. For instance, Software-in-the-Loop (SiL) testing involves executing software code in a simulated environment to examine its functionality without physical hardware. On the other hand, Hardware-in-the-Loop (HiL) testing incorporates actual hardware components—commonly, sensors, brakes, or electronic control units—into the simulation to evaluate their integration and response to the virtual environment (see Figure 2.2).

Figure 2.2 Camera-HiL demonstration from IPG Automotive [2024a].

Traditionally, on test tracks, even with the same conditions and procedures, there is no exact reproducibility due to the complex interactions that minimal exogenous influences have on the system. Virtual testing increases the reproducibility of tests and, in consequence, the confidence level of testing verdicts [Hakuli & Krug, 2014]. Furthermore, it adds scalability, immediacy, and automation, considering that thousands of test cases can run in parallel with distributed cloud infrastructures. Finally, it adds flexibility and control since massive test series are configurable with rigorous parametric variations.

2.2 Lane Detection Validation

As seen in V&V, the data limitations associated with validation also impact training and testing lane detection models. Pan et al. [2018] state that one of the reasons for assembling CULane was that the datasets available were insufficient to train their Spatial-CNN network. Composed of only a few thousand images taken in daylight with limited traffic (highway) and clear line markings, existing datasets rarely contained occlusions from other elements, fragmented sections, or signs of erosion. Instead, CULane was recorded in a dense Beijing scenario, including unfavorable lighting and weather conditions (see Figures 2.3 and 2.4). Thus, it provided a more challenging benchmark dataset for validating lane detection systems, with a volume 20 times larger than TuSimple. According to J. Li [2023], CULane is still the most requested benchmark by researchers in 2024, accounting for 63% of references in the literature.

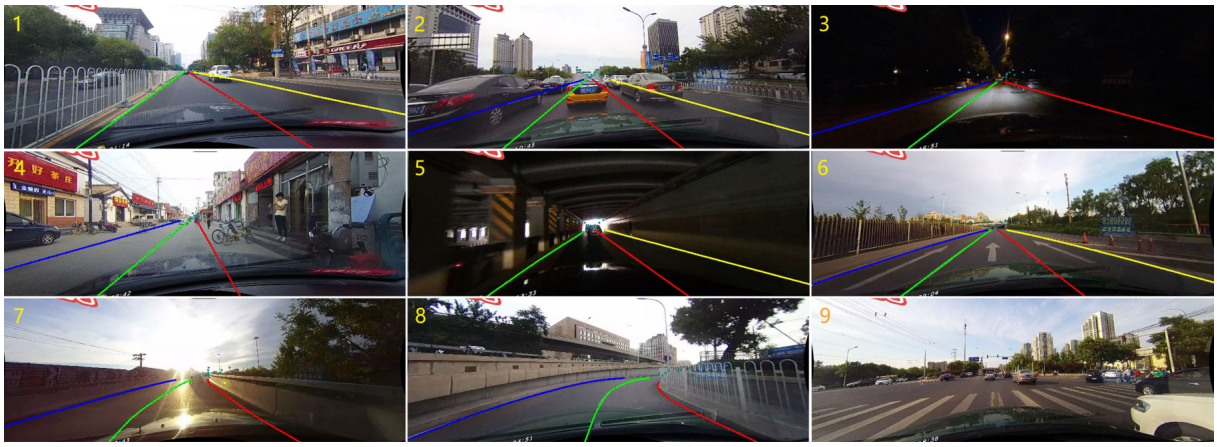


Figure 2.3 CULane dataset samples from Pan et al. [2018].

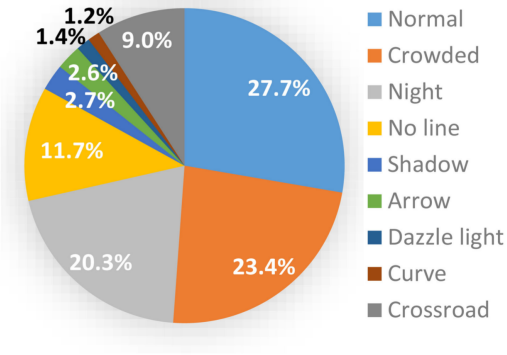


Figure 2.4 CULane dataset distribution from Pan et al. [2018].

Although several other large and robust datasets are now open source, all still require significant investment. Since typical datasets stem from in-car recordings of actual traffic on public roads and imply labeling frame-by-frame pixel-accurate lane marking points, they are both onerous and logistically complex. Furthermore, if the data specifications for a particular system change or expand, its development costs may increase immeasurably. On the contrary, synthetic datasets are much more cost-effective than real-world data because volume scales with a residual impact on effort and equipment. Also, since the annotation process in simulation is automated (rather than manual), it can be evaluated and assessed for quality and accuracy criteria. Overall, it offers control and degrees of flexibility, permitting changes in annotation requirements during the development cycle or according to different target classes.

Additionally, the degree of variability in the recorded data always constricts the learning model's capability. Regarding this, Zakaria et al. [2023] note that lane markings typically follow conventions specific to a country or region and that the ego-vehicle speed in real data collection usually ranges in low-flow traffic (up to 80 km/h), lacking higher-speed instances. In contrast, it is possible to tailor synthetic data and provide these specific environmental factors, plus adverse illumination, rare traffic events, and unusual road layouts. Fundamentally, it can support a

dataset curation strategy to control imbalances in training. For instance, if a dataset contains a 20/80 ratio between day and night or between highway and city scenarios, sheer volume will not be sufficient; the poor economy of the data will still affect the model's performance and thwart its validation. A robust V&V strategy must enforce the widest collection of testing variations within an efficient dataset. Thus, simulation environments are an effective source for complete coverage of the driving domain and a suitable degree of variance expected for training and testing.

Of course, the simulation-to-real domain gap in camera data is still considerable, even considering state-of-the-art rendering engines. The photorealism and diversity of artificial environments may be impressive. However, we should not underestimate the variance due to noise, unpredictability, or anomalies in a real-world driving dataset. Synthetic data may fruitfully expand on authentic data, yet entirely replacing its full complexity and richness will implicitly create invisible ceilings and arbitrary overfits for the generalization capabilities of a learning model.

2.2.1 Synthetic Data Generation

Regarding the methodology of this thesis, it is relevant to clarify the distinction between open-loop and closed-loop testing. In open-loop, the SuT receives inputs from the simulation but operates independently from it. As an illustration, the insulated software component would receive image or radar data and process its results without interaction with any other components, particularly without interfaces to the virtual vehicle's motion control component. On the contrary, in closed-loop testing, there is a continuous feedback loop in which the SuT receives inputs and directs its outputs back into the simulation, thus replicating automated vehicle behavior. The norm for V&V procedures is to establish a full closed-loop setup, which requires the complete ADAS architecture to be integrated into the virtual environment. However, for initial algorithm development, when motion control components for the virtual vehicle prototype are not yet ready, the only possibility is to set up an open-loop system.

Therefore, the goal of this work is to implement a pipeline for generating synthetic data using an open-loop testing methodology. In the given context, there is no complete ADAS architecture integrated or available, nor can such tasks be considered within the scope of activities. On the other hand, having a standalone lane detection model to test emulates the specific testing conditions for these functionalities in the initial phases of the V-model.

Currently, test engineers order extensive datasets of endurance driving on real roads to evaluate the performance of components based on deep learning models. Of course, since these models possess inherently opaque or emergent characteristics, it is not recommended to guarantee functional safety with scenario-based test coverage alone. To face these new challenges, Borg et al. [2018] propose that validation should significantly increase requirement specifications in scenario-based testing and both volume and diversity of data for endurance testing. However, due to the limitations imposed by system and vehicle maturity, as well as technical and financial constraints, it is onerous and complex to obtain sensor data for new benchmarks—especially those recorded with the exact specifications of the ADAS under de-

velopment. For instance, data to test camera-based perception algorithms should replicate the camera's intrinsic parameters (such as lens distortion model and field-of-view), as well as its mounting position on the vehicle. Thus, a dataset conceived for a specific project may quickly become obsolete if hardware-related prerequisites change. It is also not straightforward to add environmental or geographical diversity, particularly if it requires ground-truth labeling.

Thus, we intend to provide a virtual pipeline capable of generating endurance-based image data, and configurable to multiple sensor specifications with ground truth for lane markings. As a result, this pipeline ought to produce a benchmark dataset with sufficient circadian, climate, and road layout variety that aids the validation of lane detection algorithms in the early stages of the project.

Chapter 3

Implementation

3.1 Simulation Environment

As aforementioned, this work adopts UE5 as its simulation environment. The latest version of this engine has brought an unprecedented level of photorealism through the addition of visual components such as:

- **Lumen Global Illumination**—Real-time lighting (no lightmaps).
- **Nanite Geometry**—Millions of polygons in real-time.
- **Virtual Shadow Maps**—Soft shadows for detail streaming.
- **Temporal Super Resolution**—High-quality upsampling.

Epic Games has also released a free, downloadable open world as a demonstration: ”City Sample” (see Figure 3.1). It recreates a 1.6-km^2 modern metropolis with a dense traffic and pedestrian flow, supported by the MassAI system [Epic Games, 2024]. ”City Sample” will be the baseline project for this work since it is the most sophisticated driving simulation project made available as open-source in UE5. The programming interface comprises either Blueprints or C++ access. The former consists of a visual programming interface, usually considered ideal for rapid prototyping. If the project logic or its links to plugins are already implemented in Blueprints, which is the case with ”City Sample,” one must use these for all major changes. The implementation tasks are then performed exclusively in Blueprints to efficiently manage the high complexity of ”City Sample.”

UE5 provides the following standard methods for accessing simulation events:

- `BeginPlay`: Executes once when an actor is spawned.
- `Tick`: Executes frame-by-frame.
- `EndPlay`: Executes once the simulation closes or the actor is destroyed.

.

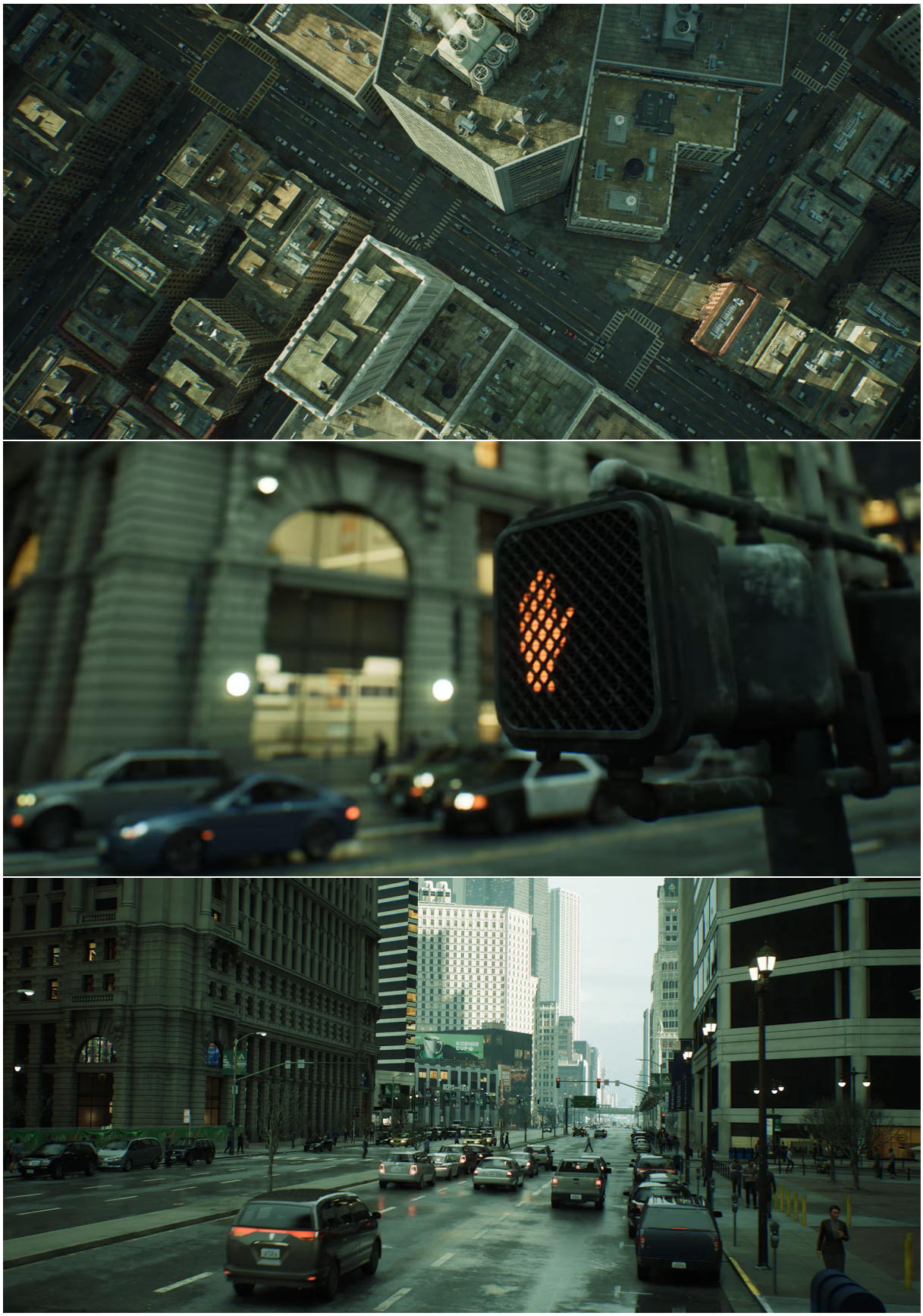


Figure 3.1 Stills from ”City Sample” project [Epic Games, 2024].

3.1.1 Ego-Vehicle Control

Establishing parametric control over current traffic participants is a prerequisite for setting up a synthetic data pipeline. Specifically, this involves integrating a newly created ego-vehicle agent into the MassAI traffic system and spawning it at any location in the city or at a predetermined point, according to the scenario parameters of the test case. The ego-vehicle must drive autonomously, blend in with traffic, and exhibit naturalistic behavior (follow traffic rules, keep a safe distance from other agents, and follow coherent routes). However, all static vehicles in the environment (parked cars) are unlinked to the MassAI execution system and cannot be set to autonomous driving. The MassAI vehicles in ”City Sample” are randomly spawned into the environment during runtime, which implies that the ego-vehicle also had to be defined/possessed at runtime.

Thus, to establish a MassAI vehicle with the POV of the first-person controller, the base class for all MassAI vehicles in ”City Sample”—`BP_Vehicle_Sandbox`—needs to be reconfigured to allow the setup of the ego-vehicle. Since the MassAI spawning process happens after the simulation launch, within a relatively fast but undefinable time interval, the possession of the vehicle cannot occur on the `BeginPlay` as well, but on the `Tick` event (see detail **#1** in Figure 3.2). This event, of course, executes itself on every frame and needs to be protected with a safeguard to guarantee that, in case any ego-vehicle instance is already under control, the possession method avoids endless repetition. When concluded, the execution will jump to spline detection (which will be described further in section 3.1.3); otherwise, it will continue with the ego-vehicle setup (see detail **#2** in Figure 3.2).

In parallel, to ensure the exact position in which every test case should start, a simple empty pawn (without mesh) is placed at a given road location before launch as a static object in the environment. At runtime, the current world location distances of all `BP_Vehicle_Sandbox` instances to this empty pawn are verified. The next vehicle to cross this pawn will be chosen as the ego-vehicle out of all the MassAI vehicles driving autonomously through the traffic environment.

Before advancing with further processing, it was convenient to filter out all static/stopped vehicle instances with the custom method `Check Speed`, to save runtime performance from the subsequent steps (see detail **#3** in Figure 3.2). After ensuring that the vehicle instance is in motion, the next task is to acquire all pawn actors with the ≪PositionTestCase≫ tag, previously placed in the environment, to trigger the test case launch on specific locations (see detail **#4** in Figure 3.2). With a list of all pawn actors containing the predefined tag and guaranteeing that the list is not empty, it is now possible to enact a loop with `For Each Loop with Break` (see detail **#5** in Figure 3.2). This loop will query each element for world location and check its distance towards the vehicle instance’s mesh location. If the distance is lower than a sensible threshold (200 cm), the given vehicle instance is now circulating precisely in one of the selected starting points (see detail **#6** in Figure 3.2). The final step is to activate the previously mentioned safeguard with the possession status boolean flag and advance for camera and view controller configuration.

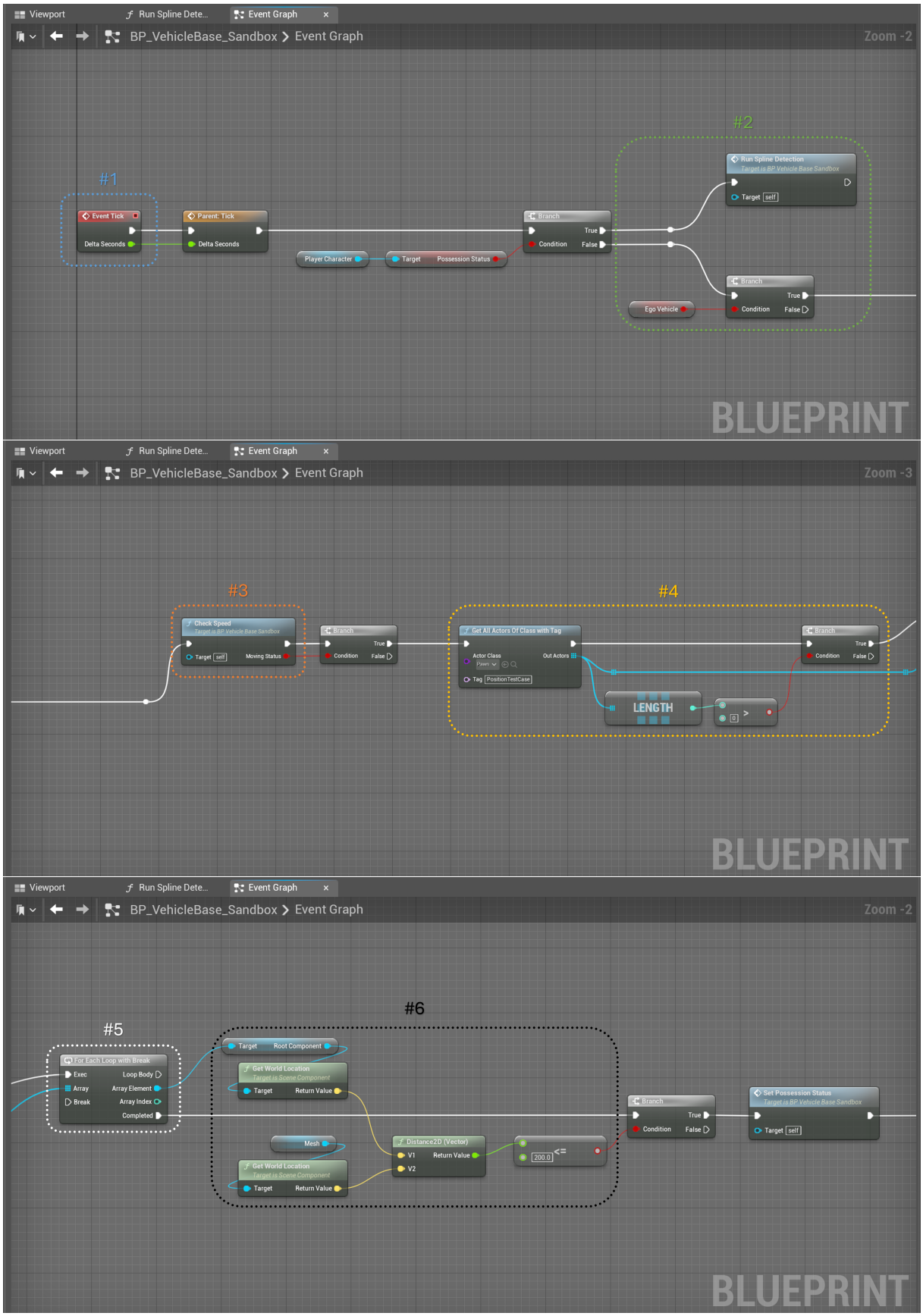


Figure 3.2 Blueprint nodes and execution flow for ego-vehicle control.

