Technical University of Applied Sciences Würzburg-Schweinfurt (THWS)
Faculty of Computer Science and Business Information Systems

Master Thesis

Simulation Environments and AI-based Autonomous Driving Solutions - a Comparative Evaluation

submitted to the Technical University of Applied Sciences Würzburg-Schweinfurt in the Faculty of Computer Science and Business Information Systems to complete a course of studies in Master of Artificial Intelligence

Mahesh Saravanan
K54723

Submitted on: 17.01.2024

Initial examiner: Prof. Dr. Christian Bachmeir
Secondary examiner: Mr. Leon Heller

Abstract

This thesis presents a comprehensive approach to evaluate and enhance autonomous vehicle simulators. The research comprises two primary works. Firstly, a systematic evaluation methodology is introduced, establishing a scoring system based on 73 discerning parameters to assist users in efficiently selecting a suitable simulator. This method streamlines the process, enabling users to assess simulator capability aligned with their specific needs.

Secondly, the thesis explores the development of a prototype simulator driven by a deep generative model. This model aims to simulate the transition of 2D lidar data and generate diverse scenarios crucial for training machine learning models in autonomous driving.

The study assesses the performance of the system of comparison by evaluating existing simulators and user preferences. Furthermore, the results of the proposed prototype simulator are discussed.

Zusammenfassung

In dieser Arbeit wird ein umfassender Ansatz zur Bewertung und Verbesserung von Simulatoren für autonome Fahrzeuge vorgestellt. Die Forschungsarbeit besteht aus zwei Hauptteilen. Erstens wird eine systematische Bewertungsmethodik eingeführt, die ein Punktesystem basierend auf 73 unterscheidbaren Parametern entwickelt, um Benutzer bei der effizienten Auswahl eines geeigneten Simulators zu unterstützen. Diese Methode vereinfacht den Prozess und ermöglicht es den Benutzern, die Fähigkeiten des Simulators entsprechend ihrer spezifischen Bedürfnisse zu bewerten.

Zweitens wird in dieser Arbeit die Entwicklung eines Prototyps eines Simulators untersucht, der durch ein tiefes generatives Modell gesteuert wird. Dieses Modell zielt darauf ab, den Übergang von 2D-Lidar-Daten zu simulieren und verschiedene Szenarien zu generieren, die für das Training maschineller Lernmodelle für das autonome Fahren entscheidend sind.

Die Studie evaluiert die Leistung des Referenzsystems durch die Bewertung bestehender Simulatoren und Benutzerpräferenzen und diskutiert die Leistung des vorgeschlagenen Prototypsimulators.

Acknowledgment

I would like to thank Prof. Dr. Christian Bachmeir from Technical University of applied sciences Würzburg Schweinfurt for providing me an opportunity to work on this topic and for his supervision throughout this endeavour.

I would also like to thank Mr. Leon Heller for his unwavering support, continuous guidance, and supervision throughout the course of this thesis. His invaluable ideas and insights significantly contributed to shaping this work.

I am also thankful to the other professors whose contributions fine-tuned and catalysed this research through their knowledge and feedback. Additionally, I express my appreciation to my family and friends for their support throughout this journey.

Contents

1 Introduction

1.1 Autonomous Vehicles

Transportation has played an important role in the human history, evolving from the invention of the rotary wheel to the modern road cars. The invention of fuel engines in the last century marked a significant revolution, impacting faster goods transportation and extending to civilian life with the introduction of road cars. This period also witnessed the development of proper road infrastructures to accommodate the advancements in vehicle performance. The fourth industrial revolution mark a start of new era where intelligent systems-controlled machines are developed which reduces the manual errors and enhancing performance. The automotive industry is also undergone this transformation with various intelligent driver assistance systems like Advanced Driver Assistance Systems (ADAS) which not only reducing human effort in driving but also ensuring safety. Engineers and scientists globally are working to enhance the capabilities of intelligent systems, aiming for complete autonomous driving, eliminating the need for human guidance.

The concept of autonomous driving dates back to 1926, with the development of a radio-controlled car in New York City [32]. In 1960s, the Transport and Road Research Laboratory in the United Kingdom laid magnetic cables beneath the road, serving as a path detection tool for self-driving cars and tested its performance. Later, in 1995 Mercedes Benz achieved a significant milestone with the development of first self-sufficient autonomous vehicle, a retrofitted Mercedes S class equipped with efficient cameras and exclusive processors for parallel computing. This vehicle reached a maximum speed of 175 kmph, covering 1500 kms from Munich, Germany, to Copenhagen, Denmark, performing various manoeuvres in traffic without human assistance [10]. In the late 2000s, carmakers like Toyota and Volvo, along with tech companies such as Google and Waymo, developed their prototypes of Autonomous Vehicles (AVs). The parallel advancements in the field of Artificial Intelligence and efficient hardware accelerated research in self-driving technology. Notably, improvements in cameras and GPUs facilitated fast and efficient processing, while intelligent algorithms like neural networks laid the foundation for Autonomous Vehicles (AVs). In 2014, Tesla Motors launched Model S, a semi-autonomous driving car. This vehicle featured a various assistance feature, including lane detection, autonomous braking and parking, and speed limit recognition

using computer vision. Tesla's entry into functional semi-autonomous driving marked a notable advancement in the field. During the same year, the Society of Automotive Engineers (SAE) had drafted a 6-level taxonomy for autonomous driving. This framework provided a standardized and structured classification system for assessing the level of autonomy in vehicles. Recognizing the growing impact of autonomous driving, several countries, including the United States of America, the United Kingdom, and Japan, made decisions to formulate laws addressing autonomous driving in the next few years. Various leading car manufacturers, including Mercedes Benz, BMW, and Volvo, have introduced their commercial fully or semi-autonomous production vehicles. They are continually enhancing their automation levels, as classified by the Society of Automotive Engineers (SAE). For instance, in 2022, Mercedes Benz achieved SAE Level 3 automation with its S-Class and EQS models, equipped with advanced driving technologies like an automated lane-keeping system. However, Tesla Motors has been a leader in this field, showcasing superior performance. The SAE J3016 classifies autonomous vehicles into six levels, ranging from fully manual driving to fully automated systems as described in Table 1.1 [40]

Level	**Name**	**Description**
0	No driving automation	This level involves no driving assistance system, and the entire driving process is manually controlled.
1	Driver assistance	Vehicles at this level feature a single automated system under the driver's supervision.
2	Partial driving automation	Multiple automated systems, such as automated steering and braking, work simultaneously under the driver's attention. The driver must be ready to take control at any moment.
3	Conditional driving automation	This level involves a system with an intelligent algorithm capable of making decisions like overtaking navigation. However, human attention is still required.
4	High driving automation	At this level, the vehicle can make optimistic decisions in the event of a failure, reducing the need for human interaction. This level is often suitable for confined areas or regions.
5	Full driving automation	Here, the vehicle can perform all driving tasks without human interaction, comparable to an experienced human driver.

Table 1.1: Levels of Driving Automation

Driver-less vehicles find applications not only in passenger cars but also across various areas like shuttling people on campuses, moving goods in warehouses, and in military operations. Especially in industries, these automated machines could transport goods across the premises which save time and minimize human error. However, ensuring their safety of operation is crucial especially when it comes to road cars. Governments world-wide are creating regulations for autonomous mobility. Moreover, vehicle manufacturers are enhancing hardware and software while conducting extensive testing to ensure safety and prevent failures.

1.2 Simulators

Simulators are the software platforms which imitate real-world systems, replicating their dynamics and features. They come in different types: software, hardware, or a mix of both. These systems replicate various functions, interacting with users, taking inputs, and providing feedback to the users. For example, driving simulators create visual environments and hardware setup for driver training and testing, Which takes control commands from users and update their states. Simulators can be classified based on various features Some of the examples are

Discrete Event Simulation: Each step happens at distinct intervals, which are mapped to specific durations. Driving simulators fall into this category, where observation states change at each step. The time duration of a single time step is defined by required precision.

Continuous Simulation: In this type of simulation, time moves continuously without distinct steps, backed by differential equations. These are useful for modelling continuous events like microbe growth in biology.

Stochastic Simulation: This type of simulation introduces a controlled amount of randomness within sensible limits in the parameters. It's helpful for studying systems with random noise, like analysing daily usage patterns on a social media site.

Deterministic Simulation: These systems operate using deterministic algorithms without any randomness. They're commonly applied in engineering. The outcomes of these simulations remain consistent and reproducible for a specific set of parameters.

HiL simulation: Hardware-in-the-Loop Simulation(HiL), is a testing method integrating physical hardware components into a simulated environment. In HIL setups, real hardware, like sensors and control systems, connects to a computerized simulation. This allows the actual hardware to interact with a virtual environment, enabling thorough testing of systems under various scenarios. In Autonomous vehicle development, car sensors and control units are connected to a simulated driving scenario, allowing engineers to assess how the physical components respond to different conditions before deploying them in real-world situations. HIL testing ensures more robust and validated performance of hardware and systems in a controlled, simulated environment before practical implementation.

Users can tweak these parameters to test and tune the results of the system. Usually, simulations help to find the optimal parameters and it have diverse applications, for events spanning millions of years to nanoseconds. Simulations play a vital role in result analysis, safety engineering, and design processes. They save significant time during testing and can enhance the testing outcomes. For instance, testing the driving of an autonomous car for 10,000 km in the real world demands a lot of time and resources, whereas using a simulator significantly reduces this time-frame. However, there's often a discrepancy between the real world and simulations, which might lead to errors in analysis. In fields like machine learning and robotics, it contributes a crucial role. One of the three paradigm of Machine learning is reinforcement learning where a virtual agent within a simulator takes diverse actions under various conditions and learns from the feedback provided. In robotics, algorithm which drives the robots are trained using simulation for tasks like localization, pick and place, faulty detection...

1.3 Problem Statement

1.3.1 Lack of system of comparison

In the growing domain of autonomous vehicle research, simulator plays a significant role in the process of development such as algorithm training, performance evaluation across diverse environments and comparison of various algorithm. There are numerous simulators, varying in types and functionalities, available in the community for direct or indirect use in research works. In addition, the domain of self-driving vehicle research itself is broad, includes distinct applications such as indoor robots, industrial autonomous vehicles (AVs), on-road vehicles, and semi-autonomous vehicles. Each application requires a unique simulator tailored to its specific requirements.

The challenge arises due to the diversity among these simulators and their requirements, each comes with its own set of advantages, features, and limitations. Consequently, com-

paring and selecting the most suitable simulator becomes a complex task. For instance, a simulator might perform well in numerous aspects, yet fail to simulate a crucial sensor required by a user, making it unsuitable for that specific application. Conversely, another simulator might lack certain features but could be better suited for the user.