3.1.2 Camera Sensor

The camera sensor incorporates camera model specifications from reference vendors Valeo [2024] and Continental Automotive [2024] as automotive standards, considering the specific lens and sensor settings:

1. **Lens**:
    - Focal Length—12.0mm;
    - Aperture—f/2.8mm;
2. **Sensor**:
    - Width—36.0mm
    - Height—20.25mm
    - Target Resolution—1920x1080
    - Aspect Ratio—1.778

The stated f/2.8 aperture and 12 mm focal length intend to reproduce a wide-angle lens with a quick aperture. A broader field of vision is usually preferable to detect active and dynamic entities such as vehicles and pedestrians. A fast aperture provides better low-light performance in dark/unfavorable lighting circumstances. UE5's blueprint class `BP_CineCameraActor` allowed fine control to implement these camera parameters (see Figure 3.3). This process also includes replicating the approximate installation position of the CULane camera to ensure that the horizon line matches the expectations of the lane detection model training. From empirical observation, the top vertex of the lane splines should reach 45%-50% of the image height.

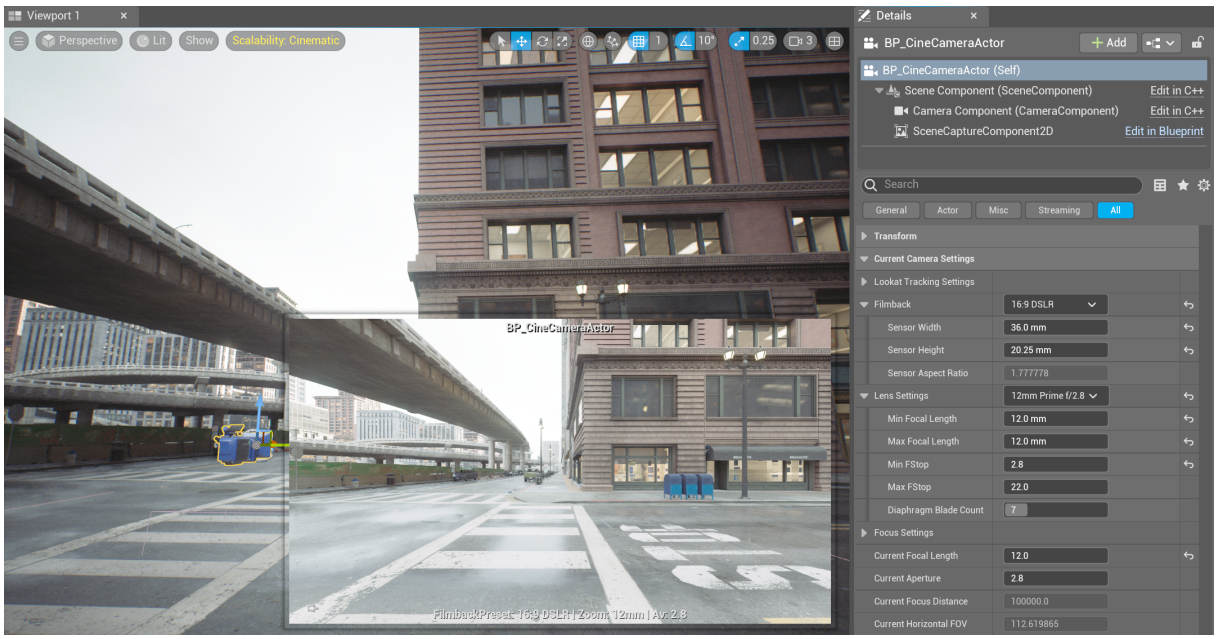


Figure 3.3 Camera sensor specifications in the virtual environment.

The subsequent step is to create the cinematic camera sensor and place it as an actor in the environment. Since the vehicle setup in Blueprint implies that it is only spawned at runtime, the camera actor must be mounted to the vehicle after its possession on the `Tick` event. The `Attach Actor To Component` node allows to incorporate an actor as a component in another actor. The camera actor is then attached to the `Spring Arm` component in the vehicle (see detail **#1** in Figure 3.4). After, the pivotal `Set View Target With Blend` node effectively determines this camera as the player controller's POV (see detail **#2** in Figure 3.4). The final step is initializing a series of variables, including a custom HUD for other functionalities (see details **#3** and **#4** in Figure 3.4), such as the option to include a signature and a copyright symbol on the viewport, and hence on the final video export.

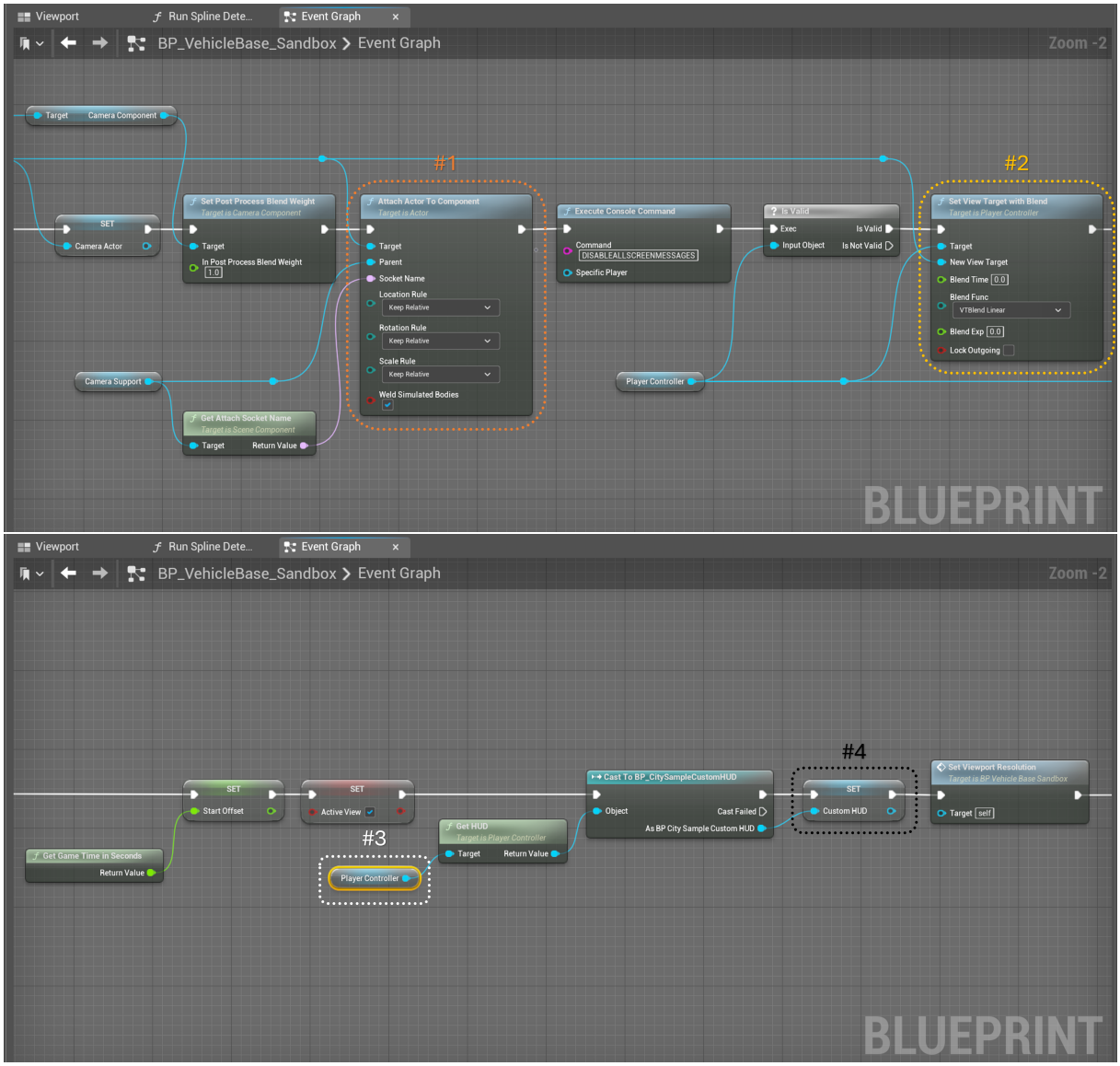


Figure 3.4 Blueprint nodes and execution flow for the camera sensor.

3.1.3 Ground Truth Labeling

For ground truth, the texture of each lane marking in the city's road network was juxtaposed by a spline component adapted to its exact position, orientation, and length. Overall, this approach required to fit hundreds of splines (see **2nd** panel in Figure 3.5). Applying splines to the road base meshes manually is remarkably time-consuming, but there is no programmatical alternative. The road textures in the "City Sample" environment are the composite result of multiple layered, fragmented, and overlapping semi-transparent textures. The road layout, which contains the road textures, is a unified ground object with a grid of nested square lots that support both building property and roads indistinctly. As a result, there are no precise coordinates for the lanes marking textures, nor is there a mesh that precisely represents the geometry of a road, allowing for the extraction of a midpoint, for example. Regardless, the validation of the synthetic data depends critically on the precision of the ground truth labeling. Therefore, it was vital to adopt multi-point splines and carefully adjust each spline point to the middle axis of each lane marking (see **3rd** panel in Figure 3.5). Particularly since in "City Sample," some road markings are slightly misaligned or off-centered to mimic an old or gritty urban asphalt.

In terms of ground truth extraction, it was necessary to establish an efficient and performative method for collecting the lane markings that immediately precede or are immediately relevant to the ego-vehicle, considering there are several hundred splines in this 1.6 km^2 area. At runtime, if the ego-vehicle is already possessed, a detection method should filter the visible splines in the current route, starting by iterating over all splines (see detail **#1** in Figure 3.6). The spline detection method first establishes a forward vector with a longitudinal magnitude of 50 meters and a perpendicular vector with a lateral magnitude of 2 meters (see detail **#2** in Figure 3.6). These ranges define the bounding limits for collecting path-relevant splines and, thus, implicitly determine the maximum perception horizon for ground truth in posterior validation. If a given system requirement demands validation for lane detection in superior ranges, this method should receive appropriate parameters before further test cases. Within this lateral range, there is also a tacit decision to extract only left and right ego lanes, excluding adjacent lanes, which would carry additional (and significant) overhead on runtime processing and dataset preparation.

Then, it filters splines within a squared distance equivalent to the longitudinal range. After, it filters splines that match the orientation of the ego-vehicle within a 60° angular radius to both allow the correct inclusion of lanes tailored to road curvatures and exclude perpendicular lanes at intersections (see details **#3** and **#4** in Figure 3.6). Within the subset of path-relevant splines for each frame, it extracts a series of 3D points from the spline within a 50-cm sampling interval—through `Get Location at Distance Along Spline`. Additionally, it will check if each 3D point is inside a virtual area of interest, using its dot product with the forward vector and the perpendicular vector concurrently. To account for occlusions between lane marking points and the ego-vehicle POV, all 3D points are also traced back to the camera to evaluate if there are interposing hits via `Line Trace by Channel`.

Figure 3.5 Total area and fine detail for lane marking ground truth labeling.

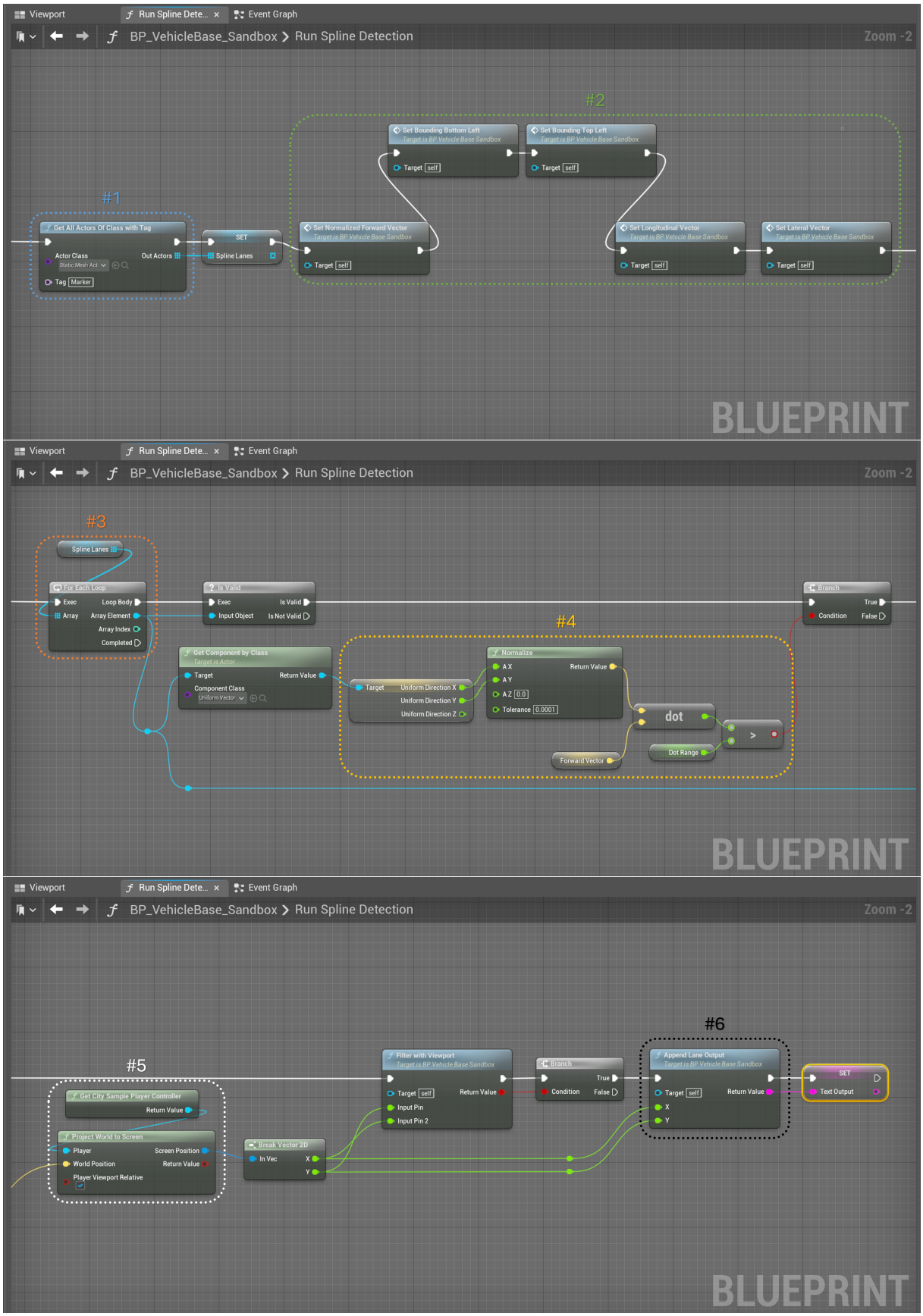


Figure 3.6 Blueprint nodes and execution flow for spline detection.

Finally, it projects these 3D world coordinates onto the 2D viewport reference system with `Project World to Screen` (see detail **#5** in Figure 3.6). One last safeguard is to ensure that these resulting 2D points are within the resolution limits with `Filter with Viewport`.

The result is a set of 2D positions determining the visible lane markings on the ego-vehicle's camera viewport. These 2D coordinates are continuously appended to a lane output list at each frame (see detail **#6** in Figure 3.6). On the `End Play` event for the given ego-vehicle object, all lists are converted into string format and stored in a text file. This file will contain a header for each frame with its elapsed timestamp and the set of respective lane boundary 2D coordinates. Figures 3.7 and 3.8 illustrate, respectively, a schematic view of the vectors of extraction and example lane marking points.

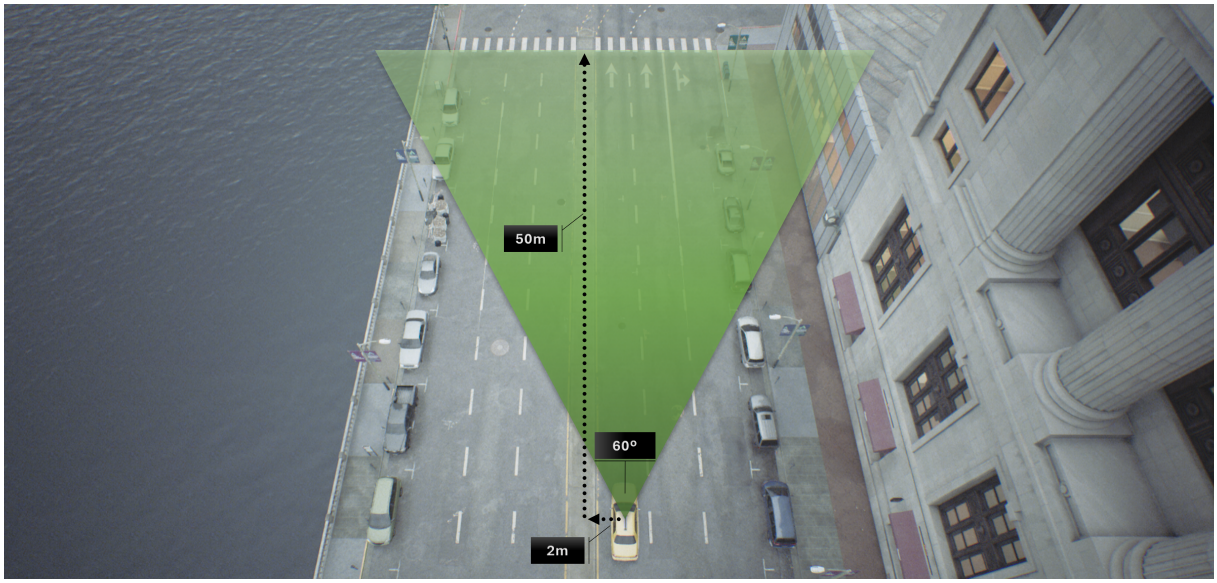


Figure 3.7 Diagram of the area of interest for ground truth extraction.

```
Timestamp 0.134518
Ego Lane Boundary  934.30249 637.848999
Ego Lane Boundary  930.785095 639.638489
Ego Lane Boundary  926.235474 641.953186
Ego Lane Boundary  920.122925 645.063049
Ego Lane Boundary  910.385254 649.474915
Ego Lane Boundary  897.619019 656.162231
Ego Lane Boundary  876.123291 667.563293
Ego Lane Boundary  986.194397 636.433533
Ego Lane Boundary  989.467407 637.81073
Ego Lane Boundary  992.760315 639.545837
Ego Lane Boundary  995.653809 641.797424
Ego Lane Boundary  1002.11084 644.768677
Ego Lane Boundary  1007.574707 648.981384
Ego Lane Boundary  1018.498535 655.27478
Ego Lane Boundary  1036.740601 665.78418
Ego Lane Boundary  1073.279297 686.869568
Ego Lane Boundary  1191.982178 750.484436
```

Figure 3.8 Output sample of lane boundary markers for a single frame.

3.1.4 Video Recording

While the Sequencer in UE5 is generally recommended for recording in simulation, its use with the exceptionally complex and computationally demanding ”City Sample” environment proved to be highly unstable and impractical.

Alternatively, the ”MSC Screen Recorder” plugin [Brito, 2024] enables real-time screen/viewport recording with minimal impact on runtime performance. For this project, it is set to 25 frames per second with an MP4 output format and a default 8-bit depth per pixel. Its usage was also notably easy: start screen capture at `Begin Play` and finish screen capture at `End Play` event via the interface to the plugin in Blueprint (see Figure 3.9). The resulting videos constitute one of the outputs of the synthetic data pipeline.