The aim of this thesis is to establish a concrete set of metric for comparison purposes. The proposed metric will be generic and adaptable to individual user preferences. Users will have the flexibility to adjust the importance of criteria based on their requirements and they are universally applicable across all types of autonomous vehicle simulators. Through this systematic approach, users could able to assign personalized ratings to simulators, facilitating informed decision-making in selecting the most suitable simulator according to their preferences and needs.

1.3.2 Simulator based on Generative AI

Generative AI had gained momentum in recent years, continuously improving its performance. This field primarily uses deep neural networks, trained on any provided data and learning its distribution. These trained models can generate novel, meaningful data samples which are not present in the training set but similar to it.

In the domain of autonomous vehicle simulators, rendering new scenarios and environments is a one of its objectives. This feature enhances its suitability for algorithm testing across diverse environments, potentially improving the driving algorithms's performance. It's also important that these new scenarios closely resemble real-world features while being meaningful and plausible.

Sensors in Autonomous cars play a important role in understanding the real world where the vehicle navigates. These vehicles utilize data from various of sensors such as cameras, Lidar, and radar to perceive their surroundings. The second part of this thesis aims to design and develop a simulator driven by a Generative AI model trained on real-world sensory data. The expected outcome is the generation of new, meaningful sensory data can effectively represent diverse and realistic environments. The simulator will operate on discrete time steps, ensuring that the generated data remains relative to preceding time steps.

1.4 Structure of the thesis

The thesis structure is as follows: Chapter 2, the Literature Review, is divided into three parts. The first segment analyses feature of popular autonomous vehicle simulators,

followed by the discussion of existing comparative metrics for these simulators. The final part explores some of the relevant GAN architectures. Chapter 3 outlines the proposed method for comparing AV simulators, while Chapter 4 details the idea of Generative Model-based Simulator. Chapter 5 presents and discusses results, followed by the Conclusion and possible future works in this thesis.

2 Literature Review

This chapter explores different works related to the topic. The initial section examines the existing state-of-the-art autonomous vehicle simulators. The second part provides a summary of previous methodologies used to define a comparative metric for evaluating autonomous vehicle simulators. The last part of this chapter discusses about the different types of Generative Adversarial networks (GANs), a type of Generative AI algorithm utilized in this thesis.

2.1 State-of-art-Simulators

Numerous simulators are available in the community, but this section analyse some relevant and popular simulators and highlighting their advantages, limitations and applications.

2.1.1 CARLA

CARLA, (Car Learning to Act) is an open-source software developed collaboratively by the Computer Vision Centre (CVC) and the Barcelona Super computing Centre (BSC) in partnership with the Toyota Research Institute. It is primarily designed for autonomous driving research and development, which features diverse and realistic environments, various climates and wide range of sensors. CARLA operates on a server-client architecture, built on Unreal Engine 4 and utilizing the OpenDRIVE standard 1.4 to define roads and urban settings. This unique structure allows the server to manage physics of simulator and computation while enabling user to communicate the server through C++ and Python APIs, providing scalability.

A notable feature of CARLA is its seamless support for developing, training, and validating machine learning algorithms. Researchers had experimented various algorithms like modular pipelines, imitation learning, and reinforcement learning within this simulator [11]. Leveraging Unreal Engine 4, CARLA offers high-quality, realistic rendering of environments. Figure 2.1 [11] showcases scenes from the simulator in different weather

conditions. Additionally, it supports an array of built in sensors such as camera, LiDAR and provides the user with various metadata, and ground truth. Moreover, CARLA offers access to diverse digital assets (actors, buildings) within the environment, meticulously designed to maintain a high level of realism [11]. However, it currently offers support for only two pre-defined urban maps covering 2.9 km and 1.4 km, which limits its diversity and generalization capabilities.

Figure 2.1: Scenes from the CARLA simulator in different weather conditions. Clockwise from top left: Day time, Rain, Shortly after rain, Sunset

2.1.2 LGSVL

LG Silicon Valley Lab (LGSVL) is an open-source simulation engine developed by LG Electronics. It utilizes the Unity gaming engine to render photorealistic environments and taking advantage of technologies like the High-Definition Render Pipeline (HDRP) from Unity [52].

This simulator is developed in two parts: the Simulation Engine and the User Autonomous Driving (AD) Stack [39]. The Simulation Engine, a customizable open-source platform, receiving its inputs from AD stack and simulate the environment, sensors, and vehicle dynamics. The AD Stack comprises three major layers: Perception, Planning, and Control, offering various user-configurable functionalities. The AD Stack and the Simulation Engine is connected through communication bridge interface, such as Cyber

RT, ensuring seamless integration [39]. While the simulator comes with various default sensors including cameras, LiDAR, and Radar, its unique feature lies in its adaptability. Users can build and configure their own sensors and importing models of real-world sensors as plugins. For instance, the plugin for Velodyne VLP-16 LiDAR replicates point cloud generation similar to its actual counterpart. These sensors' data and its mounting positions are defined through JSON-formatted text, simplifying their utilization. Figure 2.2 showcases the array of default sensors available within this simulator.

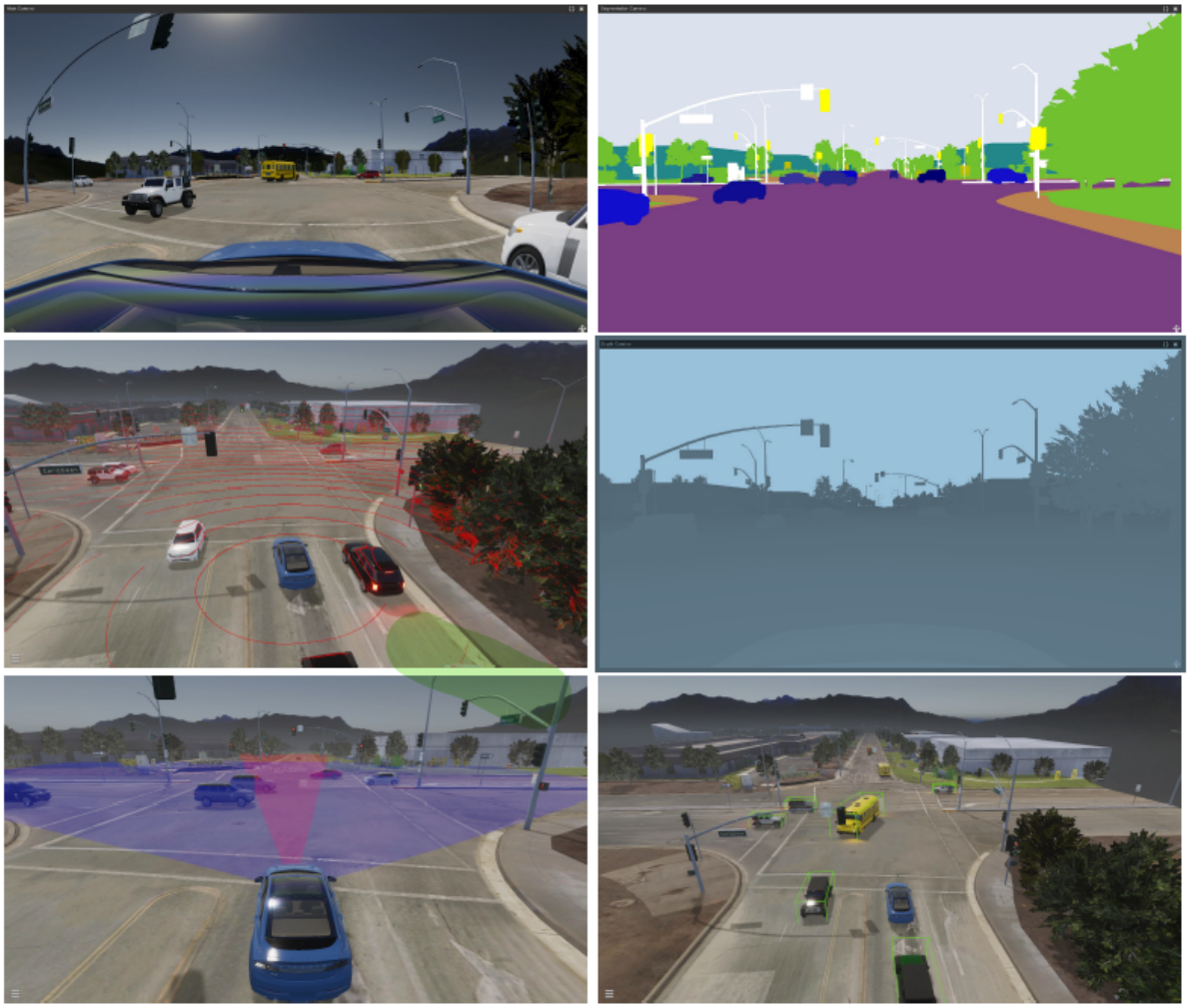


Figure 2.2: Different types of sensors. Left (top to bottom): Fish-eye camera, LiDAR, Radar; Right (top to bottom): Segmentation, Depth, 3D Bounding Box.

2.1.3 SUMMIT

SUMMIT (The Simulator for Urban Driving in Massive Mixed Traffic) is an open-source simulator developed as an extension of the CARLA simulator [11], inheriting its physics

and visual realism. Unlike many other simulators that predominantly simulate rule-based traffic with minimal randomness, SUMMIT stands out for its ability to replicate the aggressive and chaotic nature of real-world traffic. This distinctive feature attracts users interested in training and testing algorithms for vehicles navigating unregulated traffic scenarios. SUMMIT employs the 'Context-GAMMA', a velocity-space optimization crowd behaviour algorithm [5] to simulate traffic behaviour geometrically and topologically. Additionally, it utilizes real-world maps from OpenStreetMap, extracting features such as roads, sidewalks, and roundabouts which are incorporated into the simulator, enabling the replication of real-world maps. An illustrative example can be seen in Figure 2.3, which showcases the comparison between a real map and its counterpart with unregulated traffic behaviour in SUMMIT at the Magic Roundabout in England.

(a) Real map (b) Simulated map

Figure 2.3: Scenes in the real world and corresponding scenes in SUMMIT

2.1.4 Gazebo

Gazebo [27] stands as a prominent open-source simulator extensively utilized in the field of robotics and autonomous systems. Originating from Carnegie Mellon University in the early 2000s, its primary focus remains on mobile robots. The unique feautre of Gazebo is its ability to accurately emulate the physics of non-player elements within the environment. This simulator integrates the Open Dynamics Engine [36], crafted by Russel Smith, a widely adopted physics engine within the open-source community. Tailored to replicate the dynamics and kinematics associated with articulated rigid bodies, this engine boasts a comprehensive range of features. These includes various joints, collision detection, mass and rotational functions, as well as a multitude of geometries, including arbitrary triangle meshes.

Gazebo's scalability is another notable aspect, enabling the concurrent simulation of up to ten robots. This capability proves advantageous particularly in applications involving

multiple robots, such as within the domain of autonomous driving. The inclusion of multiple robots, sensors, and actuators, collectively referred to as a model, is seamlessly facilitated through APIs, allowing easy integration into the simulator. Furthermore, Gazebo extends its functionality by facilitating the effortless addition of new 3D user-defined objects using OpenGL, a standard library for 3D object creation. Additionally, GLUT (OpenGL Utility Toolkit) [50] offers an interactive user interface, enhancing its accessibility and usability.

While Gazebo excels in accurately simulating various physical phenomena, the challenge arises when attempting to replicate the extensive environments requisite for real-world autonomous driving scenarios. Despite this limitation, owing to its open-source nature, users actively engage in developing custom simulators tailored to their specific requirements such as Gazebo golf cart [43] which is developed for autonomous golf cart mobility within the golf field.

2.1.5 TORCS

TORCS [56], or The Open Racing Car Simulator, was developed in the early 2000s as an open-source platform for simulating car races. Its standout feature is its ability to adjust car settings, which is crucial in racing simulations. It falls under discrete-time simulation, where transitions are processed in small steps of 0.002 seconds. In TORCS, participants are called "robots" and are loaded as separate parts of the system. This setup allows developers to create new smart agents that follow basic rules for robot code.

TORCS is good at mimicking real racing situations, including how cars move, track designs, weather changes, and interactive settings. This flexibility also makes it useful for academic purposes, letting researchers test artificial intelligence self-driving algorithms.

However, there are challenges, like handling complex simulations and making the interactions between cars and the environment more realistic. TORCS benefits from a lively community of developers, researchers, and racing fans because it's open-source, letting people collaborate and improve the platform together.

2.1.6 NVIDIA Drivesim

The NVIDIA DRIVE Simulator is an integral part of the NVIDIA DRIVE platform, offering a comprehensive solution for autonomous vehicle development[35]. One of its notable features is the ability to simulate diverse and complex driving scenarios, allowing

developers to test their algorithms in a wide range of situations. This versatility is crucial for ensuring the robustness and reliability of autonomous systems.The simulator leverages the power of NVIDIA GPUs to provide high-fidelity graphics and realistic physics, creating an immersive environment for testing perception, planning, and control algorithms. This realistic simulation is essential for training and validating autonomous driving systems before they are deployed on real roads, enhancing safety and reliability. The figure 2.4 shows an environment captured by cameras mounted at different positions of the car within Drivesim.

Figure 2.4: Environment captured by cameras at different positions in Nvidia drivesim

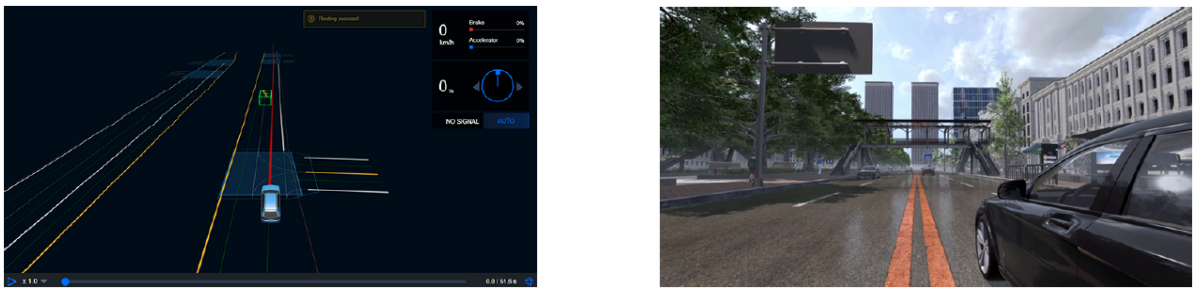
In terms of scalability, the NVIDIA DRIVE Simulator can handle large-scale simulations, accommodating the complexity of urban environments and traffic scenarios. This scalability is a key factor in its effectiveness for testing and refining autonomous vehicle algorithms under diverse and challenging conditions.The simulator's integration with the broader NVIDIA DRIVE platform enhances its capabilities. It allows seamless collaboration with other components of the platform, such as the hardware and software stack optimized for autonomous driving applications. This integration ensures a cohesive and efficient workflow for developers working on various aspects of autonomous vehicle technology.

Numerous case studies and research projects have demonstrated the efficacy of the NVIDIA DRIVE Simulator. This simulator significantly reduces the time and costs associated with developing autonomous vehicles, as virtual testing minimizes the need for extensive real-world testing. however, the simulator's reliance on high-performance GPUs could pose accessibility challenges for users without access to such hardware. Simplified traffic dynamics, difficulties in accurately simulating dynamic weather, and potential algorithmic shortcomings are factors that developers should consider. Additionally, the cost of implementation, both in terms of resources and personnel training, might be a deterrent for some.

2.2 Comparative Study on Simulators

This section examines notable studies that compare existing autonomous vehicle simulators and summarizes their approach and results.

In Guan Yang et al. work (2021), "Survey on Autonomous Vehicle Simulation Platforms," [58] the team extensively researched different autonomous vehicle simulation platforms. The objective of a simulator is broken down into five parts: Static environment simulation, Dynamic environment and behaviour simulation, Traffic flow simulation, Sensor simulation, and Vehicle dynamics simulation. They also established a taxonomy for existing simulators, categorizing them into Point Cloud-based and 3D Engine-based platforms. Point-based simulators, such as CarCraft from Waymo [55] and Apollo from Baidu [31], reconstruct the environment based on simulated sensor data. Figure 2.5a displays the map from Apollo, a point-based simulation platform. On other hand, 3D engine-based platforms, like PanoSim [42], utilize gaming engines like Unity and Unreal to render environments following laws of physics (Figure 2.5b) [58].



(a) Apollo Simulator (b) PanoSim Simulator

Figure 2.5: Point-Based and 3D Engine-Based Simulators

They further created a table comparing some of popular and relevant simulation platforms and their features as shown in figure 2.6 [58]

Type	Simulation	Country	Company/Organization	Read-world map	Unregulated behaviors	Traffic flow	Sensor	Vehicle dynamics
Point cloud	Autoware	Japan	Autoware	✓			✓	✓
	SimMobilityST	Singapore	SMART	✓		✓		
	SUMO	Germany	DLR	✓		✓		
	Zuks	America	Zoox	✓			✓	✓
	Carcraft	America	Waymo	✓			✓	✓
	Apollo	China	Baidu	✓	✓		✓	✓
3D engine	Sim4CV	Saudi Arabia	Sim4CV					✓
	Carla	Spain	IntelToyota			✓	✓	✓
	AutonoVi-Sim	America	UNCUCF		✓	✓	✓	✓
	SUMMIT	Singapore	NUS	✓	✓	✓	✓	✓
	AirSim	America	Microsoft					✓
	CarSim	America	Mechanical Simulation cooperation					✓
	LGSVL	South Korea	LG	✓			✓	✓
	Prescan	Germany	Siemens			✓	✓	✓
	CarMaker	Germany	IPG AUTOMOTIVE	✓	✓		✓	✓
	PanoSim	China	PanoSim Technologie				✓	✓
	TAD sim	China	Tencent	✓	✓	✓	✓	✓
	Octoppus	China	Huawei		✓			
	51Sim-One	China	51VR	✓			✓	✓

Figure 2.6: A Comparison table of various simulator

Although this table aids in comparing simulators roughly, it doesn't compare sufficient features for making a concrete decision, and lacks a single metric defining the level of usability of the simulator for a user. While the categorization of simulators is provided, a clear comparative method among simulators is not clearly defined. In Md Salman Ahmed et al.'s work (2016), an extensive study on connected vehicle simulators was presented [1]. The focus was on the domain of connected vehicles, including vehicle-to-vehicle and vehicle-to-server communication. The paper assessed several simulators based on their memory consumption, computing environment (Sequential or Parallel), and the number of vehicles they could handle. However, these results are specific to the connected vehicle domain and may not be applicable to other types of autonomous vehicle simulators effectively.

2.3 Generative Adversarial Network

Generative adversarial networks (GAN) mark a significant advancement in the domain of Generative AI, first introduced in 2014 by Goodfellow et al. in the paper "Generative Adversarial Networks" [14]. Since then, GANs have gained substantial momentum in the field of Generative AI especially in image generation. This section discusses some noteworthy works within the domain of GANs.

2.3.1 VGAN(Video Generative Adversarial Network)

VGAN, developed by Carl Vondrick et al. [53], specializes in generating videos with its scene dynamics. The model is capable of generating videos up to a second at full frame rate. Its training involves over 2 million pre-processed videos sourced from the internet, categorized into four distinct groups: golf courses, hospital rooms, beaches, and train stations.

The architecture of VGAN employs a standard Generator-Discriminator structure. The generated video is segmented into two features: foreground and background, assuming a fixed camera resulting in a static background. The generator comprises two network streams dedicated to foreground and background, respectively. The foreground stream consists five layers of 3D spatiotemporal convolution layers (time x width x height), which upsamples the information from a low-dimensional latent code z, sampled from a standard normal distribution. A masking layer m is introduced before the final layer, outlining the pixels of objects in the foreground. Meanwhile, the background stream utilizes five layers of 2D convolution layers (width x height), responsible for generating a background b. The background stream uses 2D convolution layers as the background

is assumed to be same for all the generated frames. The synthesis of foreground and background follows the equation: $m \cdot f + (1 - m) \cdot b$

The resultant video, comprising 32 frames with dimensions of 64x64, is generated from a 100-dimensional latent code sampled from a normal distribution. The Discriminator is designed for two primary objectives: classifying realistic scenes and recognizing plausible and smooth motion between frames. It mirrors the architecture of the foreground stream of generator with a five-layered spatiotemporal convolutional setup, employing downsampling instead of upsampling. The final layer outputs a binary classification (real or not). Batch normalization and ReLU activation functions are used after every layer in both the generator and discriminator. The VGAN is trained using Adam optimizer [26] with a batch size of 64. Results demonstrate the model's ability to generate videos with a sharp background and a slightly blurry foreground. While the resolution of the foreground might be blurred, the motion of the generated foreground is convincing. However, the user had no control over the content of generation. Figure 2.7 illustrated the frames of various generated videos.