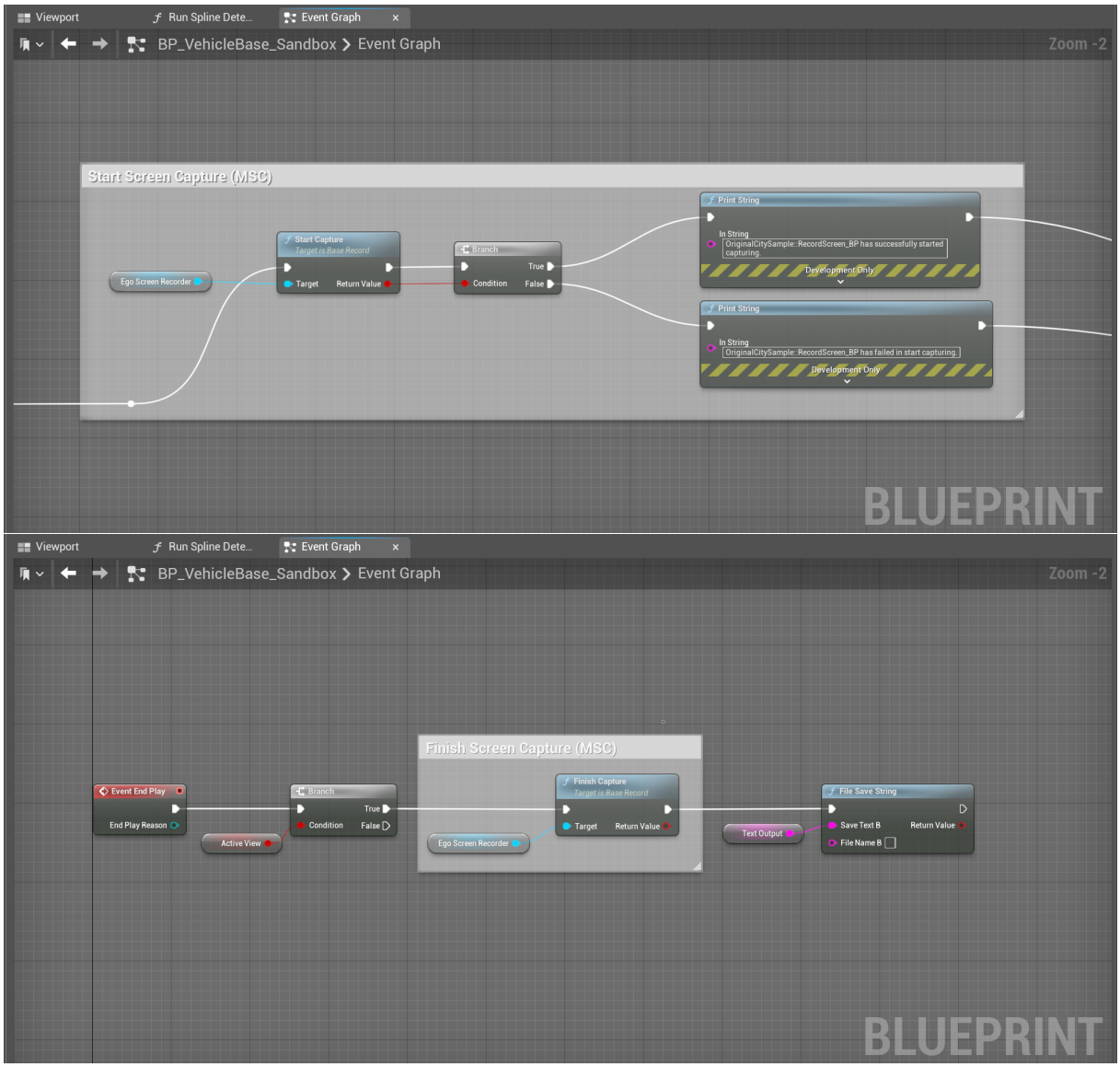


Figure 3.9 Blueprint nodes and execution flow for video extraction.

3.1.5 Dataset Preparation

For compatibility purposes with existing benchmarks, this work adopts the CULane dataset as a reference model for converting the UE5 synthetic data. This approach shall guarantee that any model trained or tested on CULane can seamlessly incorporate the new synthetic dataset.

CULane comprises 133,235 images captured from six vehicles operating in Beijing over a 55-hour period. The dataset is divided into training (88,880 images), validation (9,675 images), and test (34,680 images) sets. Each image includes manual annotations of all traffic lanes, encompassing both spline segmentation (as illustrated in Figure 3.10) and pixel-wise 2D point labeling. The pixel-wise labels consist of a sequence of 2D coordinates, spaced 10 pixels apart on the (y)-axis. The dataset labels up to four lanes: left adjacent, left ego, right ego, and right adjacent. In cases of fragmented or partially occluded lane markings, the dataset incorporates context-based estimations.

The key specifications of CULane relevant to this research are as follows:

1. **Image**:
    - Target Resolution—1640x590
    - Aspect Ratio—2.779
2. **Ground Truth**:
    - Pixel—2D coordinates;
    - Segmentation—Polynomial splines;

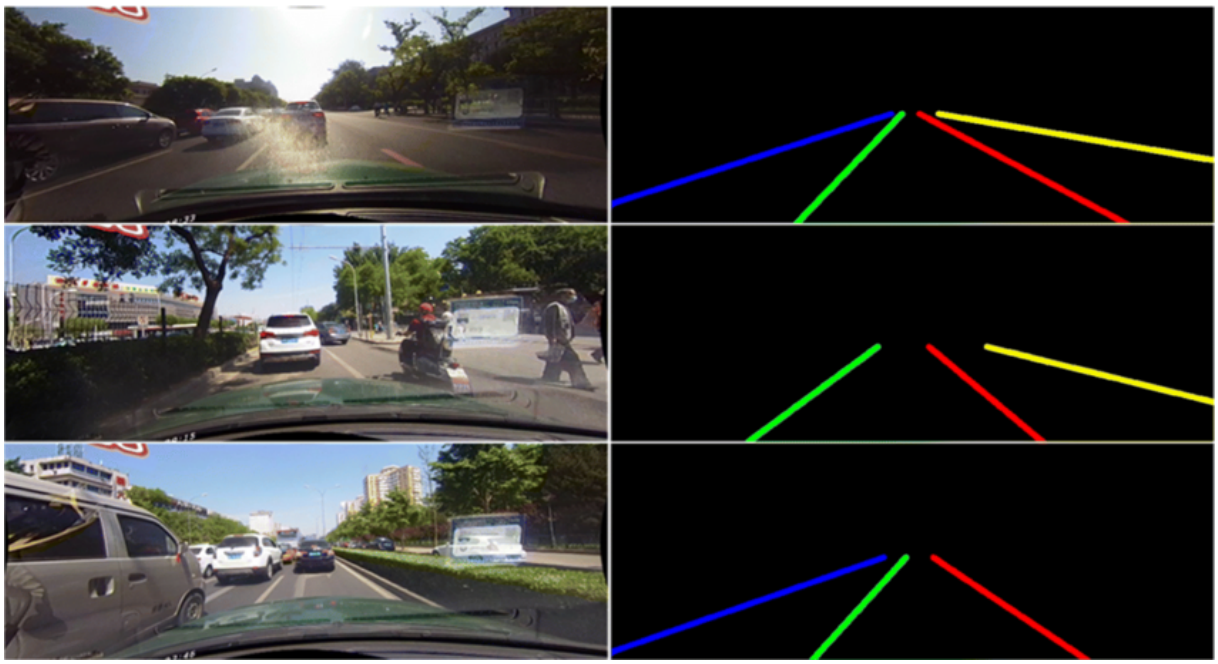


Figure 3.10 Frames and respective segmentation labels from CULane [Yang et al., 2022].

To ensure consistency with CULane, the ground truth markings from the synthetic dataset should always be extended downwards to the y-axis' root (even when partially occluded by the ego-vehicle). Since the synthetic lane marking points derive from the 3D world coordinates' projection onto a 2D coordinate system, they exhibit a vanishing distance towards the horizon and not a constant 10-pixel interval. Additionally, the ground truth extracted from UE5 (see preceding Figure 3.8) does not identify or separate lanes, nor do the points necessarily conform to a polynomial spline, due to possible outliers. On the other hand, most lane detection models expect 2D coordinate points and segmentation labels for each lane, with segments set at 30-pixel width in most benchmarks [Behrendt & Soussan, 2019; Pan et al., 2018]. Finally, the synthetic dataset contains metadata to configure the ratio of training, testing, and validation subsets of frames, instance segmentation, and pixel point labels.

Thus, a series of automated preprocessing steps should apply to the image/video and ground truth acquired from UE5. This preprocessing pipeline will yield a synthetic dataset that can integrate the same standard training, testing, and validation applicable to the CULane benchmark. In other words, the generated dataset will be technically equivalent to the reference CULane benchmark dataset for any lane detection model.

- **Image processing**:

  1. Check that the extracted video contains valid frames for all ground truth markers and ignore any video frames that do not match with labeled lanes;
  2. Trim the frame resolution with an aspect ratio proportional to the specified parameters of the reference benchmark dataset while adjusting 2D coordinates with the corresponding offset;

- **Ground truth processing**:

  1. Identify whether the current subset of timestamp points contains single or double lanes, with an approximate midpoint threshold;
  2. Separate the lane marker points by left and right ego lane based on the slope of all y values, ordered relative to x;
  3. Perform a linear regression on the existing points to extract a slope and obtain an intercept of x with $y = 0$ to extend all lanes to the base of the frame;
  4. Perform a polynomial regression on the aggregated points (including the previously calculated x root) to obtain a more realistic lane model that excludes outliers and allows, e.g., accurate lane curvatures;
  5. Interpolate a series of points from this polynomial model, starting from the lowest to the highest y value, maintaining a 10-pixel interval;
  6. Write the ground truth markers for each frame in an individual text file;
  7. Draw segmentation markers for each image in an individual image file;
  8. List subsets for training, testing, and validation in the metadata file;

This project collected 44272 frames from multiple video segments for nine established test case scenarios in UE5. Each frame has a resolution of 1920x690, and the video runs at 24 frames per second. The size of this dataset corresponds to 33% of the complete CULane and is significantly larger than its original testing set with 34680 frames [Pan et al., 2018]. Figure 3.11 illustrates three example frames and corresponding polynomial spline segmentation. Complete coding implementation of this preprocessing pipeline is available in Annex A (Figures A.1, A.2, A.3, A.4, A.5, A.6, and A.7).

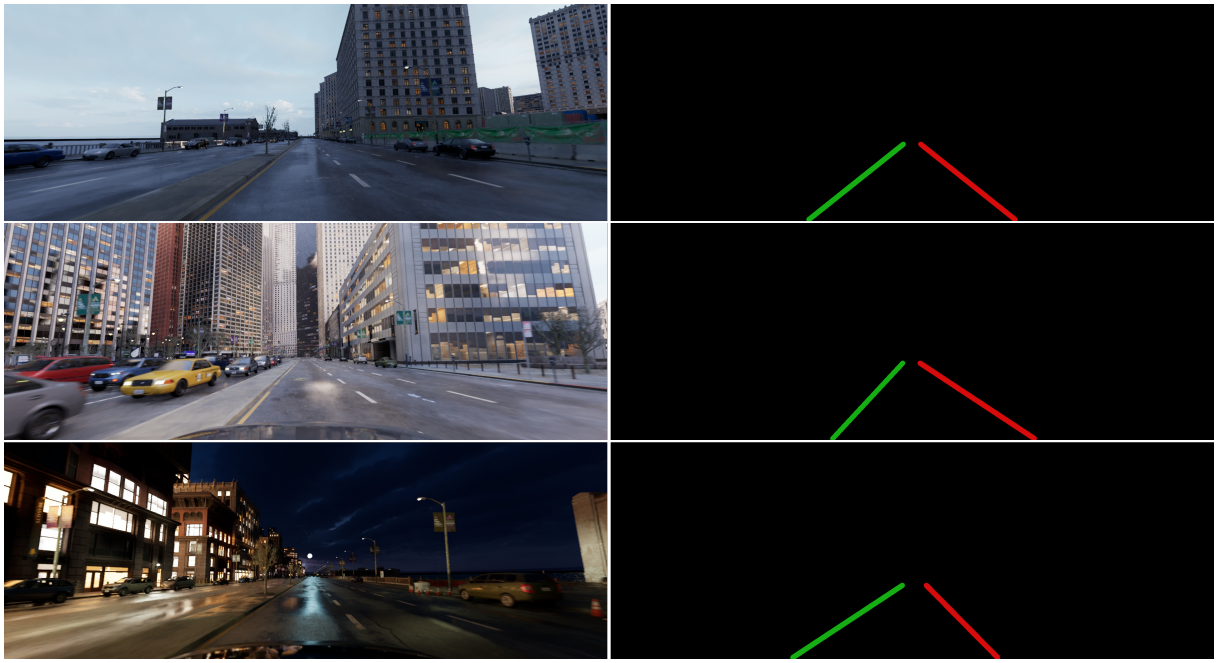


Figure 3.11 Frames and respective segmentation labels from the UE5 synthetic dataset.

3.1.6 Test Cases

As mentioned, our goal is to provide a validation framework for endurance-based testing, adjustable to various sensor specifications, with ground truth information for lane detection. Its primary outcome will be a curated synthetic dataset that includes variations in circadian phases, climate conditions, and road layouts. While traditional endurance testing often prioritizes volume or duration, this research introduces a test case methodology to improve dataset diversity and assess SuT performance beyond any limitations of benchmarks like CULane.

Test cases reflect dawn, noon, and nighttime combinations with challenging lighting and weather circumstances. Prototypical images from each of the nine test cases are shown in Figures 3.12, 3.13, and 3.14. It features foggy and glaring conditions to represent scenarios with significantly reduced contrast in road markings. Fog-filled scenes, in particular, are missing in CULane, as seen with its dataset distribution in preceding Figure 2.4 from Chapter 2.

Figure 3.12 Example frames for test cases at dawn (with clear, foggy, and glaring light).

Figure 3.13 Example frames for test cases at noon (with clear, foggy, and glaring light).

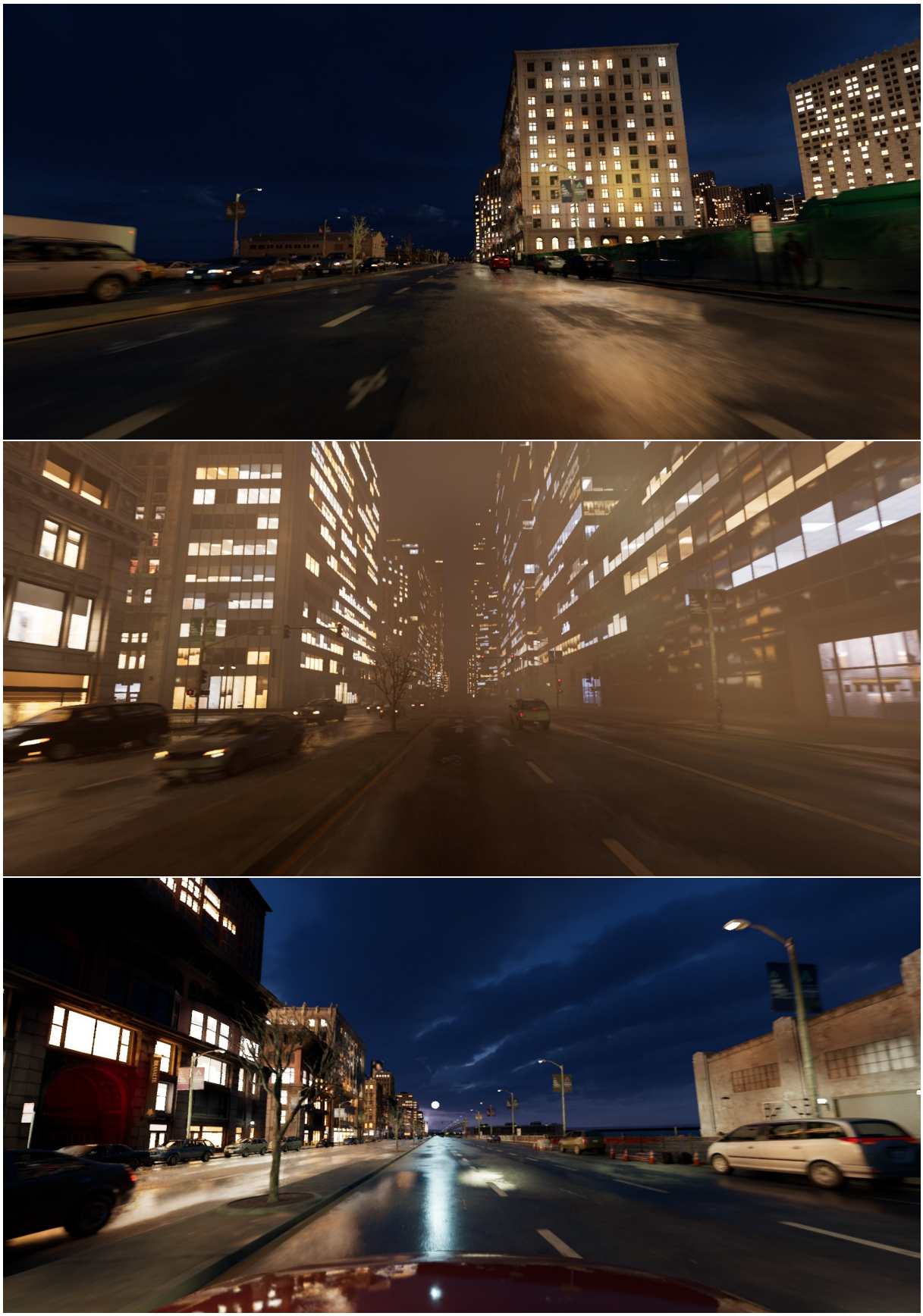


Figure 3.14 Example frames for test cases at night (with clear, foggy, and glaring light).

Chapter 4

Results & Analysis

4.1 System-under-Test

4.1.1 CLRNet

Using a novel, cross-layer network architecture, CLRNet achieved the highest F1 score (80.47%) in the CULane dataset, establishing itself as the state-of-the-art method for lane detection in 2022 [PapersWithCode, 2024]. This performance was complemented by its exceptional results on the TuSimple and LLAMAS datasets, where it achieved a maximum accuracy of 97.89% and 96.12%, respectively. In parallel, its availability as an open-source implementation [Zheng & Huang, 2022], unlike other leading lane detection models, further supports the decision to select CLRNet as the SuT for this project. Regarding real-world deployment, no performance data has yet been published until 2024.

Its authors noticed that while lane detection performance is typically strong in controlled environments, most models experience significant problems with unusual road markings, heavy traffic, or highly blurred lane markings caused by unfavorable lighting conditions [Zheng & Huang, 2022]. These difficulties highlight how important contextual information is to lane detection. In response, CLRNet includes a multi-layer feature extraction strategy, starting with high-level semantic features to capture global scene context and then progressively refining lane detection using low-level, fine-grained details.

An essential component of this cross-layer architecture is the Region of Interest (ROI) module, termed ROIGather. ROIGather captures road boundaries, adjacent lanes, and other important environmental factors. The network subsequently iterates through multiple image layers (see Figure 4.1). This cross-layer design enhances the model's ability to maintain accuracy in dynamic or cluttered driving environments or to interpret lanes when the visibility of its features is compromised. Overall, it offers greater robustness since it remarkably resembles human perception in adverse visibility conditions: when a lane is either occluded, obscured by darkness, or glare, the driver's focus immediately shifts, raising attention from the lane itself to the global layout and landmarks of the road.

The authors also introduce Line Intersection over Union (LIoU) loss—a regression technique tailored for lane predictions. In contrast to the pixel-level loss functions typically employed

in lane detection models, LIoU measures the complete lane geometry, optimizing accuracy in lane position predictions. This approach guarantees higher alignment between expected lanes and actual road markings, particularly when challenging road conditions result in partially obscured or distorted lanes.

The code in Annex B, adapted from [Zheng & Huang, 2022], defines the configuration used in this work for testing CLRNet, including network architecture and validation parameters (Figures B.3, and B.4).