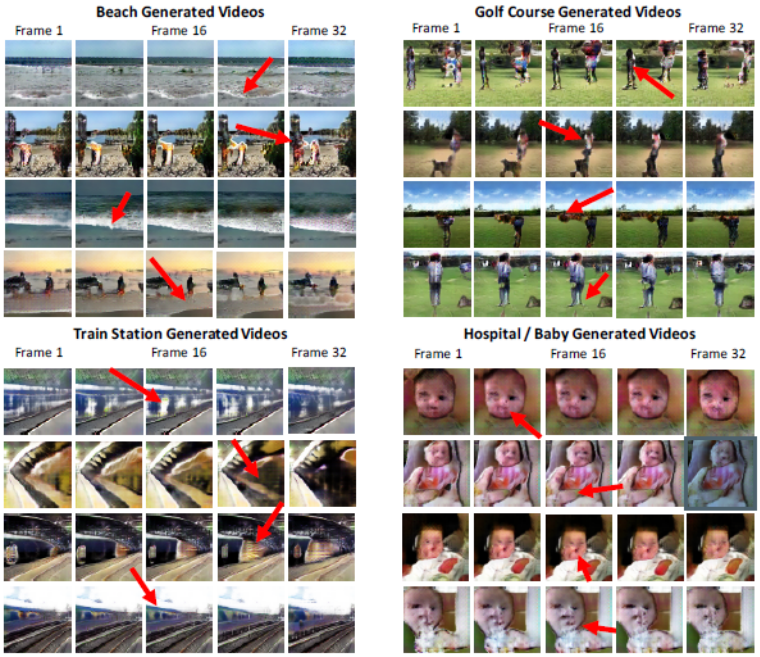


Figure 2.7: Frames of Videos generated by VGAN

2.3.2 RV-GAN

The RV-GAN, introduced by Sonam Gupta et al., [16] stands for Recurrent Variational Generative Adversarial Network and serves the purpose of unconditional video generation. This particular GAN employs a specialized variant of LSTM [19] known as

TransConv LSTM. The innovation lies in modifying the conventional ConvLSTM by integrating a transpose convolutional structure into its input-to-state transitions. This unique design allows the network to simultaneously capture spatial and temporal relationships across layers within the TC-LSTM unit. (See Figure 2.8 for a diagrammatic representation of the TransConv LSTM [16].)

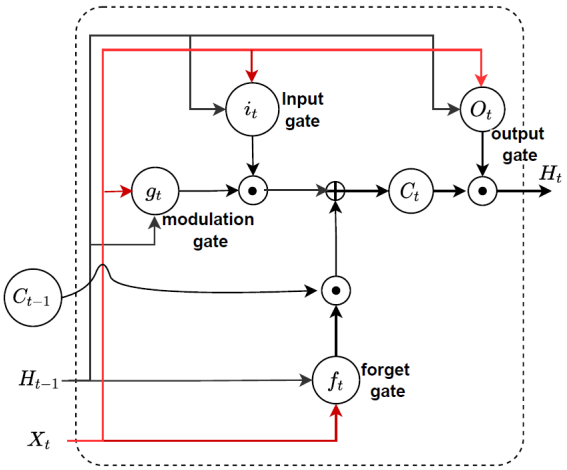


Figure 2.8: Diagrammatic representation of the TransConv LSTM. Ht-1,Ct-1,Ht,Ct are the hidden states and cell states at previous and current timestamps. Xt is input to the TC-LSTM unit.[16]

The composite model comprises a generator built with blocks of TC-LSTM units and incorporates 2 discriminators similar to Mcocgan [51]. In the generator, noise sampled from a random distribution is fed at each time step, along with the inputs of the hidden state and cell state from the preceding time step to account for temporal information. This process generates frames that exhibit strong correlation with previous ones. On the other hand, the discriminator is structured using 2D and 3D convolution layers, specifically for image and video discrimination, respectively. The entire network is trained on the combined loss function derived from both discriminators. Training of the model occurs on datasets such as Weizmann [15], UCF101[45], and MUG[2], with evaluation based on the Frechet Inception Distance metric (FID) [18]. Due to its recurrent structure, the proposed method demonstrates the ability to generate high-quality videos, extending up to 2 times the length (32 frames) of the training videos during inference. The authors substantiate the versatility of this GAN by showcasing its adaptability to class conditional video generation. RV-GAN exhibits superior performance compared to most other methods, falling short only when compared to V3GAN[25] on UCF101.

2.3.3 Imaginator

Imaginator is a conditional Generative Adversarial Network (GAN) developed by Yaohui Wang et al [54]. Its primary aim is to produce human videos depicting various expressions. Unlike VGAN [53], this model generates videos conditioned on specific class labels for expressions for example smiling, jogging... The model is designed to segment the generated video into spatial and temporal parts.

Imaginator comprises a generator and two discriminators. The generator adopts an encoder-decoder architecture with skip connections. It takes a static image featuring a person's face as input and encodes it into a latent vector p. In addition, this vector is concatenated with one-hot encoded class label c, representing the expression, and random noise sampled from a standard normal distribution. This fusion embeds spatial (p) and temporal information (c) into the latent vector. The decoder, structured as a (1+2)D convolution layer, explicitly separates temporal and spatial information. It mirrors the dimensions of the encoder's architecture and had skip connections from encoder, ensuring that each decoder layer retains embedded spatial details from the corresponding encoder layer. Moreover, the embedded class label vector is integrated into every decoder layer, ensuring the preservation of temporal information throughout the model. The generated video consists of a fixed number of frames and the last layer of the generator outputs an image with all the frames.

Two discriminators serve distinct purposes: 'd_i' evaluates individual frames of the generated video to classify real from fake based on appearance, while 'd_v' examines the sequence of frames alongside the class label to classify the dynamics within the frames as real or fake. The Generator is optimized on the combined loss function from both discriminators and a reconstruction loss for corresponding frames. Each discriminator's loss function optimizes the weights of corresponding discriminator. The ADAM optimizer with the same learning rate is used across all generator and discriminator components.

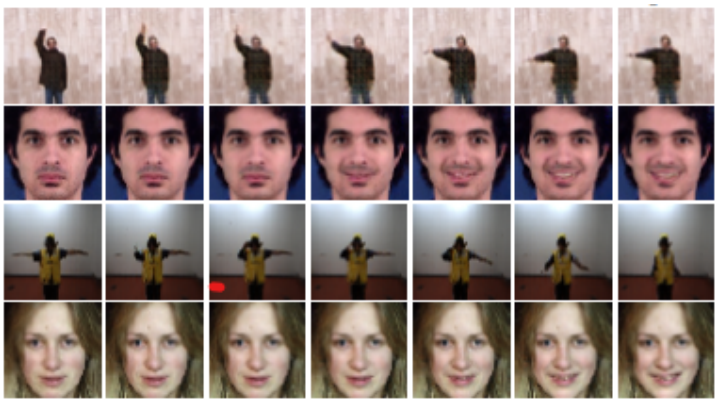


Figure 2.9: Frames of video generated by Imaginator

Evaluation metrics such as video Frechet Inception Distance (FID) [18], Peak Signal-to-Noise Ratio (PSNR), and Structural Similarity Index Measure are used to evaluate the performance of the generated videos and to monitor the training progress. Figure 2.9 provides a visual representation of the frames within the generated video.

The model was trained on various widely used datasets like MUG Facial Expression Dataset [2], NATOPS Aircraft Handling Signal Dataset [44], Weizmann Action Dataset [33], and UvA-NEMO Smile Dataset [49]. Comparative analysis shows the superior image quality compared to VGAN. Furthermore, the model's ability to control the content of generated videos using class labels holds significant potential across various applications.

2.3.4 MoCoGAN

Motion and content decomposed GAN (MoCOGAN), a Video Generation GAN introduced by Sergey Tulyakov et al. in 2016 [51], operates on a unique architecture that segments videos into Content and Motion. This segmentation allows the model to sample inputs separately from the Content and Motion subspaces (unsupervised). Notably, this architecture enables the model to generate videos depicting the same content with different actions or the same action with different content. While designed for shorter video durations, MoCOGAN doesn't require a fixed length for the generated videos. Given the shorter duration, the video's subject is assumed to remain constant.

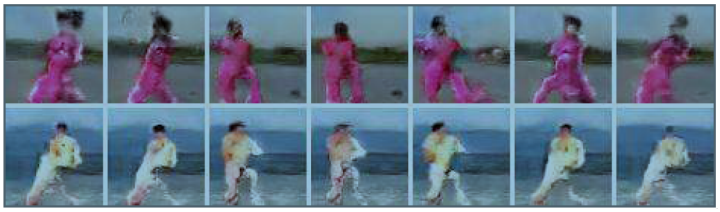
The model comprises four networks: a Recurrent Neural Network (R_m), Generator (G_1), Image Discriminator (D_i), and Video Discriminator (D_v). The Generator sequentially produces frames by taking a latent image Z as input, containing z_c and z_m. The random vector z_c represents the video's content and is sampled from a content subspace, remaining constant throughout the video as the subject remains unchanged. On the other hand, the latent vector z_m, which determines the motion trajectory of the subject, and it is recursively outputted by Rm by sampling from a motion subspace at each timestep.

The R_m parameters are learned during training, as not all motion trajectories are physically possible. The generator G takes Z (combination of z_c and z_m) and generates video frames sequentially. The Image Discriminator assesses frame quality, while the Video Discriminator evaluates subject motion in the video. The RNN is trained exclusively on the Video Discriminator loss, while the entire Generator network is trained on the combined loss of the discriminators.

The model's training utilizes the Weizmann Dataset [33] 2.10a and Tai-Chi Dataset [48] 2.10b, and its performance was assessed using the Average Content Distance (ACD). Visual representations in Figure 2.10 depict video clip frames generated by the MoCo-

GAN model. Notably, MoCoGAN had outperformed VGAN [53]and TGAN [41] in

(a) Weizmann Dataset



(b) Tai-Chi Dataset

Figure 2.10: Frames of video generated by MoCoGAN

image quality within the videos and offers the flexibility of generating videos of varying lengths. However, it lacks direct control over content creation like Imaginator as content is randomly sampled from the content subspace

2.3.5 Pix2Pix GAN

Pix2Pix, introduced in 2016 by Philip Isola et al [22], is an image translation technique using Conditional GANs. Image translation involves transforming images from one domain to another, such as converting grayscale images to colour images. In [17], the method was applied to translate satellite terrain images into Google Maps style images.

The structure of the conditional GAN resembles that of a Normal GAN, comprising a Generator and a Discriminator. In [17], the generator adopts an encoder-decoder architecture, where the encoder consisting of blocks that include a convolution layer, Batch Normalization, and ReLU activation function. The decoder mirrors the encoder, employing skip connections (known as U-Nets) between corresponding layers to retain information and prevent loss during transmission through bottle necks [22]. The Generator takes an image from the input domain and produces an image in the target domain. On the other hand, the Discriminator intakes the paired images from both domains, uses standard convolutional layers, converging in a final layer that determines the given image (real or fake). Using PatchGAN [23], the Discriminator segments images into 70x70 patches and performs binary classification for each patch, this makes the discriminator suitable for any given image size.

Training involves computing the reconstruction loss between the generated image and the target image. The generator's parameters are optimized to minimize this loss. The composed model training combines the Discriminator loss and the reconstruction loss using a weighted sum for gradient update. The dataset used for training, consisting of 1097 paired images of satellite and Google Maps images. Each image is pre-processed and rescaled to 256X256 pixels before training. Figure 2.11 illustrates the satellite images and its corresponding google map image and generated image.