- **Network Configuration**:
  - The backbone is a pre-trained ResNetWrapper (resnet34) architecture.
  - The network head includes three refinement layers, 192 predefined anchor points—as initial lane hypotheses (priors), and a fully connected hidden layer with 64 units. It outputs 36 sample points per lane with a 10-pixel interval.
  - The neck uses a Feature Pyramid Network (FPN) with input channels [128, 256, 512], output channels of 64, and three output layers. Attention mechanisms are disabled.
- **Loss Weights**:
  - Intersection over Union (IoU) loss is set at 2.0.
  - Classification loss (CLS) is set at 2.0.
  - Lane localization (XYT) loss is set at 0.2.
  - Segmentation loss is set at 1.0.
- **Image Preprocessing**:
  - The images are normalized using [103.939, 116.779, 123.68] as the mean and [1.0, 1.0, 1.0] as the standard deviation.
  - The original image dimensions are resized from 1920x690 to 800x320 pixels (the top 270 pixels are cut off).
- **Validation**:
  - The confidence threshold for lane predictions is set to 40%.
  - The Non-Maximum Suppression (NMS) threshold is set to 50%, which determines the degree of overlap allowed between different lane predictions before suppression.
  - The model is set to handle 2 lane classes (left ego and right ego lane).
  - The ideal lane width is set at 30 pixels.
  - The IoU threshold is initialized at 50% and iteratively applied within the range 50%—95%.

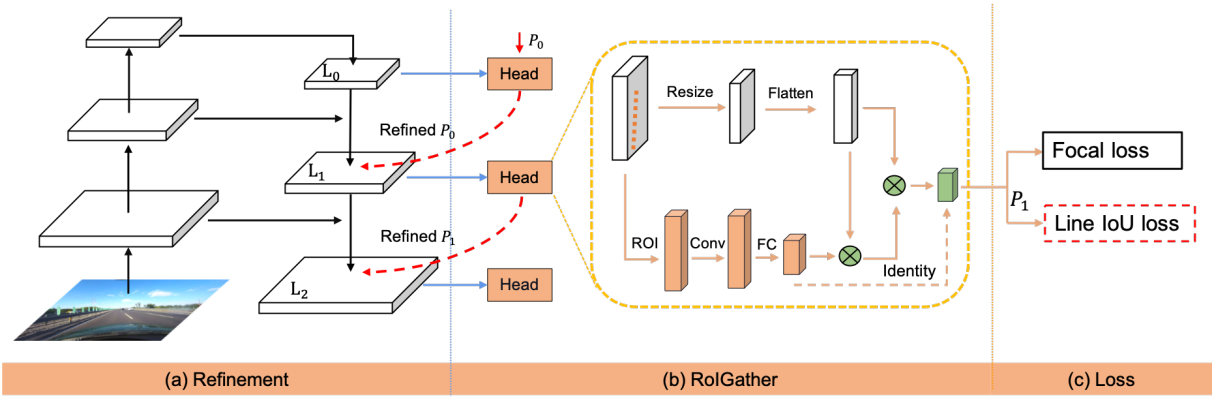


Figure 4.1 CLRNet model architecture from Zheng et al. [2022].

4.2 Metrics

Zheng et al. [2022] apply the official scoring method of the CULane benchmark to CLRNet, thus ensuring consistency in lane detection results versus other models for the same benchmark. Its source code is available in Annex B (Figures B.5, B.6 and B.7).

The CLRNet performance evaluation primarily measures the F1 score, a balance between precision and recall (see Equation 4.1). Precision in this context refers to accurately predicted lane points out of all predictions, while recall represents the proportion of accurately predicted out of all the ground truth lane points.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN} \tag{4.1}$$

To determine accuracy, each predicted lane on every frame is compared to the correlative ground truth lane segments, with an Intersection over Union (IoU) beyond a specified threshold indicating a true positive prediction. To properly execute the comparison between lane predictions and actual lane markings, both sets of points must interpolate as continuous curves—a requisite that had been already suited for the ground truth lane marking coordinates as part of the data processing steps in Chapter 3. The interpolation allows to sample predefined y-values on a 10-pixel interval and calculate the corresponding x-values from its polynomial spline, thus composing two sets with a definite number of discrete points (ranging from zero detections to the network's maximum output of 36 sample points). Both polynomial splines (lane detections and ground truth points) are then drawn as binary masks with a segment width of 30 pixels on an image grid, as specified in the validation settings above. The ratio between the overlapping area of both lanes and their union will output the IoU (see Equation 4.2). IoU thresholds range from 50%—95% to assess the model's performance under different tolerance values.

$$IoU = \frac{Area\ of\ Overlap}{Area\ of\ Union} \tag{4.2}$$

4.3 Results

CLRNet's pre-trained ResNet-34 model checkpoint [Zheng & Huang, 2022] is selected for inference with the synthetic dataset. In general, the model's performance slightly degrades compared to its original results on the CULane benchmark (see Table 4.1). The maximum F1-score (79.50%) occurred with a clear sky at dawn, indicating the importance of high ambient illumination levels for lane recognition. All test cases under a clear sky delivered values (79.50%, 74.56%, and 71.08% at dawn, noon, and night, respectively) that fell within a fair range of the mF1-score on CULane (79.73%) for the CLRNet ResNet-34 model [Zheng et al., 2022]. Regardless of the time of day, the most strenuous test conditions—fog and glare—expectedly produced the lowest results. The impact of nighttime testing was less pronounced because of the urban environment's bright street lighting. As a note, our analysis constrained the model to a two-lane maximum detection threshold.

Table 4.1 also shows that precision values generally exceeded recall, consistent with the necessary balance for lane detection functionality. For road deployment, false positives are more dangerous than false negatives since detecting incorrect lanes may lead to erratic trajectories and diverge the vehicle off-road or into adjacent traffic, while missed instances can still be recoverable.

Test	Time	Weather	Frames	Precision	Recall	F1-Score
1	Dawn	Clear	4505	77.37	81.74	79.50
2	Dawn	Fog	5605	79.98	46.91	59.13
3	Dawn	Glare	5667	62.26	45.33	52.46
4	Noon	Clear	4360	77.97	71.44	74.56
5	Noon	Fog	4517	75.87	43.35	55.17
6	Noon	Glare	5767	76.42	50.20	60.60
7	Night	Clear	4395	74.18	68.23	71.08
8	Night	Fog	4391	78.79	57.87	66.73
9	Night	Glare	5065	71.37	66.89	69.06
Total	-	-	**44272**	**74.63**	**58.43**	**65.54**

Table 4.1 Performance of the reference CLRNet model on synthetic data.

Figures 4.2, 4.3, and 4.4 illustrate the most relevant findings. Lane marking edges and other road markers are dim under virtual fog conditions, which causes irregular and oscillating lane detection. Furthermore, harsh sunlight reflections on the pavement pose the most severe challenge to the model. Here, contrast loss is nearly complete, as the road surface reflects as brightly as the color of its lanes. Both scenarios are well-known for their difficulty in real-world conditions, stressing the image sensor's dynamic range to the point of complete blur or saturation.

Figure 4.2 The model often inferred lane markings from parking boundaries.

Figure 4.3 The model misinterpreted numerous frames in fog-filled scenes.

Figure 4.4 Glare from direct sunlight creates the most severe challenges.

Multiple authors have confirmed the impact of complex weather conditions in traffic for object detection (fog, glare, rain, and snow) and reviewed the strategies adopted for improving performance [Arthi, Murugeswari, & Nagaraj, 2022]. Even though CLRNet enhances feature detection with contextual information, these scenarios still hinder its results considerably, which may stem from its rarity in the original CULane training set.

4.4 Sim-To-Real Domain Gap

By examining CLRNet's performance in synthetic data compared to the original dataset—before further adaptation measures—we proposed quantifying the raw domain gap between CULane and our experimental benchmark. Given that the best test case score obtained using the synthetic dataset closely matched the real-world dataset, the domain gap in simulation may appear to be already negligible. However, this score occurs in clear weather conditions. In fact, there is a significant variance between test cases. The aggregated results still show an average 14.19% gap versus the CLRNet ResNet-34 model's performance on real-world data (79.73%). This discrepancy is partly due to the different testing scenarios between CULane and our synthetic dataset, particularly the challenging conditions with fog and glare.

On the other hand, image rendering attributes and camera model properties of UE5 will also affect, to a certain degree, transferability from the synthetic to a real-world domain. Finally,

the original camera specifications—undisclosed in CULane—should also be considered a relevant factor in setting up any camera-based simulation. Regardless, it is expected that a model neither trained nor optimized for its target domain will inevitably suffer a drop in performance proportionate to the degree of dissimilarity from its source domain. Such a dissimilarity does occur between CULane and our synthetic dataset.

4.4.1 Synthetic Discriminator

To further examine the hypothetical domain gap inferred from CRLNet's performance drop between real-world and synthetic datasets, this research establishes a convolutional neural network (CNN) architecture acting as a "synthetic discriminator." Preprocessed road images are fed into this discriminator, which then categorizes them as pertaining to the artificial UE5 environment or the real-world CULane dataset. Table 4.2 describes the architecture of the discriminator.

Layer	Type	Parameters	Activation
Input	Input Data	1462X526	-
Conv2D	Convolutional	32x5x5	ReLU
MaxPool2D	Pooling	5x5	-
Conv2D	Convolutional	64x5x5	ReLU
MaxPool2D	Pooling	5x5	-
Conv2D	Convolutional	32x5x5	ReLU
MaxPool2D	Pooling	5x5	-
FullyConnected	Dense	512	ReLU
Dropout	Regularization	0.8	-
FullyConnected	Output	2	Softmax

Table 4.2 Minimalist CNN architecture for the synthetic discriminator.

The network utilizes three convolutional layers with ReLU activation and max-pooling for feature extraction, followed by fully connected layers with dropout for regularization. The final layer employs a softmax activation for binary classification: real versus synthetic. A subset of 5000 frames from CULane and an additional 5000 frames from the synthetic dataset are combined to create the training set for this artificial discriminator, which totals 10,000 images. Separately, another 1000 randomly chosen images from both datasets are the test set. The goal of the intentionally simplified network architecture is to assemble a neutral benchmark for the degree of similarity between synthetic and real images. Considering the high diversity of CULane, the expectation was that the discriminator's performance would likely capture the limited or recurring background patterns in the nine virtual test scenarios. The small filter sizes and low number of units are thus an attempt to avoid overfitting the synthetic data. The implementation code of this discriminator is available in Annex C (Figures C.1, C.2, and C.3).

The synthetic discriminator shows a remarkable performance and achieves an accuracy of 95.48% during training (see Figure 4.5). In the testing phase, it achieves results above 94%, which naturally leads to the conclusion that the images from both datasets (real and synthetic) are easily distinguishable, at least for this type of CNN architecture. Examination of the training dataset confirms the prior expectations. The synthetic dataset only contains nine test case scenarios, which determine limited lighting patterns, road layout, and urban infrastructure. On the other hand, the real dataset derives from 55 hours of driving in Beijing, reflecting variability on a significantly larger scale. It is the most probable hypothesis for the notable performance of the discriminator. Naturally, from this project's perspective, the ideal outcome would be to demonstrate the high level of realism of the virtual environment by showcasing a modest performance of the synthetic discriminator. In terms of future measures to shorten this differentiation, one crucial approach is to increase the number of test case scenarios in UE5, thus bringing the two data distributions to similar scales of variability.

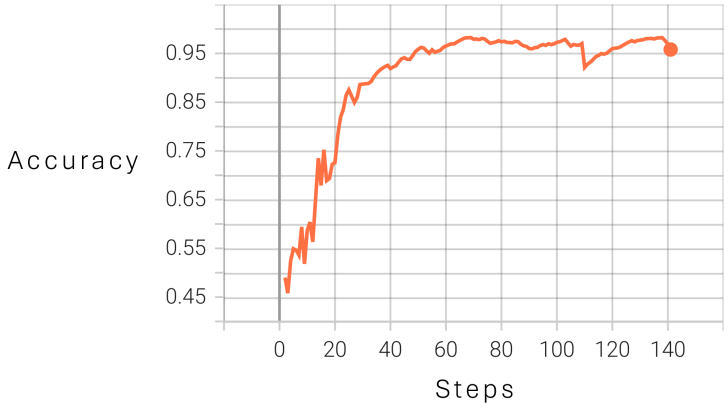


Figure 4.5 Training progress for the synthetic discriminator.

4.4.2 Domain Adaptation

Many authors have studied the problem of applying models to domains other than their source domain, i.e., their training and test data sets. Switching domains usually leads to a significant performance drop as they are part of distinct data distributions. Techniques for bridging source and target distributions fall under Domain Adaptation (DA). The same challenge arises here between real and synthetic domains. Our previous analysis led us to conclude that the degree of photorealism in each image is not yet fully mature, and, importantly, the degree of variability and complexity within the synthetic dataset is still quite limited.

Abramov, Bayer, and Heller [2020], for instance, demonstrated how classical unsupervised techniques such as Feature Distribution Matching (FDM) and Exact Histogram matching (EHM) can, to some extent, improve the similarity and, consequently, detection performance between synthetic and real-world datasets. They propose aligning color histograms' mean/covariance of source images towards the target domain. Results from domain adaptation between well-known real (KITTI, Cityscapes) and synthetic benchmarks (GTA Sim 10K, Foggy Cityscapes)

prove it is a strategy to consider [Abramov et al., 2020].

Randomization is also a well-known strategy to address the domain gap between simulation and reality. Tremblay et al. [2018] suggest arbitrarily altering several data generation parameters, including lighting, object posing, and texture, to increase informational entropy and lessen training bias. This study from 2018 first showed it was possible to train neural networks with convincing performance using only synthetic data, circumventing the need to gather manually annotated real-world data or create highly accurate synthetic environments. Nevertheless, in this context, bringing parameter randomization strategies into practice falls beyond the limits of its scope. Parameter variations would affect the urban environment in "City Sample" and require significant extra efforts, including 3D modeling and asset creation.

Another option to enhance performance between real and virtual domains is to leverage the latest capabilities of generative models, particularly to apply image-to-image translation techniques to extend the diversity and variance in the synthetic data distribution. Parmar, Park, Narasimhan, and Zhu [2024] have demonstrated the efficacy of CycleGAN (Cycle-Consistent Generative Adversarial Network) for unsupervised scene translation tasks, such as day-to-night conversion or adding weather effects like fog, snow, and rain. As a generative model, CycleGAN produces new instances by translating existing samples with entirely different image styles—including lighting, texture, and color features. The advantage of CycleGAN for this context is to operate automatic image-to-image translation in an unsupervised manner. Samples from a virtual environment will not have a direct correspondence in the real-world domain, and thus we cannot rely on supervised methods for improving their photorealism or alternating different ambiance properties. As with the prototypical CycleGAN architectures, Parmar et al. [2024] use two translation functions that convert images from the domain X to Y. Both generators use the same underlying network, Stable Diffusion Turbo, with text prompts to guide the style transfer. Cycle consistency loss implies that the images are translated from the source domain to the target domain and reconstructed again to ensure they closely resemble the original and penalize deviations from an idempotent mapping. Adversarial loss derives from coupling each generator with a discriminator that distinguishes between authentic and artificial images. Overall, minimizing both losses optimizes the model's domain adaptation capability.

Samples obtained from inference on the available pre-trained model from Parmar and Zhu [2024] demonstrate visually appealing results, particularly for the use case of rainy weather (see Figure 4.6). However, the pre-trained model outputs a resolution of 512x512, which is significantly lower than the quality level of the images from UE5 (1920x1080). To maintain a similar resolution and for the immediate purpose of domain adaptation between UE5 and CULane, it would be advisable to retrain the CycleGAN model with real driving data. Since this requires considerable video memory resources and an extended timeline, further expansion of the synthetic dataset with CycleGAN and re-testing of CLRNet is set only as a future measure.

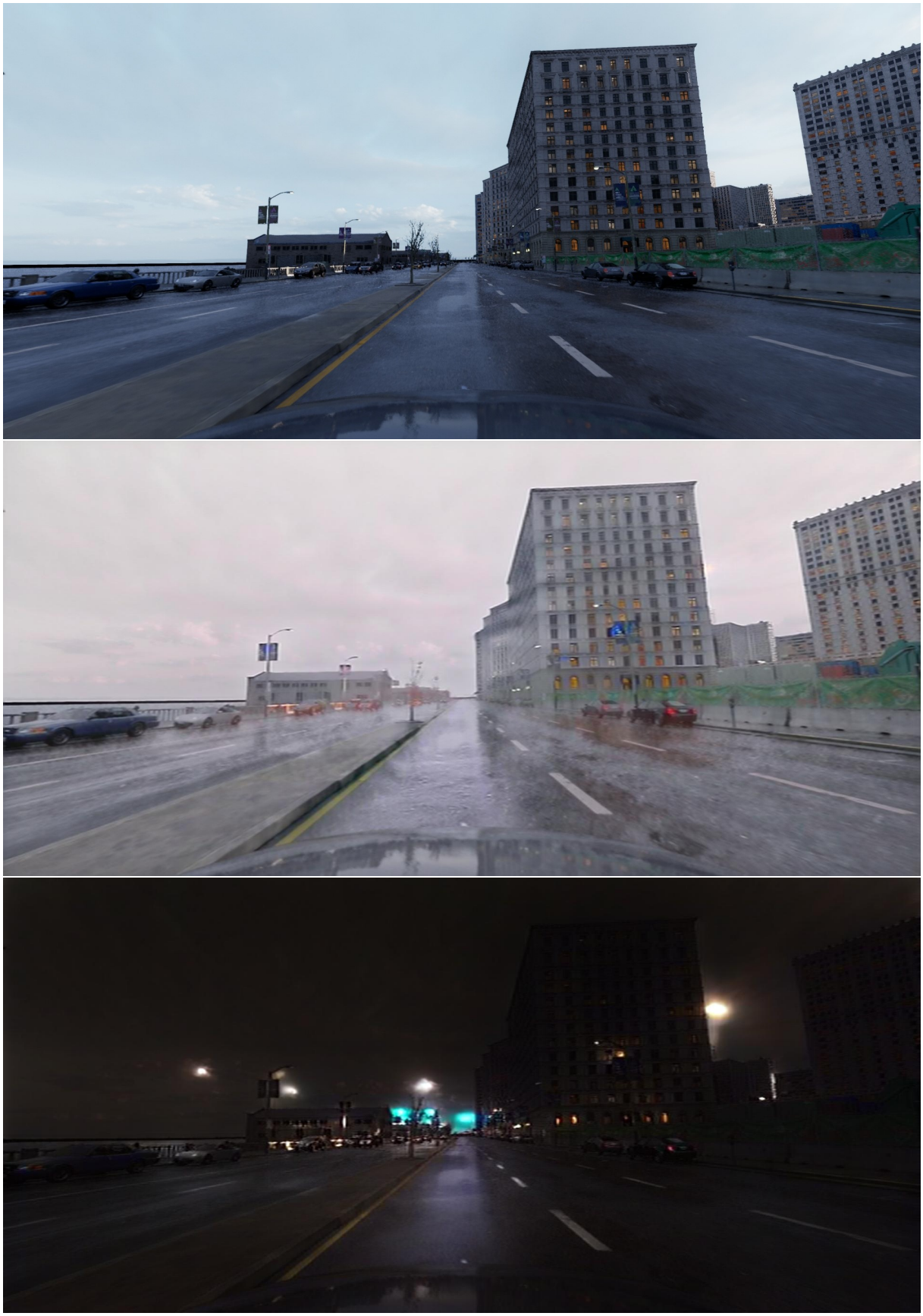


Figure 4.6 Synthetic frame and translation samples (rain, night) with Parmar and Zhu [2024].

Chapter 5

Conclusions & Future Work

Current lane detection systems still exhibit significant performance degradation in certain edge cases. While public benchmarks typically cover a wide range of traffic scenarios and weather conditions, they may still contain blind spots that impact reliability in real-world applications. The validation strategy proposed in this thesis involves extending standard benchmark datasets to achieve broader coverage of environmental variations through the complementary use of synthetic data. For validation, synthetic data can help identify latent anomalies or imbalances. For training, it can augment real-world data with additional traffic scenarios, road types, and varying weather and lighting conditions to improve coverage and generalization.