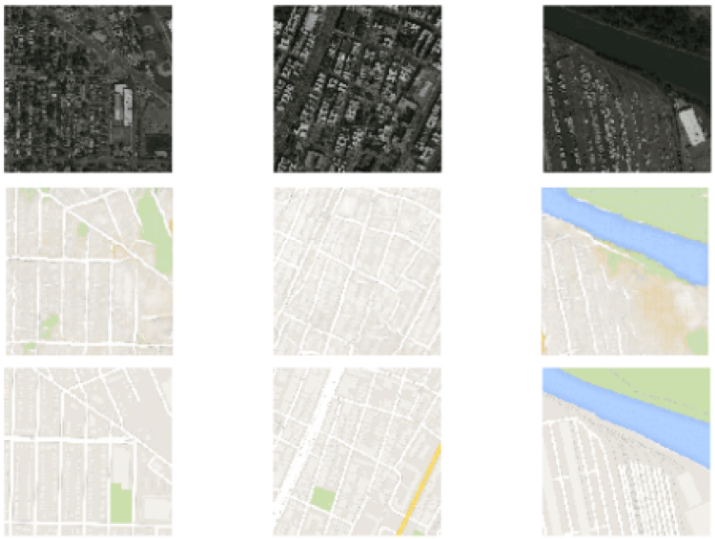


Figure 2.11: The satellite images [top] and its corresponding google map image [middle] and generated image after 10 epochs [Bottom]

While image translation using this method has diverse applications, this method often requires paired datasets, which is difficult to obtain in most of the cases.

2.3.6 SenseGen

Sensgen, a deep learning architecture introduced by Moustafa et.al., [3] is devised to generate synthetic sensor data. This framework addresses privacy concerns inherent in utilizing real data by generating sensor data from mobile phones and wearables. Its primary objective is not only to maintain essential statistical characteristics but also to replicate the comprehensive statistical properties of authentic data, serving various applications such as big data analytics. The methodology relies on a generative adversarial network featuring a generator and a discriminator. The overarching aim is to train the generator to successfully pass the discriminator's validation. To facilitate this training, the human activity recognition dataset [4] , encompassing accelerometer and gyroscope data capturing diverse human activities, is used. The generator architecture comprises

three layers of LSTM cells [19] specialized in generating time series data. Following this, a Gaussian Mixture Model (GMM) is introduced to inject stochasticity into each time step of the generated data. The LSTM cells predict the weights and parameters of the GMM, with the sampling GMM outputs guiding subsequent time-step inputs. This composite model is designed to learn the intricate distributions of the GMM, effectively representing realistic sensory data. Training of the generator continues until the generated data achieves an approximate 50 percent indistinguishability rate by the discriminator, validated through visual inspection demonstrating its close resemblance to real data. Due to the unique nature of the generated data, specific evaluation metrics are not employed, as they require tailored measures specific to the dataset in use.

3 Framework for Comparative Study

The suitability of an autonomous vehicle simulator for a particular user varies significantly based on several factors. These factors include various aspects such as the intended application, sensor support availability, integration of real-world components like vehicles and maps, computational resources... Within this proposed method 73 such parameters are identified and these parameters collectively contribute to formulate a single score for each simulator. This score serves as a means for comparison between different simulators.

3.1 Comparison Parameters

The identified parameters are categorized into six broad classifications: Sensors, Actors, Environment, Vehicle, Platform, and Driving algorithm. Each parameter is evaluated using a scoring system ranging from 0 to 1 where 0 being worse and 1 is best. In cases of binary classification, where a feature's availability is assessed, the corresponding parameter receives a score of 0 if the feature is unavailable and 1 if it is available.

3.1.1 Sensors

Sensors serve as the tool to perceive the vehicle's surrounding environment. They play a crucial role in driverless vehicles by providing inputs to the algorithm, which determines subsequent actions of the vehicle. In autonomous vehicles, multiple sensors are commonly utilized simultaneously. It is important to accurately model and integrate these sensors within the simulator to mimic those employed in real vehicles. This part examines the various aspects of sensors that can be used within an autonomous vehicle simulator.

Default Sensors: The majority of simulators offer some default sensor models that are readily available for use. A simulator will receive a score of 1 for each sensors defined below if it offers support for it; otherwise, it will be assigned a score of 0.

a. RGB Camera
An RGB camera functions by translating the 3D environment within its field of view into a 3-channel 2D image (Red, Green, Blue). Refer to Figure 3.1 for an example of data captured by an RGB camera mounted on a vehicle [29]. These images are valuable for understanding the surroundings, including aspects such as traffic lights, signage, and pedestrian crossings.

Figure 3.1: Image of vehicles surroundings captured by a RGB camera

b. Depth Camera
A depth camera captures a single-channel image embedding depth information. Each pixel in this image denotes the distance between the camera's mounting point and objects in the 3D environment it references. Figure 3.2 showcases an example data captured by a Depth Camera fixed on a vehicle. This data is commonly utilized to measure distances between the ego vehicle and other nearby vehicles.

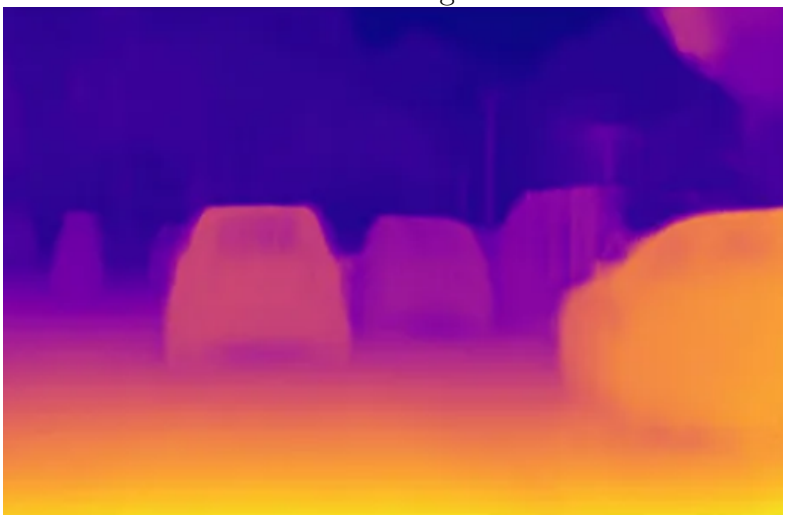


Figure 3.2: Depth Image of vehicles surroundings captured by a depth camera

c. 3D Lidar
Unlike traditional cameras, the 3D lidar creates a 3D point cloud representation of the environment. Each voxel's position corresponds to a point on surface of the objects in its surroundings. This sensor projects laser beams and calculates positions based on time of flight. Figure 3.3 [29] illustrates an view of a point cloud generated by 3D lidar. However, factors such as fog or object texture can distort the energy of the laser signals. These sensors are occasionally affixed to a rotary motor to achieve a 360-degree scan around the device.

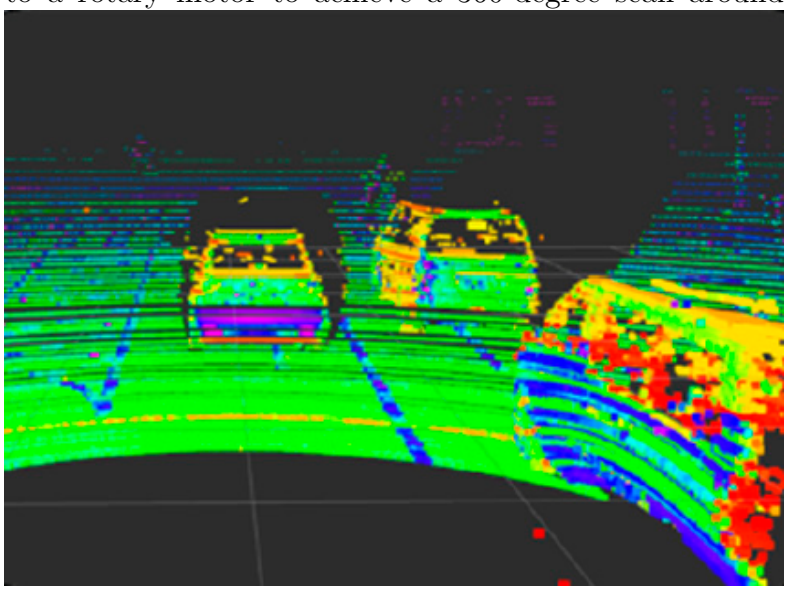


Figure 3.3: Point cloud projection of surrounding captured by 3D Lidar

d. Radar
Similar to the 3D lidar, the Radar sensor maps the environment into a 3D point cloud. However, unlike the 3D lidar, this sensor uses electromagnetic (EM) waves instead of laser beams. Radars takes the advantages of EM waves such as covering longer distances and experiencing less distortion from external factors. Both the 3D lidar and Radar are utilized to project the surrounding environment in 3D data.

e. IMU Sensor
The Inertial Measurement Unit (IMU) sensor measures specific forces like acceleration and angular velocity of a moving object. It provides information about the orientation and changes in position of the vehicle on which it is mounted on. IMU consists of sensing elements such as accelerometers and gyroscopes which aids in determining the vehicle's acceleration, change in direction, and rotation.

f. Semantic Information
Semantic Information isn't a physical sensor used directly in autonomous vehicles. Rather, it denotes a simulator's ability to offer semantic information about

every object within the environment. This information typically includes the class or name of objects, such as trees, dogs, children, women, along with the coordinates of bounding boxes encapsulating these objects. Refer to Figure 3.4 [29] for an illustration of the environment with bounding boxes and class labels of the objects. This information can be valuable in certain training algorithms like reinforcement learning, facilitating the provision of rewards and penalties.

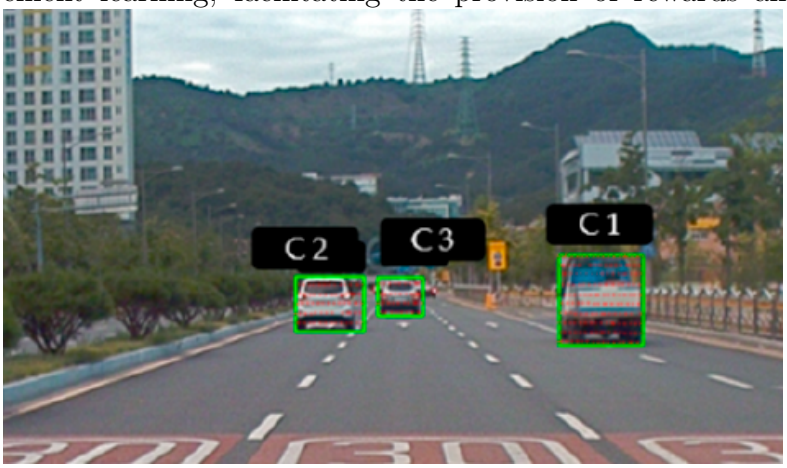


Figure 3.4: Semantic Information of cars in environment with bounding boxes

g. Force Impact sensor
This sensor measures the force experienced by a vehicle during impact with another object. This measurement directly correlates with the damage caused by both the vehicle and the object. Moreover, it can be utilized to assign weighted rewards and penalties.

h. Fuel/Battery Sensor
This sensor monitors information regarding fuel or power consumption by the vehicle. As self-driving vehicles aim for efficiency, researchers worldwide continually work to enhance fuel efficiency of the vehicle. Therefore, it's crucial for a simulator to offer such data.

i. Vehicle Speed Sensor
A widely used sensor present in nearly all vehicles, it measures the vehicle's velocity. This sensor is particularly important when navigating in areas with speed limits, such as city roads or warehouses.

j. GPS
The Global Positioning System (GPS) is a geolocation sensor utilized for locating and navigation. It's one of the most common sensors found in commercial vehicles, essential for path planning both on roads and within closed environments. Regarding simulators, it's vital for them to provide information about the coordinates of the vehicle and its destination within global boundaries with respect to the simulator scale.

k. Visibility Sensor
Visibility while driving can be affected by various environmental factors like fog, smog, or pollution. It is a measure of the maximum distance at which objects can be clearly seen which plays a critical role in safe driving. For simulators, providing this information becomes crucial, particularly when simulating foggy conditions.

l. Other Vehicle Sensors related to Vehicle Dynamics
Conventional cars possess numerous sensors measuring various car properties such as tire pressure, parking aid, engine RPM. However, not all these vehicle sensors are typically utilized in autonomous vehicle research. An ideal simulator should offer the infrastructure to model any of these vehicle sensors when needed, quantifying its ability to define and provide data related to ego vehicle.

m. Other Environment Sensors related to Perception
This parameter denotes the user's ability to define sensors measuring environmental properties. These user-defined sensors could offer data regarding the car's surroundings, like the number of people/vehicles within the car's field of perception. For example, sensors like audio recognition for identifying priority vehicles, such as ambulance and fire-fighting vehicles, fall under this category.

n. Macroscopic Global Sensors
Another user-defined sensor, this provides information about the simulation world, not just the local area near the vehicle. Such a sensor could furnish global details like traffic conditions in specific sections of the map or future weather forecasts, which can significantly aid in path planning.

Multiple Synchronous Sensors: In autonomous vehicles, multiple sensors are commonly used simultaneously to enhance reliability and provide backup in case of sensor failure. Simultaneous data from multiple sensors provide more reliable information compared to a single sensor. An AV simulator's capability to handle multiple synchronized sensor models and provide their data becomes crucial. This parameter ensures the synchronized data simulated by multiple sensors at a given time corresponds to the same time step and is coherent with each other.

User-Defined Sensor Position and Orientation: The placement and orientation of sensors within a vehicle significantly influence their application. AV Engineers meticulously design the mounting points and orientations, especially when multiple sensors are used to avoid sensor blind spots. This parameter evaluates a simulator's capacity of enabling users to define sensor location and orientation with respect to the vehicle geometry.

Precision of Sensor Readings: Sensor precision refers to the smallest measurable change in a sensor's readings. This attribute applies to various sensor types and their measurements. The precision of a sensor varies based on application needs and accuracy required. An adaptable simulator should accommodate diverse levels of precision, allowing users to configure it for various sensor according to their requirements.

Real-Time Sensor Models: Some vehicle manufacturers develop unique in-house sensors with unique features. Simulators incorporating digital models of these sensors help users to retain their specific functionalities within the simulation. Additionally, providing digital models of popular sensors readily available in the market becomes advantageous. This parameter evaluates a simulator's capability to import real-time digital models of sensors and integrate them into the simulation environment.

User-Defined Sensors and Observers: In the dynamic field of autonomous driving, new sensor technologies are frequently emerging, and existing sensors undergo performance updates to better understand the environment. In addition, users might have specific data of interest for their application or need to monitor certain aspects within a simulator. The simulator's capability to enable users to define observers to monitor data or model new sensors becomes important to cope up with evolving sensor needs and user-specific requirements.

3.1.2 Actors

Actors within the simulator constitute the diverse objects forming the environment. These actors broadly categorize into stationary and non-stationary, with stationary actors maintaining fixed positions (e.g., buildings, trees, traffic signs) and non-stationary actors changing their positions during simulation (e.g., people, animals, moving vehicles). This part explores criteria of various parameter related to actors that influences the comparison.

Geometry: Actors geometry pertains to their physical attributes, including appearance, spawning location and size.

a. Predefined Scenarios
   Environments for vehicle operation vary widely such as rural, urban, industrial... each possessing unique features and actors. For instance, observing a child is less likely in an industrial setting compared to seeing a cargo truck. This parameter

evaluates a simulator's capability to offer presets of environments with appropriate actors corresponding to various scenarios.

b. Predefined Models for Actors
This parameter evaluates the simulator's ability to offer access to meticulously designed digital assets (actors) across various categories such as buildings, vehicles, people, and traffic signs. These models are pre-configured across various scenarios and ready for deployment within the simulator environment.

c. User-Defined Actors
Apart from the preconfigured libraries of 3D models, the simulator should allow users to import custom 3D models of actors while maintaining a standardized scale within the simulator. This feature can aid user to tailor the simulator for their use case. For example, develop a simulator for driverless vehicle in golf field demands integration of 3D models of golf carts and players.

d. Number of Actors and Spawning Location
The nature of an environment is influenced by the number of actors within it. Users should have the ability to adjust the number of actors and their location to configure the environment. For example, configuring the traffic density in a particular road by defining number and location of vehicles or placing specific objects like traffic light poles.

e. Recreation of Real-Time Models
Simulators should possess real-time 3D models of actors such as cars, buildings while preserving their physical properties like geometry and textures. This feature appeals to users working with real-time vehicles or entities.

f. Path/Destination of Actors
Configurability of the moving trajectories for non-stationary actors, allows users to define paths and destinations. Actors should also respawn in new locations when they move beyond environment boundaries. This parameter ensures this capability of a simulator providing flexibility in defining their movement patterns.

g. Context-Aware Spawning
Ensuring that the spawning or respawning of actors aligns with contextual reasoning. For instance, spawning more ambulances near hospitals demonstrates intelligent spawning based on context. This parameter evaluates the simulator's ability to spawn actors sensibly in relation to their surroundings.

Behaviour: Actor behavior encompasses their characteristics, such as motion, response, behavior patterns, and sensitivity.

a. Predefined Rules
Simulators can define certain preset rules governing actor behaviour, such as traffic regulations in specific countries, readily available for use within the simulation.

b. User-Defined Rules
Allowing users to define specific behaviour patterns for actors within the simulator in addition to predefined rules, provides better customization options.

c. Distinguishable Behavioural Patterns
Simulators aiming to replicate real environments must simulate distinct behavioural patterns among actors. For example, distinguishing between the behaviour of an adult, a child, and a dog walking on a sidewalk. A simulator earns a score of 1 for this parameter if it is capable of accurately simulating these behavioural differences.

d. Sensitivity
Sensitivity refers to an actor's responsiveness or reaction to actions taken by other actors, including the ego vehicle within the environment. A score of 0.5 is given if actors respond solely to the ego vehicle, while a score of 1 is granted if actors respond to all fellow actors. This parameter evaluates the simulator's sensitivity to fellow actors' actions.

e. Level of Aggressiveness in Driving
The driving pattern varies among various parts of the world. Simulators should allow users to define driving patterns' aggressiveness using metrics like violations per kilometer driven or similar parameters. This parameter enables users to train and test in various driving conditions by defining custom behavioural patterns.

f. Reproducibility
Certain random behaviours exhibited by actors can elicit unique responses from the ego vehicle. The simulator should enable users to reproduce the same set of random actions performed by actors, known as seeding. This parameter evaluates the simulator's capability to reproduce a specific set of actions performed by actors for further analysis.

3.1.3 Environment

The simulator's environment includes the weather, terrains, maps, and conditions where the vehicle operates. This part explores various the criteria for various comparative parameters related to environment.

Pre-defined Libraries: This parameter assesses the simulator's capability to feature pre-designed environments consisting of diverse landscape and weather conditions. The simulator earns a score of 0.5 if it satisfies one condition and a score of 1 if it fulfils both.

User-Defined Environment: Simulator tools enabling users to define their own maps and various weather conditions enhance customization and adaptability, allowing for efficient simulation tailored to specific needs.

Real World Maps: The capability to import or utilize real-world maps within the simulator holds significant importance, particularly for developing commercial autonomous vehicles that directly navigate on real road. A simulator scores 1 if it facilitates user access to import or integrate real-world maps and associated features.

Photorealism: Photorealism is the concept of how closely a simulator's environment resembles the real world in appearance. Assessing photorealism is subjective, thus scored by users through visual inspection on a scale of 0 to 1. A simulator that closely replicates reality receives a higher score, while deviations affect the score inversely.

Variable Resolution: The ability to vary the level of detail in the rendered environment by adjusting resolution. Variable resolution provides flexibility, allowing users to balance computational resources against information accuracy. With lower resolution considerable computation resources can be saved but potentially losing detail and vice versa. This parameter checks the ability of a simulator which enables the user to vary the resolution.

Context-Aware Map Planning: The simulator-generated map should possess contextual coherence, ensuring that elements such as buildings, roads, and traffic lights correlate logically within the environment. This feature scores 1 when a simulator has the capability of generating contextually plausible maps.

User-Defined Scenarios: A versatile simulator should allow users to generate specific scenarios, such as road constructions or obstacles like fallen trees on roads. This parameter is scored by its ability to facilitate user scenario generation.

Material Definition: The optical properties of environment materials can influence sensor data, impacting sensors like Lidar and Cameras. For instance, Lidar may fail to detect glass walls due to the laser beam passing through it rather than reflecting. This parameter evaluates the simulator's capability to enable users to define materials and texture of objects in the environment.

Variable Level of Sun Shades: Shades of the sun can influence significantly in replicating the time of day and weather ultimately impacting the sensor readings. This parameter refers to the simulator's ability to allow users to manipulate sun shades, considering factors like time of day and weather.

Special Optical Features: Real-world optical phenomena like sun glare, mirage, dust, and fog can impact driver visibility and sensor data for driverless vehicles. Therefore, it's also important for simulators to model these features which impacts sensor data and subsequent vehicle actions.

Data Visualization and Analytical Tools: Simulator tools facilitating the display of various data and analyses during training and testing offer valuable insights to users. This parameter assesses whether the simulator allows users to define data visualization and monitoring tools, which could include sensory data from the vehicle, algorithm-related data, or annotations like bounding boxes and semantic truth.