The primary outcome of this work is an automated framework for generating synthetic data—designed for endurance testing of lane detection systems. Its datasets can seamlessly integrate training, testing, and validation pipelines by complying with a standardized benchmark structure and its official scoring methods. In particular, the synthetic dataset developed in this work is aligned with the CULane benchmark, ensuring compatibility with lane detection models trained or tested on it. The framework also provides an efficient method for extracting accurate ground truth labels, including lane markings, from the simulated environment.

In addition, this work examined the domain gap between simulation and real-world testing by using CLRNet as a reference for the experimental synthetic benchmark. While performance on the synthetic dataset under clear weather closely matched its results on CULane, we observed significant variance across other test cases, with an aggregated average gap of 14.19%. In part, the noticeable weaknesses found in fog and glaring sunlight scenes led us to investigate CLRNet's training dataset and conclude that CULane does contain few fog-filled scenes, which likely contributed to this performance drop.

Nevertheless, these findings also highlight the need for further research into the discrepancies between simulated and real-world environments. Future work shall focus on developing techniques to bridge this domain gap and improve the generalization of models eventually trained on synthetic data. As shown with image-to-image translation techniques in this work, generative models could be employed to mitigate the representational constraints and visual biases that still limit the effectiveness of simulation environments.

Annex A

Data Processing

```python
import os
import re
import cv2
import time
import pathlib
import warnings
import numpy as np
import matplotlib.pyplot as plt
from copy import deepcopy

from scripts.data.processing import Processing
from scripts.validation.spline_regression import SplineRegression
from scripts.validation.lane_markings.lane_markings import LaneMarkings

warnings.filterwarnings('ignore', category=RuntimeWarning)

ENVIRONMENT_TYPE = {
    0: '0_normal',
    1: '1_crowd',
    2: '2_hlight',
    3: '3_shadow',
    4: '4_noline',
    5: '5_arrow',
    6: '6_curve',
    7: '7_cross',
    8: '8_night'
}


class CULaneDataset:
    CULANE_WIDTH = 1920
    CULANE_HEIGHT = 690
    # CULANE_HEIGHT = 1080
    RESIZE_WIDTH = 0
    RESIZE_HEIGHT = 0
    SEG_WIDTH = 30

    def __init__(self, video_path, ground_truth_path, root_dataset_folder):
        self.capture = cv2.VideoCapture(video_path)
        self.valid_frames = self.check_video(self.capture)
        self.safe_exit()
        self.video_name = pathlib.Path(video_path).stem.replace('-', '').replace('UEVIDEOFOOTAGE', '')
        self.dataset_name = f'CULane_UE5Dataset_{self.video_name}/'
        self.dataset_folder = f'{root_dataset_folder}{self.dataset_name}'
        self.target_folder = f'{self.dataset_folder}frames/{self.video_name}/'
        self.segmentation_folder = f'{self.dataset_folder}seg_labels/{self.video_name}/'
        self.metadata_folder = f'{self.dataset_folder}list/'
        self.metadata_split_folder = f'{self.dataset_folder}list/test_split/'
        self.metadata_split_test = f'{self.metadata_split_folder}test'
        self.metadata_test = f'{self.metadata_folder}test.txt'
        self.metadata_validation = f'{self.metadata_folder}val.txt'
        self.metadata_validation_gt = f'{self.metadata_folder}val_gt.txt'
        self.ground_truth_markers = Processing().get_ground_truth_markers(ground_truth_path)
        self.capture = cv2.VideoCapture(video_path)
        self.highest_y_value = -np.inf
        self.lowest_y_value = np.inf
        try:
            os.makedirs(self.dataset_folder)
        except FileExistsError:
            print('[DEBUG] Dataset::__init__:: Target folder already exists.')
            raise FileExistsError
        try:
```

Figure A.1 Code from scripts.data.dataset.py (I).

```
            os.makedirs(self.segmentation_folder)
        except FileExistsError:
            print('[DEBUG] Dataset::__init__:: Target folder already exists.')
        try:
            os.makedirs(self.metadata_folder)
        except FileExistsError:
            print('[DEBUG] Dataset::__init__:: Target folder already exists.')
        try:
            os.makedirs(self.metadata_split_folder)
        except FileExistsError:
            print('[DEBUG] Dataset::__init__:: Target folder already exists.')

    def create_dataset(self, debug=False):
        valid_points = {}
        valid_codes = {}
        valid_lines = {}
        for i, ground_truth in self.ground_truth_markers.items():
            if i in self.valid_frames.keys():
                if len(ground_truth) > LaneMarkings.LANE_MIN_POINTS:
                    valid_points[i] = []
                    valid_codes[i] = []
                    code = ''
                    # Order according to x coordinate to facilitate lane separation
                    ground_truth = ground_truth[ground_truth[:, 0].argsort()]
                    ground_truth[:, 1] = LaneMarkings.HEIGHT - ground_truth[:, 1]
                    lanes = LaneMarkings.separate_lanes_dynamic(ground_truth)
                    if len(lanes) > 2:
                        print(f'[DEBUG] Dataset::create_dataset:: Frame #{i} contains more than 2 lanes.')
                    for lane in lanes:
                        # Re-order according to y coordinate to facilitate spline regression
                        lane = lane[lane[:, 1].argsort()]
                        ground_truth_spline = self.do_splining(lane)
                        valid_points[i].append(ground_truth_spline.points)
                        CULaneDataset.write_markers(self.target_folder, i, ground_truth_spline.points)
                        valid_lines[i] = True
                        self.plot_debug(ground_truth_spline) if debug else None
                    if len(lanes) > LaneMarkings.SINGLE_LANE:
                        code += '1 1 1 0'
                    else:
                        code += '1 0 1 0'
                    valid_codes[i].append(code)
        valid_points = CULaneDataset.filter_dict_by_common_keys(valid_lines, valid_points)
        valid_codes = CULaneDataset.filter_dict_by_common_keys(valid_lines, valid_codes)
        CULaneDataset.draw_markers(self.segmentation_folder, valid_points)
        CULaneDataset.extract_frames(self.target_folder, self.capture, valid_points)
        CULaneDataset.write_metadata(self.dataset_folder, self.target_folder, self.metadata_test)
        for name in ENVIRONMENT_TYPE.values():
            CULaneDataset.write_metadata(
                self.dataset_folder, self.target_folder, f'{self.metadata_split_test}{name}.txt')
        CULaneDataset.write_metadata(self.dataset_folder, self.target_folder, self.metadata_validation)
        CULaneDataset.write_metadata_gt(
            self.dataset_folder, self.target_folder, self.segmentation_folder,
            self.metadata_validation_gt, valid_codes)
        for k, v in valid_points.items():
            for a in v:
                self.highest_y_value = max(self.highest_y_value, a[:, 1].max())
                self.lowest_y_value = min(self.lowest_y_value, a[:, 1].min())
        self.safe_exit()

    def safe_exit(self):
        self.capture.release()
        cv2.destroyAllWindows()
```

Figure A.2 Code from scripts.data.dataset.py (II).

```python
    @staticmethod
    def filter_dict_by_common_keys(dict1, dict2):
        common_keys_set = set(dict1.keys()) & set(dict2.keys())
        filtered_dict2 = {key: dict2[key] for key in common_keys_set}
        return filtered_dict2

    @staticmethod
    def check_video(capture):
        valid_frames = {}
        assert capture.isOpened(), f'[ERROR] Dataset::check_video:: Video stream is closed.'
        index_frame = 0
        print(f'[INFO] Dataset::check_video:: Checking frame status on video capture.')
        while capture.isOpened():
            status, frame = capture.read()
            index_frame += 1
            if status:
                valid_frames[index_frame] = True
            else:
                break
        print(f'[INFO] Dataset::check_video:: Video contains {index_frame} valid frames.')
        return valid_frames

    @staticmethod
    def plot_debug(gt_spline):
        plt.figure()
        plt.scatter(gt_spline.points[:, 0], gt_spline.points[:, 1], color='blue', label='Lane')
        plt.show()
        plt.close()

    @staticmethod
    def do_splining(lane, original_gt=False):
        gt_spline = None
        try:
            points = deepcopy(lane)
            gt_spline = SplineRegression(points)
            if original_gt:
                # Original ground truth
                gt_spline.points = points
            else:
                gt_spline.calculate_GT()
            # Reverse y-axis
            gt_spline.points[:, 1] = CULaneDataset.CULANE_HEIGHT - gt_spline.points[:, 1]
            # Filter out points that are below the cut frame offset
            target_offset = ((LaneMarkings.HEIGHT - CULaneDataset.CULANE_HEIGHT) // 2)
            gt_spline.points[:, 1] += target_offset
            gt_spline.points = gt_spline.points[gt_spline.points[:, 1] <= CULaneDataset.CULANE_HEIGHT]
        except Exception as e:
            print(f'[ERROR] Dataset::do_splining:: {e}.')
        return gt_spline

    @staticmethod
    def write_metadata(root_folder, target_folder, metadata_file):
        with open(f'{metadata_file}', 'w') as file:
            for filename in os.listdir(target_folder):
                if pathlib.Path(filename).suffix == '.jpg':
                    path = target_folder.replace(root_folder, '/')
                    content = f'{path}{filename}\n'
                    file.write(content)

    @staticmethod
    def write_metadata_gt(root_folder, target_folder, segmentation_folder, metadata_file, codes):
        with open(f'{metadata_file}', 'w') as file:
```

Figure A.3 Code from scripts.data.dataset.py (III).

```
        target_files = [_ for _ in os.listdir(target_folder) if pathlib.Path(_).suffix == '.jpg']
        segmentation_files = [_ for _ in os.listdir(segmentation_folder) if pathlib.Path(_).suffix == '.png']
        target_index = [int(re.findall(r'\d+', s)[0]) for s in target_files]
        segmentation_index = [int(re.findall(r'\d+', s)[0]) for s in segmentation_files]
        assert target_index == segmentation_index, '[ERROR] Found mismatches between frames and labels.'
        frame_path = target_folder.replace(root_folder, '/')
        label_path = segmentation_folder.replace(root_folder, '/')
        pairs = zip(target_files, segmentation_files, sorted(codes.keys()))
        content = '\n'.join(map(lambda x: f'{frame_path}{x[0]} {label_path}{x[1]} {" ".join(codes[x[2]])}', pairs))
        file.write(content)
        # print(f'[INFO] Dataset::write_metadata_GT:: Written metadata for ground truth validation.')

    @staticmethod
    def write_markers(target_folder, frame_number, frame_points):
        # print(f'[DEBUG] Dataset::write_markers:: Writing label for frame #{frame_number}.')
        with open(f'{target_folder}frame_{frame_number:05d}.lines.txt', 'a') as file:
            frame_points = frame_points.flatten().tolist()
            s = ' '.join([str(x) for x in frame_points])
            file.write(f'{s}\n')

    @staticmethod
    def draw_markers(segmentation_folder, valid_points, debug=False):
        for index, segments in valid_points.items():
            image = np.zeros((CULaneDataset.CULANE_HEIGHT, CULaneDataset.CULANE_WIDTH, 3), dtype=np.uint8)
            for n, segment in enumerate(segments):
                for i in range(len(segment)-1):
                    # Draw point from each segment
                    start_point = segment[0]
                    end_point = segment[len(segment)-1]
                    cv2.line(image, start_point, end_point, (n + 2, n + 2, n + 2), thickness=CULaneDataset.SEG_WIDTH//2)
            if debug:
                cv2.imshow('', image)
                cv2.waitKey(0)
                cv2.destroyAllWindows()
            # Save the image to disk
            cv2.imwrite(f'{segmentation_folder}seg_frame_{index:05d}.png', image)

    @staticmethod
    def cut_frame(frame, target_height):
        # Crop the frame to the desired height
        start = (frame.shape[0] - target_height)//2
        end = frame.shape[0]-start
        result = deepcopy(frame[start:end, :])
        return result

    @staticmethod
    def extract_frames(target_folder, capture, valid_points, debug=True, continuous=True, show=False, record=True):
        assert capture.isOpened(), f'[ERROR] Dataset::extract_frames:: Video stream is closed.'
        index_frame = 0
        missed_frames = 0
        while capture.isOpened():
            status, frame = capture.read()
            index_frame += 1
            if status:
                if any(index_frame == i for i in valid_points.keys()):
                    frame = CULaneDataset.cut_frame(frame, CULaneDataset.CULANE_HEIGHT)
                    # Save the frame to a file
                    cv2.imwrite(f"{target_folder}frame_{index_frame:05d}.jpg", frame)
                    if debug:
                        for segments in valid_points[index_frame]:
                            for point in segments:
                                frame = cv2.circle(frame, tuple(point), 3, (124, 252, 0), -1)  # LAWN GREEN
```

Figure A.4 Code from scripts.data.dataset.py (IV).

```
                    scale = 100
                    width = int(frame.shape[1] * scale / 100)
                    height = int(frame.shape[0] * scale / 100)
                    frame = cv2.resize(frame, (width, height), interpolation=cv2.INTER_AREA)
                    if show:
                        cv2.imshow('Frame', frame)
                    if record:
                        cv2.imwrite(f'./samples/{target_folder.split("/")[-2]}_frame_{index_frame:05d}.jpg', frame)
                    if continuous:
                        # Continuous mode
                        if cv2.waitKey(1) & 0xFF == ord('e'):
                            break
                    else:
                        # Hold before next frame
                        while not cv2.waitKey(25) & 0xFF == ord('q'):
                            time.sleep(0.1)
                else:
                    # Silent mode
                    if cv2.waitKey(25) & 0xFF == ord('q'):
                        break
            else:
                missed_frames += 1
                break
        print(f'[INFO] Dataset::extract_frames:: Extracted total of {index_frame - missed_frames} frames.')


if __name__ == '__main__':

    root_dataset_dir = '../../dataset/'
    input_dir = 'F:/'

    v_path = f'{input_dir}'
    gt_path = f'{input_dir}'

    v_files = [
        r'D:\Unreal Datasets\TestCase_DawnClear\2024-02-11-21-45-01.avi',
    ]
    gt_files = [
        r'D:\Unreal Datasets\TestCase_DawnClear\2024-02-11-21-45-01.txt',
    ]

    total_frames = 0
    for v, gt in zip(v_files, gt_files):
        print(f'[INFO] ------------------------------------------------------------------------')
        print(f'[INFO] Dataset:: Processing video {v.split("/")[-1]}.')
        try:
            d = CULaneDataset(f'{v}', f'{gt}', root_dataset_dir)
            d.create_dataset()
            total_frames += len(d.valid_frames.keys())
        except FileExistsError:
            print(f'[INFO] Dataset:: Video is already processed.')
        print(f'[INFO] ------------------------------------------------------------------------')
    print(f'[INFO] Dataset:: Total {total_frames} frames collected.')
```

Figure A.5 Code from scripts.data.dataset.py (V).

```python
import re
import numpy as np


class Processing:

    @staticmethod
    def get_output_markers(path):
        output_markers = {}
        points = None
        with open(path, 'r') as f:
            lines = f.readlines()
            for line in lines:
                if 'Frame' in line:
                    frame_number = int(re.findall(r"(\d+)", line)[0])
                    if points is not None:
                        output_markers[frame_number] = points
                    points = np.empty((0, 2), np.uint16)
                elif 'EgoLane' in line:
                    coordinates = re.findall(r"[-+]?(?:\d*\.*\d+)", line)
                    width, height = tuple(map(int, (map(float, coordinates))))
                    points = np.append(points, np.array([[width, height]]), axis=0)
        return output_markers

    @staticmethod
    def get_ground_truth_markers(path):
        ground_truth_markers = {}
        points = None
        frame_counter = 0
        with open(path, 'r') as f:
            lines = f.readlines()
            for line in lines:
                if 'Timestamp' in line:
                    if points is not None:
                        ground_truth_markers[frame_counter] = points
                    points = np.empty((0, 2), np.uint16)
                    frame_counter += 1
                elif 'Moving Status' in line:
                    pass
                elif 'Ego Lane Boundary' in line:
                    coordinates = re.findall(r"[-+]?(?:\d*\.*\d+)", line)
                    width, height = tuple(map(int, (map(float, coordinates))))
                    points = np.append(points, np.array([[width, height]]), axis=0)
        return ground_truth_markers
```

Figure A.6 Code from scripts.data.processing.py (I).

```python
from math import ceil, floor
from scipy.stats import linregress
import numpy as np, pandas as pd
import statsmodels.formula.api as smf


class SplineRegression:

    def __init__(self, points):
        assert len(points) > 0, 'SplineRegression:: List of points is empty.'
        points = points[points[:, 1].argsort()]
        self.points = SplineRegression.extend(points)
        x, y = zip(*self.points[self.points[:, 0].argsort()])
        self.line = pd.DataFrame(columns=['x', 'y'])
        self.line.x = tuple(x)
        self.line.y = tuple(y)

    @staticmethod
    def extend(points):
        x = points[:, 0]
        y = points[:, 1]
        # Fit a linear regression line
        slope, intercept, _, _, _ = linregress(x, y)
        # Find the x-value where y is equal to 0 (also replace zero values with a small epsilon)
        slope = np.where(slope == 0, 1e-10, slope)
        root_x = -intercept / slope
        try:
            root_x = int(root_x)
        except ValueError as e:
            print(f'[ERROR] {e}. Skipping point.')
            return points
        # Extend the points to include the root
        x = np.insert(x, 0, int(root_x))
        y = np.insert(y, 0, 0)
        z = []
        for i in range(len(x)):
            z.append((x[i], y[i]))
        z = np.array(z)
        return z

    def calculate(self):
        formula = 'y ~ bs(x, df=7, degree=7)'
        model_spline = smf.ols(formula=formula, data=self.line)
        result_spline = model_spline.fit()
        y_pred = result_spline.predict(self.line.x)
        # Important to allow for correct comparison in np.diff, since -0.0 do happen
        y_pred = y_pred.astype(int)
        # Determine sorting order (ascending or descending)
        is_ascending = np.all(np.diff(y_pred) >= 0)
        min_y_pred = ceil(min(y_pred))
        max_y_pred = floor(max(y_pred))
        result = []
        for y in np.arange(min_y_pred, max_y_pred + 1, 10):
            if is_ascending:
                x_interp = np.interp(y, self.line.y, self.line.x)
            else:
                x_interp = np.interp(y, self.line.y[::-1], self.line.x[::-1])
            result.append((int(x_interp), int(y)))
        self.points = np.asarray(result)
        self.points = self.points[self.points[:, 0].argsort()]
        self.line = np.array([self.line['x'], self.line['y']]).T
```

Figure A.7 Code from scripts.validation.spline_regression.py (I).