Rendering: During testing, the quality of scene rendering aids in analysing algorithm performance. Users should have the ability to modify rendering parameters such as view angle and frames per second, ensuring adaptable display quality. This parameter evaluates the simulator's capability to offer configurable rendering options, earning a score of 1 if available.

Recording: Recording the training process allows users to analyse performance and playback sessions for future evaluation. This parameter accounts for the simulator's ability to record the training process alongside associated data, aiding users in comprehensive analysis.

3.1.4 Ego Vehicle

The agent vehicle, or ego vehicle, is the vehicle which is subjected to training or testing for autonomous driving within the simulator. This part explores various parameters evaluating the simulator based on the ego vehicle.

Vehicle Geometry: Vehicle Geometry refers to the dimensions and bounding box encapsulating the vehicle in the 3D environment. This parameter evaluates the simulator's capability to allow users to configure the geometry of the ego vehicle.

Wheel Geometry: This parameter involves the ability to modify suspension and steering system parameters of the ego vehicle within the simulator. Ensuring adherence to physics related to wheel geometry is important as it impacts driving performance significantly.

Real-World Models: This parameter assesses the simulator's feature of enabling the user to import and use of 3D models of real-world vehicles developed by vehicle manufacturers. This feature is important and time saving for researchers from vehicle manufacturing companies.

Real-World Physics: This parameter is a check for the simulator's capacity to simulate various physical parameters associated with driving, including wheel slip, crash damage, and vehicle inertial behaviour. Accurate replication of these features is vital for developing algorithms for real-world on-road autonomous vehicles.

Input Commands: The driving algorithm generates action commands for the vehicle (e.g., acceleration, steering angle) based on observations. Configurability of input commands for every component within the ego vehicle is crucial to accommodate various algorithms. This parameter ensures the simulator's ability to configure input commands for the vehicle.

Other Components: Minor components like headlights and horns can have minor impact in driving. The simulator should allow users to model the intensity or impact of these components. This parameter evaluates the simulator's feature for adjusting the effects of such components.

3.1.5 Platform

The framework within which the simulator operates plays a crucial role in its versatility and usability. This part evaluates various comparative parameter related to simulator's framework.

Cross-Platform Compatibility: This parameter assesses the compatibility of the simulator software across diverse operating systems, determining support for OS environments such as Linux, ROS, Windows, MacOS, and any other relevant OS. The simulator earns a score of 1 for each supported operating system from the listed ones.

Open Source: Evaluating whether the simulator or parts of it are open source, enabling users to tailor the simulator to their specific needs by importing components as plugins and developing customized software. A score of 1 is given if the simulator allows for such user-driven customizations.

Scripting Language Support: This parameter evaluates the breadth of scripting language support within the simulator, crucial for facilitating communication between users and the simulator. The assessment includes popular languages like Python, C++, C, R, Java, and other relevant scripting languages. The simulator earns a score of 1 for each supported scripting language from the listed options.

Scalability: This parameter evaluates the simulator's ability to provide various light versions to suit diverse user needs. Offering various individual light versions with unique features, alongside a comprehensive pro version encompassing all features, the simulator ensures adaptability to different user requirements. This feature contributes to making the software more streamlined and efficient for users by allowing them to opt for lighter versions catering to their specific needs.

Computational Resource Evaluation: In this study, the computational resource consumed by a software is calculated based on electrical energy consumption by a baseline system under maximum operation. The configuration of the baseline system and its power consumption at it maximum usage is tabulated in Table 3.1

The score is calculated by comparing the simulator's power consumption during operation against the maximum power consumption of the baseline system (approximately

Level	Name	Description
Processor	Intel i9 processor	200–250 Watts
GPU	Nvidia RTX 3090 – 24 GB	400 Watts
RAM and ROM	16 GB and 500 GB SSD	15 Watts
Monitor	27" 4K resolution Monitor	50 Watts
Fan and other components		25 Watts

Table 3.1: Baseline system configuration

750 watts). The score for the computation resource consumed by a simulator A is given by

$$\text{Score} = 1 - \left(\frac{\text{power consumed by simulator}}{\text{maximum power consumption by baseline system (750 W)}} \right)$$

Community Support: Community support is crucial for public software, evaluating three streams: structured documentation, tutorials, and discussion forums. Each stream receives a score of 1 if available.

Distributed Computing: This parameter assesses the simulator's ability to execute multiple simulation instances on different machines, converging into a coordinated and coherent simulation system. In addition, training of Machine learning can be parallelized when a simulator supports GPU operations. This parallelization capability enhances efficiency by reducing runtime and increasing the number of simulation episodes within a given time.

3.1.6 Driving Algorithm

This section delves into the comparative assessment of the driving algorithm used to navigate the ego vehicle within the simulator, analysing various parameters critical to algorithm and associated data.

Data Exchange: The interaction between the algorithm and the simulation engine often occurs through communication channels like APIs or dedicated communication bridges. This parameter evaluates the presence of pre-defined communication plugins, scoring 1 if available.

Data Logging: Data logging involves collecting and storing simulation-generated data for future analysis. This parameter assesses the simulator's support for user-controlled data logging, scoring 1 if it aids users in logging specific data of interest.

Hardware in Loop (HIL): The incorporation of real-time hardware, such as sensors and vehicle controls, within the simulator aids in testing hardware performance. This parameter evaluates simulator support for any hardware involvement during operation.

ML Libraries: Support for popular machine learning libraries like TensorFlow, PyTorch, and Scikit simplifies machine learning model training. This parameter assesses the simulator's compatibility with integrating such ML libraries, enabling seamless utilization of these tools within the simulator environment

3.2 Methodology

This section outlines the process of deriving a single score for a simulator using the above-mentioned parameters. The scoring system comprises two main components which constitute to form the final scores: Base Score and User Weight

3.2.1 Base Score

Each simulator being compared will receive scores for all defined parameters on a scale from 0 to 1 and the scores are evaluating only the simulators's capability. This produces a base score vector for each simulator, usually provided by the developers of the simulation software.

3.2.2 User Weight

Users assign weights ranging from 0 to 1 to each defined parameter based on the individual importance. These user-assigned scores create a user weight vector that reflects the significance of each comparative parameter to the user and their application.

3.2.3 Final Score

The final score, denoting the suitability of a set of n simulators $S = (S1, S2, S3 \dots Sn)$ for a specific user (U), is calculated using the dot product of the base score vectors of simulators (S) and the user weight vector of user U. This computation generates a usability score for each simulator, allowing users to assess and compare simulators based on their preferences.

4 Generative Model Based Simulators

4.1 Neural Networks

Neural networks are mathematical functions that establish a mapping between input and output values. Comprising multiple perceptron organized in interconnected layers, these function architectures form the backbone of complex computation.

4.1.1 Perceptron

A Perceptron stands as a linear binary classifier expressed through the equation [34] :

$$Z = W \cdot X + B$$

Here, W denotes the weight vector $w_1, w_2, ...w_n$, X signifies the input vector $x_1, x_2, ...x_n$ and B represents the bias. The weights within W symbolize the significance of each input in vector X toward the prediction. Refer to Figure 4.1 for an illustration of a neuron in a perceptron with a single output and multiple inputs.

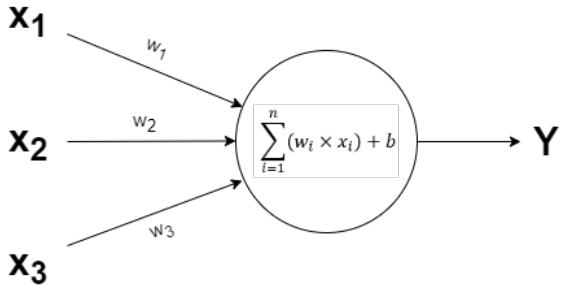


Figure 4.1: Perceptron

4.1.2 Multi-layer Perceptron

Neural networks are constructed by interconnecting multiple layers of neurons [9], integrating non-linear activation functions such as sigmoid and ReLU between these layers (Figure 4.2) [24]. The process of computing the output from this composite function for a given input is termed Forward Propagation. Neural networks, as universal ap-

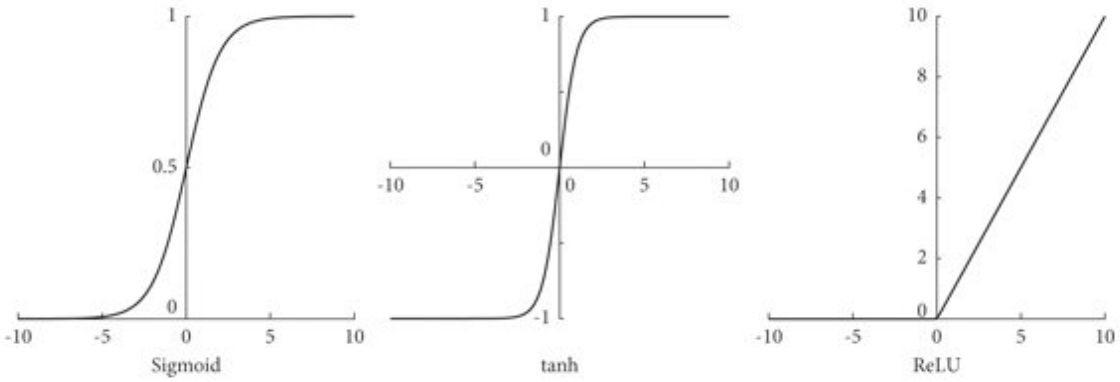


Figure 4.2: Non-linear activation functions - a) Sigmoid b) tanh c) ReLU

proximators which efficiently approximate nearly any continuous function [20]. The incorporation of multiple layers of neurons and activation functions enables them with the ability to approximate intricate functions. The weights and biases associated with the neurons initiated randomly and learned during training. Figure 4.3 illustrates a two-layered network). Mathematically, the network function with input X is represented as

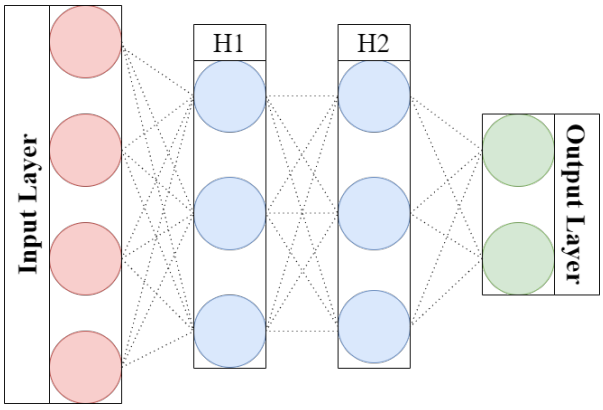


Figure 4.3: Fully connected network with 2 hidden layer

a probabilistic function, with the output Y expressed as $p(y|x)$.