Annex B

CLRNet

```python
import os
import numpy as np
import pickle as pkl
import os.path as osp
from .registry import DATASETS
from .base_dataset import BaseDataset
import clrnet.utils.culane_metric as culane_metric

LIST_FILE = {
    'train': 'list/train_gt.txt',
    'val': 'list/val.txt',
    'test': 'list/test.txt',
}

CATEGORYS = {
    'normal': 'list/test_split/test0_normal.txt',
}


@DATASETS.register_module
class CULane(BaseDataset):
    def __init__(self, data_root, split, processes=None, cfg=None):
        super().__init__(data_root, split, processes=processes, cfg=cfg)
        self.list_path = osp.join(data_root, LIST_FILE[split])
        self.load_annotations()
        self.data_infos = None
        self.max_lanes = None
        self.split = split

    def load_annotations(self):
        self.logger.info('CULane:: Loading CULane annotations...')
        os.makedirs('cache', exist_ok=True)
        cache_path = 'cache/culane_{}.pkl'.format(self.split)
        if os.path.exists(cache_path):
            with open(cache_path, 'rb') as cache_file:
                self.data_infos = pkl.load(cache_file)
                self.max_lanes = max(
                    len(anno['lanes']) for anno in self.data_infos)
                return
        self.data_infos = []
        with open(self.list_path) as list_file:
            for line in list_file:
                infos = self.load_annotation(line.split())
                self.data_infos.append(infos)
        with open(cache_path, 'wb') as cache_file:
            pkl.dump(self.data_infos, cache_file)

    def load_annotation(self, line):
        infos = {}
        img_line = line[0]
        img_line = img_line[1 if img_line[0] == '/' else 0::]
        img_path = os.path.join(self.data_root, img_line)
        infos['img_name'] = img_line
        infos['img_path'] = img_path
        if len(line) > 1:
            mask_line = line[1]
            mask_line = mask_line[1 if mask_line[0] == '/' else 0::]
            mask_path = os.path.join(self.data_root, mask_line)
            infos['mask_path'] = mask_path

        if len(line) > 2:
            exist_list = [int(_) for _ in line[2:]]
            infos['lane_exist'] = np.array(exist_list)
```

Figure B.1 Code from clrnet.datasets.culane.py (I) [Zheng & Huang, 2022].

```
        anno_path = img_path[:-3] + 'lines.txt'  # remove sufix jpg and add lines.txt
        with open(anno_path, 'r') as anno_file:
            data = [
                list(map(float, line.split())) for line in anno_file.readlines()
            ]
        lanes = [[(lane[i], lane[i + 1]) for i in range(0, len(lane), 2)
                  if lane[i] >= 0 and lane[i + 1] >= 0] for lane in data]
        lanes = [list(set(lane)) for lane in lanes]  # remove duplicated points
        lanes = [lane for lane in lanes if len(lane) > 2]  # remove lanes with less than 2 points
        lanes = [sorted(lane, key=lambda x: x[1])
                 for lane in lanes]  # sort by y
        infos['lanes'] = lanes

        return infos

    def get_prediction_string(self, pred):
        ys = np.arange(270, 590, 8) / self.cfg.ori_img_h
        out = []
        for lane in pred:
            xs = lane(ys)
            valid_mask = (xs >= 0) & (xs < 1)
            xs = xs * self.cfg.ori_img_w
            lane_xs = xs[valid_mask]
            lane_ys = ys[valid_mask] * self.cfg.ori_img_h
            lane_xs, lane_ys = lane_xs[::-1], lane_ys[::-1]
            lane_str = ' '.join([
                '{:.5f} {:.5f}'.format(x, y) for x, y in zip(lane_xs, lane_ys)
            ])
            if lane_str != '':
                out.append(lane_str)

        return '\n'.join(out)

    def evaluate(self, predictions, output_basedir):
        print('Generating prediction output...')
        for idx, pred in enumerate(predictions):
            output_dir = os.path.join(
                output_basedir,
                os.path.dirname(self.data_infos[idx]['img_name']))
            output_filename = os.path.basename(
                self.data_infos[idx]['img_name'])[:-3] + 'lines.txt'
            os.makedirs(output_dir, exist_ok=True)
            output = self.get_prediction_string(pred)

            with open(os.path.join(output_dir, output_filename),
                      'w') as out_file:
                out_file.write(output)

        result = culane_metric.eval_predictions(output_basedir,
                                                self.data_root,
                                                self.list_path,
                                                iou_thresholds=np.linspace(0.5, 0.95, 10),
                                                official=True)

        return result[0.5]['F1']
```

Figure B.2 Code from clrnet.datasets.culane.py (II) [Zheng & Huang, 2022].

```
net = dict(type='Detector', )

backbone = dict(
    type='ResNetWrapper',
    resnet='resnet34',
    pretrained=True,
    replace_stride_with_dilation=[False, False, False],
    out_conv=False,
)

num_points = 36
max_lanes = 2
sample_y = range(690, 0, -10)
heads = dict(type='CLRHead', num_priors=192, refine_layers=3, fc_hidden_dim=64, sample_points=36)
iou_loss_weight = 2.
cls_loss_weight = 2.
xyt_loss_weight = 0.2
seg_loss_weight = 1.0

work_dirs = "work_dirs/clr/r34_culane"
neck = dict(type='FPN', in_channels=[128, 256, 512], out_channels=64, num_outs=3, attention=False)
test_parameters = dict(conf_threshold=0.4, nms_thres=50, nms_topk=max_lanes)
epochs = 15
batch_size = 24

optimizer = dict(type='AdamW', lr=0.6e-3)
total_iter = (88880 // batch_size) * epochs
scheduler = dict(type='CosineAnnealingLR', T_max=total_iter)

eval_ep = 3
save_ep = 10
img_norm = dict(mean=[103.939, 116.779, 123.68], std=[1., 1., 1.])
ori_img_w = 1920
ori_img_h = 690
img_w = 800
img_h = 320
cut_height = 270

train_process = [
    dict(
        type='GenerateLaneLine',
        transforms=[
            dict(name='Resize', parameters=dict(size=dict(height=img_h, width=img_w)), p=1.0),
            dict(name='HorizontalFlip', parameters=dict(p=1.0), p=0.5),
            dict(name='ChannelShuffle', parameters=dict(p=1.0), p=0.1),
            dict(name='MultiplyAndAddToBrightness', parameters=dict(mul=(0.85, 1.15), add=(-10, 10)), p=0.6),
            dict(name='AddToHueAndSaturation', parameters=dict(value=(-10, 10)), p=0.7),
            dict(name='OneOf', transforms=[
                    dict(name='MotionBlur', parameters=dict(k=(3, 5))),
                    dict(name='MedianBlur', parameters=dict(k=(3, 5)))
                ], p=0.2),
            dict(name='Affine',
                 parameters=dict(translate_percent=dict(
                     x=(-0.1, 0.1),
                     y=(-0.1, 0.1)),
                     rotate=(-10, 10),
                     scale=(0.8, 1.2)),
                 p=0.7),
            dict(name='Resize', parameters=dict(size=dict(height=img_h, width=img_w)), p=1.0),
        ],
    ),
    dict(type='ToTensor', keys=['img', 'lane_line', 'seg']),
]
```

Figure B.3 Code from configs.clrnet.clr_resnet18_culane_UE5.py (I) [Zheng & Huang, 2022].

```
val_process = [
    dict(type='GenerateLaneLine',
         transforms=[dict(name='Resize', parameters=dict(size=dict(height=img_h, width=img_w)), p=1.0)],
         training=False),
    dict(type='ToTensor', keys=['img']),
]

dataset_path = r'\msc-thesis\dataset\CULane_UE5Dataset_20240225141120'
dataset_type = 'CULane'
dataset = dict(
    train=dict(type=dataset_type, data_root=dataset_path, split='train', processes=train_process),
    val=dict(type=dataset_type, data_root=dataset_path, split='test', processes=val_process),
    test=dict(type=dataset_type, data_root=dataset_path, split='test', processes=val_process)
)

workers = 0
log_interval = 100
num_classes = 4 + 1
ignore_label = 255
bg_weight = 0.4
lr_update_by_epoch = False
```

Figure B.4 Code from configs.clrnet.clr_resnet18_culane_UE5.py (II) [Zheng & Huang, 2022].

```python
import os
import cv2
import argparse
import numpy as np
from functools import partial
from scipy.interpolate import splprep, splev
from scipy.optimize import linear_sum_assignment
from shapely.geometry import LineString, Polygon


def draw_lane(lane, img=None, img_shape=None, width=30):
    if img is None:
        img = np.zeros(img_shape, dtype=np.uint8)
    lane = lane.astype(np.int32)
    for p1, p2 in zip(lane[:-1], lane[1:]):
        cv2.line(img, tuple(p1), tuple(p2), color=(255, 255, 255), thickness=width)

    return img


def discrete_cross_iou(xs, ys, width=30, img_shape=(590, 1640, 3)):
    xs = [draw_lane(lane, img_shape=img_shape, width=width) > 0 for lane in xs]
    ys = [draw_lane(lane, img_shape=img_shape, width=width) > 0 for lane in ys]
    ious = np.zeros((len(xs), len(ys)))
    for i, x in enumerate(xs):
        for j, y in enumerate(ys):
            ious[i, j] = (x & y).sum() / (x | y).sum()

    return ious


def continuous_cross_iou(xs, ys, width=30, img_shape=(590, 1640, 3)):
    h, w, _ = img_shape
    image = Polygon([(0, 0), (0, h - 1), (w - 1, h - 1), (w - 1, 0)])
    xs = [LineString(lane).buffer(distance=width / 2., cap_style=1, join_style=2).intersection(image) for lane in xs]
    ys = [LineString(lane).buffer(distance=width / 2., cap_style=1, join_style=2).intersection(image) for lane in ys]
    ious = np.zeros((len(xs), len(ys)))
    for i, x in enumerate(xs):
        for j, y in enumerate(ys):
            ious[i, j] = x.intersection(y).area / x.union(y).area

    return ious


def interp(points, n=50):
    x = [x for x, _ in points]
    y = [y for _, y in points]
    tck, u, _, _, _ = splprep([x, y], s=0, t=n, k=min(3, len(points) - 1))
    u = np.linspace(0., 1., num=(len(u) - 1) * n + 1)

    return np.array(splev(u, tck)).T


def culane_metric(pred, anno, width=30, iou_thresholds=0.5, official=True, img_shape=(590, 1640, 3)):
    _metric = {}
    for thr in [iou_thresholds]:
        tp = 0
        fp = 0 if len(anno) != 0 else len(pred)
        fn = 0 if len(pred) != 0 else len(anno)
        _metric[thr] = [tp, fp, fn]

    interp_pred = np.array([interp(pred_lane, n=5) for pred_lane in pred], dtype=object)  # (4, 50, 2)
    interp_anno = np.array([interp(anno_lane, n=5) for anno_lane in anno], dtype=object)  # (4, 50, 2)
```

Figure B.5 Code from clrnet.utils.culane.culane_metric.py (I) [Zheng & Huang, 2022].

```
    if official:
        ious = discrete_cross_iou(interp_pred, interp_anno, width=width, img_shape=img_shape)
    else:
        ious = continuous_cross_iou(interp_pred, interp_anno, width=width, img_shape=img_shape)

    row_ind, col_ind = linear_sum_assignment(1 - ious)
    _metric = {}
    for thr in [iou_thresholds]:
        tp = int((ious[row_ind, col_ind] > thr).sum())
        fp = len(pred) - tp
        fn = len(anno) - tp
        _metric[thr] = [tp, fp, fn]

    return _metric


def load_culane_img_data(path):
    with open(path, 'r') as data_file:
        img_data = data_file.readlines()
    img_data = [line.split() for line in img_data]
    img_data = [list(map(float, lane)) for lane in img_data]
    img_data = [[(lane[i], lane[i + 1]) for i in range(0, len(lane), 2)] for lane in img_data]
    img_data = [lane for lane in img_data if len(lane) >= 2]

    return img_data


def load_culane_data(data_dir, file_list_path):
    with open(file_list_path, 'r') as file_list:
        filepaths = [
            os.path.join(
                data_dir, line[1 if line[0] == '/' else 0:].rstrip().replace(
                    '.jpg', '.lines.txt')) for line in file_list.readlines()]
    data = []
    for path in filepaths:
        img_data = load_culane_img_data(path)
        data.append(img_data)

    return data


def eval_predictions(pred_dir, anno_dir, list_path, iou_thresholds=0.5, width=30, official=True, sequential=False):
    import logging
    logger = logging.getLogger(__name__)
    logger.info('Calculating metric for List: {}'.format(list_path))
    predictions = load_culane_data(pred_dir, list_path)
    annotations = load_culane_data(anno_dir, list_path)
    img_shape = (590, 1640, 3)

    if sequential:
        results = map(partial(
            culane_metric, width=width, official=official, iou_thresholds=[iou_thresholds], img_shape=img_shape),
            predictions)
    else:
        from multiprocessing import Pool, cpu_count
        from itertools import repeat
        with Pool(cpu_count()) as p:
            results = p.starmap(culane_metric, zip(predictions, annotations,
                                                   repeat(width),
                                                   repeat([iou_thresholds]),
                                                   repeat(official),
                                                   repeat(img_shape)))
```

Figure B.6 Code from clrnet.utils.culane.culane_metric.py (II) [Zheng & Huang, 2022].

```
    mean_f1, mean_prec, mean_recall, total_tp, total_fp, total_fn = 0, 0, 0, 0, 0, 0
    ret = {}
    for thr in [iou_thresholds]:
        tp = sum(m[thr][0] for m in results)
        fp = sum(m[thr][1] for m in results)
        fn = sum(m[thr][2] for m in results)
        precision = float(tp) / (tp + fp) if tp != 0 else 0
        recall = float(tp) / (tp + fn) if tp != 0 else 0
        f1 = 2 * precision * recall / (precision + recall) if tp != 0 else 0
        logger.info('iou thr: {:.2f}, tp: {}, fp: {}, fn: {}, precision: {}, recall: {}, f1: {}'.format(
            thr, tp, fp, fn, precision, recall, f1))
        mean_f1 += f1 / len([iou_thresholds])
        mean_prec += precision / len([iou_thresholds])
        mean_recall += recall / len([iou_thresholds])
        total_tp += tp
        total_fp += fp
        total_fn += fn
        ret[thr] = {'TP': tp, 'FP': fp, 'FN': fn, 'Precision': precision, 'Recall': recall, 'F1': f1}
    if len([iou_thresholds]) > 2:
        logger.info('mean result, total_tp: {}, total_fp: {}, total_fn: {}, precision: {}, recall: {}, f1: {}'.format(
            total_tp, total_fp, total_fn, mean_prec, mean_recall, mean_f1))
        ret['mean'] = {
            'TP': total_tp,
            'FP': total_fp,
            'FN': total_fn,
            'Precision': mean_prec,
            'Recall': mean_recall,
            'F1': mean_f1
        }
    return ret


def main():
    args = parse_args()
    for list_path in args.list:
        results = eval_predictions(args.pred_dir, args.anno_dir, list_path, width=args.width, official=args.official,
                                   sequential=args.sequential)

        header = '=' * 20 + ' Results ({})'.format(os.path.basename(list_path)) + '=' * 20
        print(header)
        for metric, value in results.items():
            if isinstance(value, float):
                print('{}: {:.4f}'.format(metric, value))
            else:
                print('{}: {}'.format(metric, value))
        print('=' * len(header))


def parse_args():
    parser = argparse.ArgumentParser(description="Measure CULane's metric")
    parser.add_argument("--pred_dir", help="Path to directory containing the predicted lanes", required=True)
    parser.add_argument("--anno_dir", help="Path to directory containing the annotated lanes", required=True)
    parser.add_argument("--width", type=int, default=30, help="Width of the lane")
    parser.add_argument("--list", nargs='+', help="Path to txt file containing the list of files", required=True)
    parser.add_argument("--sequential", action='store_true', help="Run sequentially instead of in parallel")
    parser.add_argument("--official", action='store_true', help="Use official way to calculate the metric")

    return parser.parse_args()


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

Figure B.7 Code from clrnet.utils.culane.culane_metric.py (III) [Zheng & Huang, 2022].

```python
import os
import argparse
import torch.backends.cudnn as cudnn
from clrnet.utils.config import Config
from clrnet.engine.runner import Runner


def main():
    args = parse_args()
    os.environ["CUDA_VISIBLE_DEVICES"] = ','.join(
        str(gpu) for gpu in args.gpus)

    cfg = Config.fromfile(args.config)
    cfg.gpus = len(args.gpus)

    cfg.load_from = args.load_from
    cfg.resume_from = args.resume_from
    cfg.finetune_from = args.finetune_from
    cfg.view = args.view
    cfg.seed = args.seed

    cfg.work_dirs = args.work_dirs if args.work_dirs else cfg.work_dirs

    cudnn.benchmark = True

    runner = Runner(cfg)

    if args.validate:
        runner.validate()
    elif args.test:
        runner.test()
    else:
        runner.train()


def parse_args():
    parser = argparse.ArgumentParser(description='Train a detector')
    parser.add_argument('config', help='train config file path')
    parser.add_argument('--work_dirs', type=str, default=None, help='work dirs')
    parser.add_argument('--load_from', default=None, help='the checkpoint file to load from')
    parser.add_argument('--resume_from', default=None, help='the checkpoint file to resume from')
    parser.add_argument('--finetune_from', default=None, help='the checkpoint file to resume from')
    parser.add_argument('--view', action='store_true', help='whether to view')
    parser.add_argument('--validate', action='store_true', help='whether to evaluate the checkpoint during training')
    parser.add_argument('--test', action='store_true', help='whether to test the checkpoint on testing set')
    parser.add_argument('--gpus', nargs='+', type=int, default='0')
    parser.add_argument('--seed', type=int, default=0, help='random seed')
    args = parser.parse_args()

    return args


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

Figure B.8 Code from main.py (I) [Zheng & Huang, 2022].

Annex C

Synthetic Discriminator

```python
import os
import cv2
import tflearn
import matplotlib
import numpy as np
from tqdm import tqdm
import tensorflow as tf
from random import shuffle
import matplotlib.pyplot as plt
from tflearn.layers.estimator import regression
from tflearn.layers.conv import conv_2d, max_pool_2d
from tflearn.layers.core import input_data, dropout, fully_connected

matplotlib.use('QtAgg')


class Discriminator:

    def __init__(self, ep, lr, width, height, test_dir, load_test, test_file, train_dir, load_train, train_file):
        self.ep = ep
        self.lr = lr
        self.model = None
        self.width = width
        self.height = height
        self.test_dir = test_dir
        self.load_test = load_test
        self.test_file = test_file
        self.train_dir = train_dir
        self.load_train = load_train
        self.train_file = train_file
        self.model_name = 'RealversusSynthetic-{}-{}.model'.format('cnn-3l-512u', lr)

    @staticmethod
    def label_img(img):
        word_label = img.split(' ')[0]
        # One-hot encoding (assuming 2 classes)
        if 'real' in word_label:
            return [1, 0]
        elif 'synthetic' in word_label:
            return [0, 1]
        else:
            print(f'[DEBUG] Discriminator::label_img:: Invalid label encountered: {word_label}.')
            raise ValueError

    @staticmethod
    def process_data(input_dir, output_file, width, height):
        data = []

        for file in tqdm(os.listdir(input_dir)):
            label = Discriminator.label_img(file)
            path = os.path.join(input_dir, file)
            image = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
            image = cv2.resize(image, (width, height))
            data.append(np.array([np.array(image), np.array(label)], dtype="object"))
        for image, label in data:
            assert image.shape[0] == height and image.shape[1] == width, \
                f'[DEBUG] Discriminator::process_data:: Expected image shape {width}x{height}, got {image.shape}.'

        shuffle(data)
        np.save(output_file, data)
        return data
```

Figure C.1 Code from scripts.data.discriminator.py (I).

```
def create_network(self):
    tf.compat.v1.reset_default_graph()
    convnet = input_data(shape=[None, self.width, self.height, 1], name='input')
    convnet = conv_2d(convnet, 32, 5, activation='relu')
    convnet = max_pool_2d(convnet, 5)
    convnet = conv_2d(convnet, 64, 5, activation='relu')
    convnet = max_pool_2d(convnet, 5)
    convnet = conv_2d(convnet, 128, 5, activation='relu')
    convnet = max_pool_2d(convnet, 5)
    convnet = conv_2d(convnet, 64, 5, activation='relu')
    convnet = max_pool_2d(convnet, 5)
    convnet = conv_2d(convnet, 32, 5, activation='relu')
    convnet = max_pool_2d(convnet, 5)
    convnet = fully_connected(convnet, 1024, activation='relu')
    convnet = dropout(convnet, 0.8)
    convnet = fully_connected(convnet, 2, activation='softmax')
    convnet = regression(convnet, optimizer='adam', learning_rate=LR,
                         loss='categorical_crossentropy', name='targets')

    self.model = tflearn.DNN(convnet, tensorboard_dir='log')

def train(self):
    if self.load_train:
        train_data = np.load(self.train_file, allow_pickle=True)
    else:
        train_data = Discriminator.process_data(self.train_dir, self.train_file, self.width, self.height)

    # Create validation set with 10% total data
    threshold = int(len(train_data)*0.1)
    train = train_data[:-threshold]
    validation = train_data[-threshold:]

    x = np.array([i[0] for i in train]).reshape((-1, self.width, self.height, 1))
    y = np.array([i[1] for i in train])
    test_x = np.array([i[0] for i in validation]).reshape((-1, self.width, self.height, 1))
    test_y = np.array([i[1] for i in validation])

    self.model.fit(
        {'input': x},
        {'targets': y},
        n_epoch=self.ep,
        validation_set=(
            {'input': test_x},
            {'targets': test_y}),
        snapshot_step=5,
        show_metric=True,
        run_id=self.model_name)

    self.model.save(self.model_name)

def test(self):
    if self.load_test:
        test_data = np.load(self.test_file, allow_pickle=True)
    else:
        test_data = Discriminator.process_data(self.test_dir, self.test_file, self.width, self.height)

    self.model.load(self.model_name)
    valid_count = 0
    total_count = 0
    fig, axes = plt.subplots(4, 5, figsize=(20, 7))

    for num, data in enumerate(test_data):
```

Figure C.2 Code from scripts.data.discriminator.py (II).

```
            image, label = data
            image = image.reshape(self.width, self.height, 1)

            model_out = self.model.predict([image])[0]
            classification = "real" if np.argmax(model_out) == 0 else "synthetic"

            if classification == label:
                valid_count += 1

            total_count += 1
            if total_count <= 20:
                axes.flat[num].imshow(image)
                axes.flat[num].set_title(classification)
                axes.flat[num].axis("off")  # Turn off axes for cleaner visualization

            print(f"[INFO] Discriminator::run:: Classifying image #{total_count} as {classification}: "
                  f"{'valid' if classification == label else 'invalid'}.")

        accuracy = (valid_count / total_count) * 100
        print(f"[INFO] Discriminator::run:: Total accuracy: {accuracy:.2f}%.")
        plt.tight_layout()
        plt.show()


if __name__ == '__main__':

    EP = 1
    LR = 1e-3
    WIDTH = 1462
    HEIGHT = 526
    TRAIN_FILE = 'small_train_data.npy'
    TRAIN_DIR = 'small_train'
    LOAD_TRAIN = True
    TEST_FILE = 'small_test_data.npy'
    TEST_DIR = 'small_test'
    LOAD_TEST = True

    d = Discriminator(EP, LR, WIDTH, HEIGHT, TEST_DIR, LOAD_TEST, TEST_FILE, TRAIN_DIR, LOAD_TRAIN, TRAIN_FILE)
    d.create_network()
    d.train()
    d.test()
```

Figure C.3 Code from scripts.validation.discriminator.py (III).

Annex D

ICoMS 2024 Publication

Harness the Unreal: Evaluating State-of-the-Art Lane Detection with Synthetic Data

Pedro Amaro Costa
Department of Electronics, Telecommunications and Computer Engineering, Lisbon School of Engineering (ISEL/IPL)
Portugal
a43254@alunos.isel.pt

Arnaldo Joaquim Abrantes
Department of Electronics, Telecommunications and Computer Engineering, Lisbon School of Engineering (ISEL/IPL)
Portugal
arnaldo.abrantes@isel.pt

Abstract

Autonomous driving systems rely heavily on lane detection. Therefore, ensuring its robustness and reliability is crucial for road safety. This work proposes to validate one of the leading lane detection models in the CULane benchmark with an alternative synthetic dataset—with full automated ground truth labeling—from Epic Games' Unreal Engine 5, a dynamically enriched, photorealistic simulation environment. By providing a range of diverse and challenging conditions (circadian, climatic, and road types), we aim to analyze the algorithm's performance and, in parallel, to collect reference indicators of the domain gap versus its original real-world dataset. Findings reinforce the role of synthetic data to expand testing coverage and minimize the imbalance of training datasets for safety-critical applications.

CCS Concepts

• **Computing methodologies**; • **Modeling and simulation**; • **Simulation evaluation**; • **Machine learning**; • **Cross-validation**; • **Artificial intelligence**; • **Computer vision**; • **Computer vision problems**; • **Object detection**;

Keywords

Synthetic Data, Domain Gap, Lane Detection, Autonomous Driving

ACM Reference Format:
Pedro Amaro Costa and Arnaldo Joaquim Abrantes. 2024. Harness the Unreal: Evaluating State-of-the-Art Lane Detection with Synthetic Data. In *2024 7th International Conference on Mathematics and Statistics (ICoMS 2024), June 23–25, 2024, Amarante, Portugal*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3686592.3686605

1 INTRODUCTION

Lane detection, part of the perception stack in autonomous driving systems, guarantees that the environment model correctly captures the traffic layout of the road. The aim is to accurately extract lane boundaries and continuously track these markings in real-time. It assists in segmenting the suitable navigation path, enabling autonomous vehicle motion control and trajectory planning. As noted by Zakaria et al. [1], in recent years, modern lane detection methods have mainly employed deep learning models, either in stand-alone end-to-end architectures or in combination with classical computer vision approaches, supported by geometric modeling in conventional pipelines (denoising, feature extraction, model fitting, and lane tracking). Although classical methods offer pixel-preciseness, transparency, and predictability (non-stochastic training), they can be too extensive or sluggish for real-time applications and often rely on complex heuristics or handcrafted configurations that struggle to adapt to eventful scenarios or unexpected road layouts. In contrast, results have overwhelmingly shown the high generalization capabilities of convolutional neural networks and encoder-decoder structures [2]. However, a trade-off remains in this transition.

Safety-critical applications such as lane detection require formal verification of robustness and reliability. First, to ensure it maintains functional behavior with erratic or glaring lighting, adverse weather, faded, omitted, or fragmented lane marking sections, and, secondly, that it does not degrade its performance yet upholds a threshold of confidence throughout its life cycle. Deep learning techniques now add a new challenge: safeguarding the system if the underlying components possess inherently opaque or emergent characteristics. Borg et al. [3] defend that ADAS development standards should necessarily augment testing specifications and procedures in response to the increasing dependency on neural networks. Accordingly, to expand from exclusive on-road testing to simulation and synthetic data, matching the exponential growth of formal verification scenarios and the corresponding coverage needed.

1.1 Related Work

CULane [4], TuSimple [5], and LLAMAS [6] are some of the most frequently referenced lane detection benchmarks in the literature. These datasets derive from extensive real-world driving data captured by camera-equipped test vehicles, generally ranging across various traffic situations, geographical locations, weather, and lighting conditions. Its ground truth is manually annotated and usually consists of pixel-wise markers for lane-fitting polynomial lines; in certain circumstances, it also provides lane-type information (solid, dashed).

Regarding synthetic techniques, many works have used the open-source simulator CARLA [7] for data generation. Tran et al. [8] trained and validated a U-Net segmentation network for extracting lane marking features with a dataset of 4000 images from the simulator. CarlaScenes [9] is a benchmark dataset specifically designed to analyze the performance of odometry models on a collection of specific test cases with camera and LIDAR-labeled data. Simulanes

ICoMS 2024, June 23–25, 2024, Amarante, Portugal

ACM ISBN 979-8-4007-0722-3/24/06
https://doi.org/10.1145/3686592.3686605

[10] is a CARLA-based dataset generator that supports a domain adaptation technique with adversarial generative and feature discriminators. The goal is to train the learning model on a synthetic domain with simulation-based labeled scenes and predict lanes on a given real-data domain.

More recent examples of synthetic data generation already incorporate Unreal Engine 5 (UE5). Damian et al. [11] trained object recognition algorithms with data from the platform. Accurate 3D replicas of real-world objects are scanned using photogrammetry, imported into various simulation scenarios, and compiled as a training set for YOLOv8 to increase robustness. Hu et al. [12] developed a synthetic dataset using UE5 to evaluate the impact of camera shutter types on object detection's accuracy, specifically for low-speed traffic participants (pedestrians).

Garnett et al. [13] proposed to train lane detection models with a reduced part of the TuSimple and LLAMAS datasets (10%); the remainder consisted of synthetically generated data from 3D modeling (Blender), with parameterized variability in road and lane topology, topography, and curvature. Nie et al. [14] generated artificially foggy scenes with an atmospheric scattering model and applied them as a data augmentation method to the CULane dataset, improving lane detection of the state-of-the-art models on both fog and open sky images.

2 METHODOLOGY

2.1 Synthetic Versus Real Data

The data constraints associated with developing lane detection models are well known. As observed by Pan et al. [4], one of the reasons for developing CULane was that datasets available at the time were either "too small or too simple" to train their Spatial-CNN network. Composed of only a few thousand images taken in daylight with limited traffic (highway) and clear line markings, they rarely contained occlusions from other elements, fragmented sections, or signs of erosion. Instead, CULane stems from a dense Beijing urban scenario and includes unfavorable lighting or weather conditions. Thus, it provides a more challenging benchmark dataset for validating lane detection systems, with a volume 20 times larger than TuSimple. According to Li [15], CULane is still the most requested benchmark by researchers, accounting for 63% of references in the literature.

Although several other large and robust datasets are now open source, all still require significant investment. Since typical datasets stem from in-car recordings of actual traffic on public roads and imply labeling frame-by-frame, pixel-accurate lane marking points, they are both onerous and logistically complex. Furthermore, if the data specifications for a particular system change or expand, thus demanding added volume, its development costs may increase immeasurably. On the contrary, synthetic datasets are much more cost-effective than real-world data because volume scales with a residual impact on effort and equipment. Also, since the annotation process in simulation is automated (rather than manual), it can be assessed for quality and accuracy criteria. Overall, it offers control and flexibility, permitting changes in annotation requirements during the development cycle or according to different target classes.

The degree of variability in the recorded data constricts any learning model's capacity. Zakaria et al. [1] note that lane markings in real-world collections typically follow country or region-specific conventions and that vehicle speed tends to be low-flow traffic (up to 80 km/h), lacking higher-speed instances. In contrast, it is possible to tailor synthetic data and provide these specific environmental factors, plus adverse illumination, rare traffic events, and unusual road layouts. Fundamentally, it can support a dataset curation strategy to control imbalances in training. For instance, if a dataset contains a 20/80 ratio between day and night scenarios, sheer volume will not be sufficient; the poor economy of the data will still affect the model's performance and thwart its validation. A robust verification and validation (V&V) strategy should enforce the most comprehensive and efficient collection of testing variations within the development dataset. In this sense, simulation environments are an effective option for completing coverage of the driving domain and adding the needed degree of variance expected for training and testing.

Of course, the simulation-to-real domain gap in camera data is still considerable, even considering state-of-the-art rendering engines. The photorealism and diversity of artificial environments may be impressive. However, variance in a real-world driving dataset should not be underestimated (from signal noise, unpredictability of traffic occurrences, or anomalies). Even with advanced generation techniques, synthetic environments will not encompass the full range of complexity found on the road. Synthetic data may fruitfully expand authentic data, yet attempting to replace it entirely may implicitly create invisible ceilings and arbitrary overfits for the generalization capabilities of a learning model.

2.2 Simulation Environment

This work adopts UE5 as its simulation environment. The latest version of this engine has brought an unprecedented level of photorealism to graphics rendering. In addition, Epic Games has also released a free, downloadable open world as a demonstration: «City Sample». It recreates a 16-km2 modern metropolis with a dense flow of traffic and pedestrians supported by Epic's MassAI system.

Establishing parametric control over the current traffic participants is a prerequisite for setting up a synthetic data pipeline. Specifically, this involves integrating the newly created ego vehicle agent into the MassAI traffic system and spawning it at a predetermined point in the city. The ego vehicle must drive autonomously, blend in with traffic, and exhibit naturalistic behavior (obey traffic rules, keep a safe distance from other agents, and follow coherent routes). To replicate existing datasets, the camera sensor was integrated into the ego vehicle with specifications closely resembling those found in the CULane dataset. The camera was mounted in a position identical to the CULane setup, ensuring alignment of the horizon line (the upper vertex of the lane splines typically occupies approximately 45-50% of the image height).

As for ground truth, spline components are applied to the road base meshes. The texture of each lane marking section in the city's road network is juxtaposed by a spline adjusted to match its exact position, orientation, and size. At runtime, a detection function filters in the splines in front of the vehicle (up to a longitudinal range of 50 meters, defined as an appropriate limit for the perception of

Figure 1: Mosaic with test case samples. From left to right: clear, fog, and glare. From top to bottom: dawn, noon, and night.

the horizon) and matches its driving orientation within a 60°angle radius (to allow the correct inclusion of lanes tailored to road curvatures and exclude perpendicular lanes at intersections). The points from the spline are collected with a 50-cm sampling interval in between. The 3D world coordinates of these points are projected onto the camera's 2D coordinate reference system. The result is a set of 2D positions that determine the visible lane markings on the ego vehicle's camera viewport.

The CULane dataset is the template model for converting the synthetic UE5 dataset into a standard benchmark format. It provides 133235 images at 25 frames per second [4]. Each frame contains manual labeling of all traffic lanes with cubic splines. When occluded by vehicles or not visible, the lane markings are estimated depending on the context. To ensure consistency with CULane, the ground truth points are converted to a cubic spline using spline regression and subsampled at a 10-pixel' uniform interval. Since detection models expect instance segmentation for each lane, these are generated from the previously computed cubic splines, with a constant 30 pixels as the lane's segment width, following the configurations of most lane detection datasets [4, 6]. Finally, the result is a synthetic dataset that can feed the training or testing pipeline of any lane detection model already configured for the CULane benchmark.

2.3 System Under Test

Using a novel cross-layer network architecture, CLRNet achieved the highest F1 score (80.47%) at CULane in 2022 [16]. Its results also marked impressive 97.89% and 96.12% as maximum scores on the TuSimple and LLAMAS datasets, respectively. There is no performance report regarding real-world deployment. The primary innovation lies in its initial high-level lane detection process, followed by a progressive refinement of fine-grained features to pinpoint their precise location [16]. This hybrid method focuses on contextual data (ROIGather) to maintain lane detection accuracy in highly dynamic situations. In addition, it introduces Line IoU Loss as a regression method over the entire lane. Overall, it offers greater robustness since it remarkably resembles human vision in adverse visibility conditions: when a lane is either occluded, obscured by darkness, or glare, the perception focus immediately shifts, raising cognitive attention from the lane itself to the global layout and landmarks of the road.

3 EXPERIMENTAL RESULT

We selected the CLRNet ResNet-34 model checkpoint [17] for inference on our synthetic dataset. The dataset totals 44272 frames from multiple video segments covering nine established test cases, recorded at 24 frames per second with 1920x690 resolution. The size of this dataset corresponds to 33% of the full CULane and is significantly larger than its respective testing set, with 34680 frames [4]. Test cases were prepared to reflect variations in circadian phases and challenging weather conditions. Dawn, noon, and night determine three different types of lighting. To illustrate complex driving situations, foggy and glaring sunlight scenarios are set. Figure 1 contains example frames.

3.1 Performance

Overall, the model's performance has slightly degraded compared to its original results on the CULane benchmark (see Table 1). The maximum F1-score (79.50%) occurred with a clear sky at dawn, indicating the importance of high ambient illumination levels for lane recognition. All test cases under a clear sky delivered values (79.50%, 74.56%, and 71.08% at dawn, noon, and night, respectively) that fell within a fair range of the mF1-score on CULane (79.73%) for the CLRNet ResNet-34 model [16]. Regardless of the time of day, the most strenuous test conditions—fog and glare—expectedly produced the lowest results. The impact of nighttime testing was less pronounced because of the urban environment's bright street lighting. As a note, our analysis constrained the model to a two-lane maximum detection threshold.

Figures 2, 3, and 4 illustrate the most relevant findings. Lane marking edges and other road markers are dim under fog conditions, which causes irregular and oscillating lane detection. Furthermore, harsh sunlight reflections on the pavement pose the most severe challenge to the model. Here, contrast loss is nearly complete,

Table 1: Performance of the reference model on synthetic data

Test Case	Daytime	Weather	Frames	Precision	Recall	F1-Score	IoU Threshold
1	Dawn	Clear	4505	77.37	81.74	79.50	0.5
2	Dawn	Fog	5605	79.98	46.91	59.13	0.5
3	Dawn	Glare	5667	62.26	45.33	52.46	0.5
4	Noon	Clear	4360	77.97	71.44	74.56	0.5
5	Noon	Fog	4517	75.87	43.35	55.17	0.5
6	Noon	Glare	5767	76.42	50.20	60.60	0.5
7	Night	Clear	4395	74.18	68.23	71.08	0.5
8	Night	Fog	4391	78.79	57.87	66.73	0.5
9	Night	Glare	5065	71.37	66.89	69.06	0.5
Total	–	–	**44272**	**74.63**	**58.43**	**65.54**	**0.5**

as the road surface reflects as brightly as the color of its lanes. Both scenarios are well-known for their difficulty in real-world conditions, stressing the image sensor's dynamic range to the point of complete blur or saturation. Arthi et al. [18], among others, have focused on the impact of adverse weather conditions (fog and glare as well as rain and snow) and reviewed the multiple strategies adopted for improving object detection. Even though CLRNet enhances feature detection with contextual information, these scenarios still hinder its results considerably, which may stem from its rarity in the original CULane training set.

Table 1 shows that precision values generally exceeded recall, consistent with the necessary balance for lane detection functionality. For road deployment, false positives are more dangerous than false negatives since detecting incorrect lanes may lead to erratic trajectories and diverge the vehicle off-road or into adjacent traffic, while missed instances can still be recoverable. The Intersection-over-Union (IoU) threshold describes the minimum spatial overlap for a true positive. As the established lane width is 30 pixels wide in CULane [4], the detection result is valid if it overlaps the ground truth segment by at least 50%.

3.2 Domain Gap

By examining CLRNet's performance in synthetic data compared to the original dataset—before any cross-domain adaptation measures—we proposed to quantify the raw domain gap or degree of similarity between CULane and our experimental benchmark. Given that the best test case score obtained using the synthetic dataset (in clear weather conditions) closely matched the real-world dataset, the domain gap in simulation may appear to be already negligible. However, there is a significant variance between test cases, and the aggregated results still show a 14.19% gap compared to the CLRNet ResNet-34 model's performance on real-world data (79.73%). This discrepancy is partly due to the different testing scenarios between CULane and our synthetic dataset, in particular the challenging conditions with fog and glare. In parallel, UE5 image rendering attributes and its camera model properties will also affect, to a certain degree, the model's transferability to the synthetic domain. Finally, the original camera specifications—undisclosed in CULane—should also be considered a relevant factor in setting up any camera-based simulation.

As expected, if a model is neither trained nor optimized for its target domain, it will inevitably suffer a drop in performance proportionate to the degree of dissimilarity from its source domain. In this regard, Abramov et al. [19], among others, have demonstrated how classical unsupervised techniques such as Feature Distribution Matching (FDM) and Exact Histogram Matching (EHM) can, to some extent, improve the similarity and, consequently, detection performance between synthetic and real-world datasets.

4 CONCLUSION

The validation strategy proposed in this work is to extend standard datasets and achieve broad coverage of environmental variations through the complementary role of synthetic data. Leading lane detection systems may exhibit significant performance degradation in certain edge cases. Even though CULane covers an impressive range of traffic scenarios and weather conditions, it rarely contains fog-filled scenes, which may later represent a blind spot for any model trained on it. Synthetic data can help predict latent anomalies or imbalances with ease and cost-effectiveness. Thus, real-world data sets may be augmented with traffic scenarios, roadway types, weather, and lighting conditions to increase validation coverage and generalization capability before deployment.

In parallel, this work investigated the gap between simulation and the real-world domain by establishing CLRNet as a reference for the proposed experimental synthetic benchmark. The indicators raise interest in further research into this residual, still-remaining divergence. An immediate option is to reverse roles—by training on synthetic data and validating with authentic data—to determine whether the cross-domain results are bound by some symmetry. Furthermore, employ unsupervised cross-domain fitting methods to control the representational constraints and visual bias that continue to impair simulation environments.

Acknowledgments

This work is supported by NOVA LINCS (UIDB/04516/2020) with the financial contribution of FCT.IP.

References

[1] Noor J. Zakaria, Mohd I. Shapiai, Rasli A. Ghani, Najib Yasin, Mohd Z. Ibrahim, and Nurbaiti Wahid. 2023. Lane Detection in Autonomous Vehicles: A Systematic Review. In IEEE Access 11 (January 2023), 3729-3765. https://doi.org/10.1109/ACCESS.2023.3234442.

Figure 2: Ground truth (left) and lane detection (right). The model often inferred lane markings from parking boundaries.

Figure 3: Ground truth (left) and lane detection (right). The model misinterpreted numerous frames in fog-filled scenes.

Figure 4: Ground truth (left) and lane detection (right). Glare from direct sunlight creates the most severe challenges.

[2] Weiyu Hao. 2023. Review on lane detection and related methods. In Cognitive Robotics 3 (April 2023), 135-141. https://doi.org/10.1016/j.cogr.2023.05.004.

[3] Markus Borg, Cristofer Englund, Krzysztof Wnuk, Boris Duran, Christoffer Levandowski, Shenjian Gao, Yanwen Tan, Henrik Kaijser, Henrik Lönn, and Jonas Törnqvist. 2019. Safely Entering the Deep: A Review of Verification and Validation for Machine Learning and a Challenge Elicitation in the Automotive Industry. In Journal of Automotive Software Engineering 1, 1 (January 2019), 1-19. https://doi.org/10.2991/jase.d.190131.001.

[4] Xingang Pan, Jianping Shi, Ping Luo, Xiaogang Wang, and Xiaoou Tang. 2017. Spatial As Deep: Spatial CNN for Traffic Scene Understanding. In 32nd Conference on Artificial Intelligence (AAAI'18), February 2-7, 2018, New Orleans, LA, USA, 7276-7283. https://doi.org/10.48550/arXiv.1712.06080.

[5] TuSimple. 2017. TuSimple Benchmark. Retrieved February 5, 2024, from https://github.com/TuSimple/tusimple-benchmark.

[6] Karsten Behrendt, and Ryan Soussan. 2019. Unsupervised Labeled Lane Markers Using Maps. In IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), October 27-28, 2019, Seoul, Korea, 832-839. https://doi.org/10.48550/arXiv.1712.06080.

[7] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An Open Urban Driving Simulator. In 1st Annual Conference on Robot Learning (CoRL), November 13-15, 2017, Mountain View, CA, USA, 1-16. https://doi.org/10.48550/arXiv.1711.03938.

[8] Le-Anh Tran, and My-Ha Le. 2019. Robust U-Net-based Road Lane Markings Detection for Autonomous Driving. In International Conference on System Science and Engineering (ICSSE), July 20-21, 2019, Dong Hoi, Vietnam, 62-66. https://doi.org/10.1109/ICSSE.2019.8823532.

[9] Andreas Kloukiniotis, Andreas Papandreou, Christos Anagnostopoulos, Aris Lalos, Petros Kapsalas, Duong-Van Nguyen, and Konstantinos Moustakas. 2022. CarlaScenes: A synthetic dataset for odometry in autonomous driving. In IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), June 19-20, 2022, New Orleans, LA, USA, 4519-4527. https://doi.org/10.1109/CVPRW56347.2022.00498.

[10] Chuqing Hu, Sinclair Hudson, Martin Ethier, Mohammad Al-Sharman, Derek Rayside, and William Melek. 2022. Sim-to-Real Domain Adaptation for Lane Detection and Classification in Autonomous Driving. In IEEE Intelligent Vehicles Symposium (IV), June 4-9, 2022, Aachen, Germany, 457-463. https://doi.org/10.1109/IV51971.2022.9827450.

[11] Alexandru Damian, Claudiu Filip, Anamaria Nistor, Irina Petrariu, Cătălin Mariuc, and Valentin Stratan. 2023. Experimental Results on Synthetic Data Generation in Unreal Engine 5 for Real-World Object Detection. In 17th International Conference on Engineering of Modern Electric Systems (EMES), June 9-10, 2023, Oradea, Romania, 1-4. https://doi.org/10.1109/EMES58375.2023.10171761.

[12] Yue Hu, Gourav Datta, Kira Beerel, and Peter Beerel. 2023. Let's Roll: Synthetic Dataset Analysis for Pedestrian Detection Across Different Shutter Types. arXiv: 2309.08136. Retrieved from https://arxiv.org/abs/2309.08136.

[13] Noa Garnett, Roy Uziel, Netalee Efrat, and Dan Levi. 2020. Synthetic-to-Real Domain Adaptation for Lane Detection. In 15th Asian Conference on Computer Vision (ACCV), December 3-4, 2020, Kyoto, Japan, 52-67. https://doi.org/10.1007/978-3-030-69544-6_4.

[14] Xiangyu Nie, Zhejun Xu, Wei Zhang, Xue Dong, Ning Liu, and Yuanfeng Chen. 2022. Foggy Lane Dataset Synthesized from Monocular Images for Lane Detection Algorithms. In Sensors 22, 14 (July 2022), 5210. https://doi.org/10.3390/s22145210.

[15] Junyan Li. 2023. Lane Detection with Deep Learning: Methods and Datasets. In Information Technology and Control 52, 2 (February 2023), 297-308. https://doi.org/10.5755/j01.itc.52.2.32841.

[16] Tu Zheng, Yifei Huang, Yang Liu, Wenjian Tang, Zheng Yang, Deng Cai, and Xiaofei He. 2022. CLRNet: Cross Layer Refinement Network for Lane Detection. In IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 19-20, 2022, New Orleans, LA, USA, 898-897. https://doi.org/10.1109/CVPR52688.2022.00097.

[17] Tu Zheng, and Yifei Huang. 2022. CLRNet: Cross Layer Refinement Network for Lane Detection. Retrieved February 5, 2024, from https://github.com/Turoad/CLRNet.

[18] V. Arthi, R. Murugeswari, and Nagaraj P. 2022. Object Detection of Autonomous Vehicles under Adverse Weather Conditions. In International Conference on Data Science, Agents & Artificial Intelligence (ICDSAAI), December 8-10, 2022, Chennai, India, 1-8. https://doi.org/10.1109/ICDSAAI55433.2022.10028795.

[19] Alexey Abramov, Christopher Bayer, and Claudio Heller. 2020. Keep it Simple: Image Statistics Matching for Domain Adaptation. In IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 14-19, 2020, Seattle, WA,

USA, 1-9. https://doi.org/10.48550/arXiv.2005.12551.

References

Abramov, A., Bayer, C., & Heller, C. (2020, June). Keep it Simple: Image Statistics Matching for Domain Adaptation. In *Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops* (pp. 1–9). IEEE Press. doi: 10.48550/arXiv.2005.12551

Arthi, V., Murugeswari, R., & Nagaraj, P. (2022, December). Object Detection of Autonomous Vehicles under Adverse Weather Conditions. In *Proceedings of the 2022 International Conference on Data Science, Agents & Artificial Intelligence* (pp. 1–8). IEEE Press. doi: 10.1109/ICDSAAI55433.2022.10028795

AVSimulation. (2024). *SCANeR.* Retrieved from `https://www.avsimulation.com/scaner`

Behrendt, K., & Soussan, R. (2019, October). Unsupervised Labeled Lane Markers Using Maps. In *Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision Workshop* (pp. 832–839). IEEE Press. doi: 10.1109/ICCVW.2019.00111

Borg, M., Englund, C., Wnuk, K., Duran, B., Levandowski, C., Gao, S., ... Törnqvist, J. (2018, December). Safely Entering the Deep: A Review of Verification and Validation for Machine Learning and a Challenge Elicitation in the Automotive Industry. *Journal of Automotive Software Engineering*, *1*(1), 2589–2258. doi: 10.2991/JASE.D.190131.001

Brito, M. (2024). *MSC Screen & Camera Recorder.* Retrieved from `https://www.unrealengine.com/marketplace/product/msc-screen-recorder`

Built In. (2024). *V-Model.* Retrieved from `https://builtin.com/software-engineering-perspectives/v-model`

CARLA. (2024). *CARLA Documentation.* Retrieved from `https://carla.org`

Continental Automotive. (2021). *EU General Safety Regulation.* Retrieved from `https://www.continental-automotive.com/en/industry/trucks-and-buses/eu-general-safety-regulations`

Continental Automotive. (2024). *Multi Function Mono Camera.* Retrieved from `https://www.continental-automotive.com/en/components/cameras/multi-function-mono-camera-mfc525`

Damian, A., Filip, C., Nistor, A., Petrariu, I., Mariuc, C., & Stratan, V. (2023, June). Experimental Results on Synthetic Data Generation in Unreal Engine 5 for Real-World Object Detection. In *Proceedings of the 17th International Conference on Engineering of Modern Electric Systems* (pp. 1–4). IEEE Press. doi: 10.1109/EMES58375.2023.10171761

dSPACE. (2024). *Empowering Safe Autonomous Driving.* Retrieved from https://www.dspace.com/en/pub/home/applicationfields/ind-appl/automotive-industry/autonomous-driving

Epic Games. (2024). *City Sample.* Retrieved from https://www.unrealengine.com/marketplace/product/city-sample

European Commission. (2020). *Next Steps Towards 'Vision Zero' – EU Road Safety Policy Framework 2021-2030*. Luxembourg: Publications Office. doi: 10.2832/391271

European Union. (2019, December). Regulation (EU) 2019/2144 of the European Parliament and of the Council of 27 November 2019 on Type-Approval Requirements for Motor Vehicles and Their Trailers, and Systems, Components and Separate Technical Units Intended for Such Vehicles, as Regards Their General Safety and the Protection of Vehicle Occupants and Vulnerable Road Users. *Official Journal L325*, 1—40. Retrieved from https://data.europa.eu/eli/reg/2019/2144/oj

Garnett, N., Uziel, R., Efrat, N., & Levi, D. (2020). Synthetic-to-Real Domain Adaptation for Lane Detection. In *Proceedings of the 15th Asian Conference on Computer Vision, month = December* (p. 52-67). Springer International Publishing. doi: 10.48550/arXiv.2007.04023

Hakuli, S., & Krug, M. (2014). Virtual Integration in the Development Process of ADAS. In *Handbook of Driver Assistance Systems: Basic Information, Components and Systems for Active Safety and Comfort* (pp. 1–14). Cham, Germany: Springer International Publishing. doi: 10.1007/978-3-319-09840-1_8-1

Hauer, F., Schmidt, T., Holzmüller, B., & Pretschner, A. (2019, October). Did We Test All Scenarios for Automated and Autonomous Driving Systems? In *Proceedings of the 2019 IEEE Intelligent Transportation Systems Conference* (p. 2950-2955). IEEE Press. doi: 10.1109/ITSC.2019.8917326

Hu, C., Hudson, S., Ethier, M., Al-Sharman, M., Rayside, D., & Melek, W. (2022). Sim-to-Real Domain Adaptation for Lane Detection and Classification in Autonomous Driving. In *Proceedings of the 2022 IEEE Intelligent Vehicles Symposium, month = July* (pp. 457–463). IEEE Press. doi: 10.1109/IV51971.2022.9827450

Hu, Y., Datta, G., Beerel, K., & Beerel, P. (2023). *Let's Roll: Synthetic Dataset Analysis for Pedestrian Detection Across Different Shutter Types.* doi: 10.48550/arXiv.2309.08136

InterRegs Limited. (2021). *EU Regulation on Emergency Lane Keeping Systems Published.* Retrieved from https://www.interregs.com/articles/spotlight/229/eu-regulation-on-emergency-lane-keeping-systems-published

IPG Automotive. (2024a). *Camera-in-the-Loop.* Retrieved from https://www.ipg-automotive.com/en/products-solutions/test-systems/camera-in-the-loop

IPG Automotive. (2024b). *CarMaker.* Retrieved from https://www.ipg-automotive.com/

Jermakian, J. S. (2011, May). Crash Avoidance Potential of Four Passenger Vehicle Technologies. *Accident Analysis & Prevention*, *43*(3), 732–740. doi:

10.1016/J.AAP.2010.10.020

Kloukiniotis, A., Papandreou, A., Anagnostopoulos, C., Lalos, A., Kapsalas, P., Nguyen, D.-V., & Moustakas, K. (2022, June). CarlaScenes: A Synthetic Dataset for Odometry in Autonomous Driving. In *Proceedings of the 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops* (pp. 4519–4527). IEEE Press. doi: 10.1109/CVPRW56347.2022.00498

Li, J. (2023, February). Lane Detection with Deep Learning: Methods and Datasets. *Information Technology and Control*, *52*(2), 297–308. doi: 10.5755/J01.ITC.52.2.32841

Li, Y., Yuan, W., Zhang, S., Yan, W., Shen, Q., Wang, C., & Yang, M. (2024, March). Choose Your Simulator Wisely: A Review on Open-Source Simulators for Autonomous Driving. *IEEE Transactions on Intelligent Vehicles*, *9*(5), 4861–4876. doi: 10.1109/TIV.2024.3374044

Mordor Intelligence. (2024). *Driving Simulator Market Size & Share Analysis - Growth Trends & Forecasts (2024 - 2029).* Retrieved from `https://www.mordorintelligence.com/industry-reports/driving-simulator-market`

Moten, S., Celiberti, F., Grottoli, M., & van der Heide, A. (2018, June). X-in-the-Loop Advanced Driving Simulation Platform for the Design, Development, Testing, and Validation of ADAS. In *Proceedings of the 2018 IEEE Intelligent Vehicles Symposium* (p. 1-6). IEEE Press. doi: 10.1109/IVS.2018.8500409

Nie, X., Xu, Z., Zhang, W., Dong, X., Liu, N., & Chen, Y. (2022, July). Foggy Lane Dataset Synthesized from Monocular Images for Lane Detection Algorithms. *Sensors*, *22*(14), 5210. doi: 10.3390/S22145210

NVIDIA. (2020). *While the World Works from Home, NVIDIA's AV Fleet Drives in the Data Center.* Retrieved from `https://blogs.nvidia.com/blog/2020/05/19/nvidia-fleet-drives-in-the-data-center`

NVIDIA. (2024). *NVIDIA DRIVE Sim.* Retrieved from `https://developer.nvidia.com/drive/simulation`

Pan, X., Shi, J., Luo, P., Wang, X., & Tang, X. (2018, February). Spatial as Deep: Spatial CNN for Traffic Scene Understanding. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and 8th AAAI Symposium on Educational Advances in Artificial Intelligence* (pp. 7276–7283). AAAI Press. doi: 10.1609/AAAI.V32I1.12301

PapersWithCode. (2024). *Lane detection.* Retrieved from `https://paperswithcode.com/task/lane-detection`

Parmar, G., Park, T., Narasimhan, S., & Zhu, J.-Y. (2024). *One-Step Image Translation with Text-to-Image Models.* doi: 10.48550/arXiv.2403.12036

Parmar, G., & Zhu, J.-Y. (2024). *PyTorch Implementation of "One-Step Image Translation with Text-to-Image Models".* Retrieved from `https://github.com/GaParmar/img2img-turbo`

SAE International. (2021, April). Taxonomy and Definitions for Terms Related to Driving

Automation Systems for On-Road Motor Vehicles (Revised). *J3016*, *4970*(724), 1–5. doi: 10.4271/J3016_201609

Sternlund, S. (2016, September). The effectiveness of lane departure warning systems—a reduction in real-world passenger car injury crashes. *Traffic Injury Prevention*, *18*(2), 225–229. doi: 10.1080/15389588.2016.1230672

Tang, S., Zhang, Z., Zhang, Y., Zhou, J., Guo, Y., Liu, S., ... Liu, Y. (2023, July). A Survey on Automated Driving System Testing: Landscapes and Trends. *ACM Transactions on Software Engineering and Methodology*, *32*(5), 1–62. doi: doi.org/10.1145/3579642

Tran, L.-A., & Le, M.-H. (2019, July). Robust U-Net-Based Road Lane Markings Detection for Autonomous Driving. In *Proceedings of the 2019 International Conference on System Science and Engineering* (pp. 62–66). IEEE Press. doi: 10.1109/ICSSE.2019.8823532

Tremblay, J., Prakash, A., Acuna, D., Brophy, M., Jampani, V., Anil, C., ... Birchfield, S. (2018, June). Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization. In *Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops* (pp. 1082–1090). IEEE Press. doi: 10.1109/CVPRW.2018.00143

TuSimple. (2017). *TuSimple Benchmark.* Retrieved from `https://github.com/TuSimple/tusimple-benchmark`

TÜV SÜD. (2022). *Revision of the EU General Safety Regulation.* Retrieved from `https://www.tuvsud.com/resource-centre/stories/revision-of-the-eu-general-safety-regulation`

Unreal Engine. (2024). *dSPACE Drives Advancements in Autonomous Vehicle Testing.* Retrieved from `https://www.unrealengine.com/spotlights/dspace-drives-advancements-in-autonomous-vehicle-testing`

Valeo. (2024). *Smart Front Camera.* Retrieved from `https://www.valeo.com/catalogue/cda/smart-front-camera`

VI-GRADE. (2024). *VI-WorldSim.* Retrieved from `https://www.vi-grade.com/products/vi-worldsim`

Visure Solutions. (2024). *ASPICE: Definition, Compliance, Tools, and Certifications.* Retrieved from `https://visuresolutions.com/blog/automotive/aspice`

Wachenfeld, W., & Winner, H. (2016). The release of autonomous vehicles. In *Autonomous Driving: Technical, Legal and Social Aspects* (pp. 425–449). Berlin, Germany: Springer International Publishing. doi: 10.1007/978-3-662-48847-8_21

Wishart, J., Como, S., Forgione, U., Weast, J., Weston, L., Smart, A., ... Ramesh, S. (2021, February). Literature Review of Verification and Validation Activities of Automated Driving Systems. *SAE International Journal of Connected and Automated Vehicles*, *3*(4), 267–323. doi: 10.4271/12-03-04-0020

Xu, H., Wang, S., Cai, X., Zhang, W., Liang, X., & Li, Z. (2020, August). CurveLane-NAS: Unifying Lane-Sensitive Architecture Search and Adaptive Point Blending. In *Proceedings of the 16th European Conference on Computer Vision* (pp. 689–704).

Springer International Publishing. doi: 10.1007/978-3-030-58555-6_41

Yang, X., Yu, Y., Zhang, Z., Huang, Y., Liu, Z., Niu, Z., ... Li, S. (2022, July). Lightweight Lane Marking Detection CNNs by Self Soft Label Attention. *Multimedia Tools and Applications*, *82*(4), 5607—5626. doi: 10.1007/S11042-022-13442-6

Yu, F., Chen, H., Wang, X., Xian, W., Chen, Y., Liu, F., ... Darrell, T. (2020, June). BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning. In *Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 2633–2642). IEEE Press. doi: 10.1109/CVPR42600.2020.00271

Zakaria, N. J., Shapiai, M. I., Ghani, R. A., Yassin, M. N. M., Ibrahim, M. Z., & Wahid, N. (2023, January). Lane Detection in Autonomous Vehicles: A Systematic Review. *IEEE Access*, *11*, 3729–3765. doi: 10.1109/ACCESS.2023.3234442

Zheng, T., & Huang, Y. (2022). *PyTorch Implementation of "CLRNet: Cross-Layer Refinement Network for Lane Detection" (CVPR2022 Acceptance).* Retrieved from `https://github.com/Turoad/CLRNet`

Zheng, T., Huang, Y., Liu, Y., Tang, W., Yang, Z., Cai, D., & He, X. (2022, June). CLRNet: Cross Layer Refinement Network for Lane Detection. In *Proceedings of the 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops* (pp. 898–897). IEEE Press. (https://github.com/Turoad/CLRNet) doi: 10.1109/CVPR52688.2022.00097