4.1.3 Loss Functions

Loss functions, also termed cost functions, evaluate the deviation between the original and predicted data points. These metrics guide network training towards minimizing this deviation. Popular loss functions include Mean Square Error [38], Entropy loss [8], and KL divergence [28]. The loss functions are selected based on the data and specific use case.

4.1.4 Optimizers

Optimizers represent the algorithm used in neural network training. They facilitate learning the network's parameters (weights and biases) to minimize the network's cost function. Usually, this is achieved by computing parameter gradients with respect to the loss function and updating these parameters by a step size in a manner that shifts their gradients towards the global minimum:

$$P_{\text{new}} = P_{\text{old}} - \alpha \nabla L(P_{\text{old}})$$

Here, $\nabla L(P_{\text{old}})$ denotes the gradient of the loss function with respect to P_{old}. Typically, in multi-layered networks, parameter gradients are calculated using the chain rule. The learning rate (α) signifies the speed of parameter updates, often diminished during training, to prevent skipping the minimum. This iterative parameter update process called as Backpropagation. Various optimization techniques such as RMSprop, ADAM [26], Adagrad [12], and Adadelta [59] are used in neural network training.

4.1.5 Training

The training process includes iteratively backpropagating through the network until the loss is minimized and stabilizes. Segmentation of the training data into batches enhances computational efficiency and expedites training. The backpropagation across all batches in the training data constitutes an epoch. Metrics like accuracy, monitor the training progress. Furthermore, a small set of training data which are not utilized for training, is employed to validate the model during the training.

4.1.6 Batch Normalization

Batch normalization is a technique in neural networks, optimizing the training process by normalizing layer inputs within mini-batches [21]. Computation of batch-wise mean and variance enables normalization of inputs, resulting in faster convergence, mitigating vanishing or exploding gradients, and reducing sensitivity to weight initialization.

4.2 Generative AI

Generative AI represents a domain uses artificial intelligence concepts to generate novel data instances. Initially, Various machine learning methods such as deep learning are used for classification and prediction. However, over the years these models have evolved to create new data instances. Trained to learn the density of the distributions of training data, these models include various architectures tailored to specific data. For instance, Convolutional Neural Networks performs well in learning patterns and features within images, while Recurrent Neural Networks effectively learn temporal relationships in data sequences.

Once the model learned the distribution of the training data, these models possess the capability to generate new data points by sampling from the learned distribution. These generated data points typically diverge from the training corpus, yet effectively trained generative models exhibit the ability to generate data resembling the features found in the training data. This versatile methodology spans diverse fields, including image, music, and text generation. Generative modelling, based on learning the density of data distribution, categorized into two categories: Implicit and Explicit Density Models.

Explicit Density Models : Explicit Density Models explicitly learn and estimate the probability distribution of input data. Variational Autoencoders (VAEs) [37] and traditional probabilistic models is a typical example of this approach by explicitly modelling the probability distribution within a defined space. VAEs acquire a probabilistic representation of the data, enabling the generation of new samples through sampling from this learned distribution.

Implicit Density Models : In contrast, Implicit Density Models do not explicitly define the probability distribution. Instead, they focus on generating new data points without directly modelling the probability distribution. Generative Adversarial Networks (GANs) fall in this category. GANs employ a generator network tasked with creating

samples without explicitly defining the underlying probability distribution. The generator's objective is to generate data indistinguishable from real data, without directly modelling of the probability distribution.

The introduction of Generative Adversarial Networks in 2014 by Goodfellow et al [14]. marked a pivotal breakthrough in Generative AI. Initially applied for image generation, GANs stood out for producing high-quality images in contrast to the blurry images generated by VAEs. Early challenges such as Mode failure and training instability were addressed in subsequent GAN variants, solidifying its prominence in the domain. These models exhibit versatility across various data types such as videos, audios, signals...

The capacity to generate diverse and high-quality data finds various applications. In a proposed approach, these generated data drive a simulator used for training autonomous vehicles. Effective training of an Autonomous vehicle demands exposure to diverse scenarios, which existing simulators might limit. In this proposed methodology, a neural network model drives the simulator, generating requisite sensory data for autonomous vehicle training. This approach facilitates training the driving model on diverse scenarios, augmenting the vehicle's ability to generalize across various environments.

4.3 Training data

The training of a machine learning model in supervised setup, particularly in the domain of generative deep learning, heavily relies on data. The model's performance scales proportionally with the volume of training data on which it is trained on.

In this research, the synthesis of sensory data of autonomous vehicles alongside corresponding action labels proves challenging to obtain in substantial quantities from existing open-source datasets. While datasets like [47], [13], [7] and [57] provide comprehensive sensory data at each time step, they lack the corresponding action labels.

Addressing this challenge, this research synthesizes data derived from a simulator, incorporating action labels. The generated data is rooted in the mathematical model of a 2D Lidar sensor installed on a vehicle navigating within a confined environment. Each instance in the dataset comprises (Observation at time step t, action label, Observation at time step $t+1$). A dataset consisting of 150,000 such data points is synthesized and used to train the generative model. The anticipated outcome is the generation of new data points closely resembling those produced within the simulator environment.

4.3.1 Mathematical Model-based Simulator

The simulator is constructed using Python, featuring an environment mapped to the floor plan of our university building. The walls in the floor plan are represented as lines within the simulator, defined by their endpoint coordinates. A virtual vehicle navigating this environment is represented by a rectangular box scaled in accordance with the global environment boundaries. Simulated changes in the vehicle's position and orientation are executed by updating the vehicle boundary coordinates based on action commands.

The action command is formulated as a three-dimensional vector which includes the vehicle's velocity, turn angle, and turn direction. Vehicle velocity ranges between -5 to 5 units, while turn angle confined within 60 degrees on either side. The simulated 2D Lidar, mimicking observations from the vehicle, comprises a 360-sized vector. Each vector entry signifies the distance between the sensor and the closest obstacle around the vehicle for a given angle (360 degrees). This distance calculation entails projecting lines at each angle and checking for intersection with the environment's walls. The Cramer rule facilitates identifying intersection points between angle lines and wall lines, enabling the determination of closest distances of the obstacles from the vehicle. This iterative process across 360 degrees yields the observation at given timestep. Refer to Figure 4.4 illustrating the vehicle, its projection lines, and the walls within the simulator.

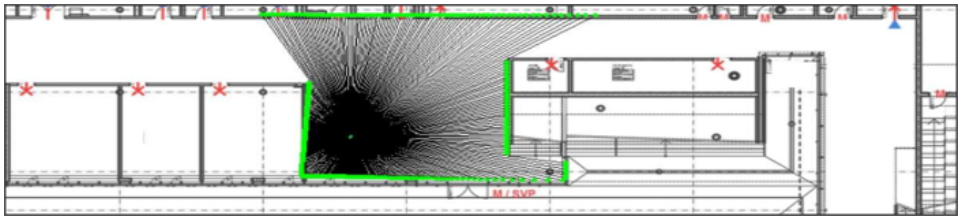


Figure 4.4: Observation of the environment within the simulator

Successive sensor data calculated across two time steps are stored as a tuple alongside the corresponding action command, constituting each data point: (O_t, A_t, O_{t+1}) Figure 4.5 visualizes the data variation across two consecutive time steps. Gaussian noise is injected in the data to make it robust, and the vehicle initializes at a new location upon collision or crossing walls. Randomly logged data points across time steps are shuffled, culminating in the logging of 150,000 data points for further training the generative model.

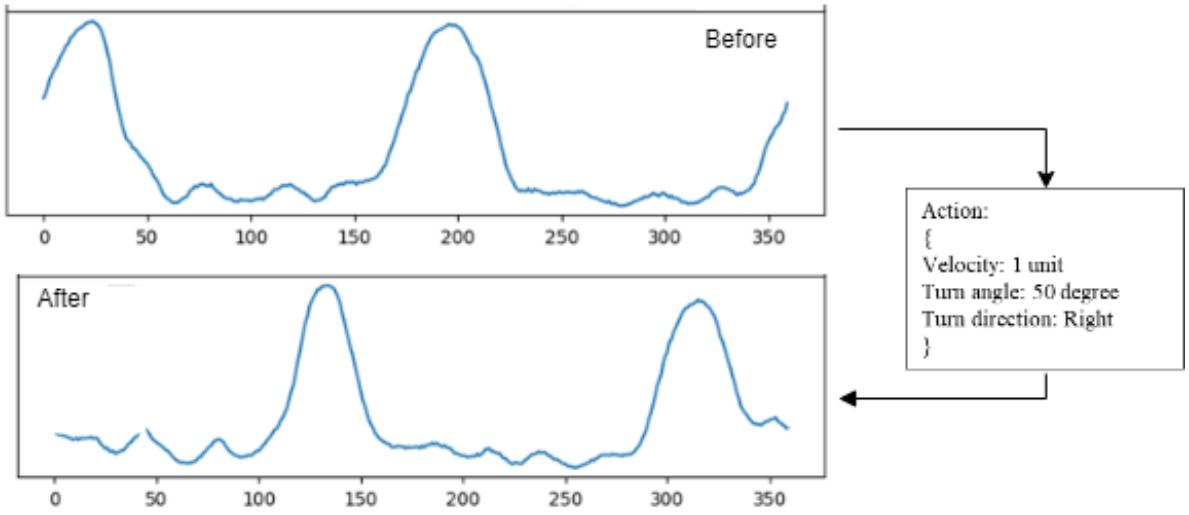


Figure 4.5: Observations at two consecutive timesteps

4.4 Conditional Generative Adversarial Networks

In Generative Adversarial Networks (GANs), the typical setup involves two models: the Generator G and the Discriminator D. Traditionally, the generator G produces new instances y based on noise z sampled from a random distribution, and the generated data is represented as $Y = G(z)$. However, this approach lacks control over the content of the generated data.

In this work, a conditional GAN architecture is employed, altering the generator's output to be conditioned on specific inputs—namely, the observation at the previous time step O_t and the action label A_t. The Generator, in this context, generates the observation for the next time step based on the given previous timestep and action: $O_{t+1} = G(O_t, A_t)$.

Conversely, the discriminator evaluates the conditional probability of the observation at the next time step O_{t+1} being real and plausible for given O_t and A_t:

$$D(O_t, O_{t+1}, A_t) = P(O_{t+1}|O_t, A_t)$$

This conditional setup allows for controlled generation of data, where the generator's output is influenced by inputs, leading to more tailored and context-aware generation.

4.4.1 Generator

The generator adopts an encoder-decoder architecture. Taking the previous timestep's sensor observation O_t and action label A_t as input, the encoder encodes the observation data into a 32-dimensional latent code. Simultaneously, the action label is passed through a Dense layer, and its output is concatenated with every layer of the encoder (refer to Figure 4.6). Additionally, a random 32-dimensional noise sampled from a normal distribution is passed through a linear layer and concatenated with the latent code, introducing noise for the generation of new objects within the output.

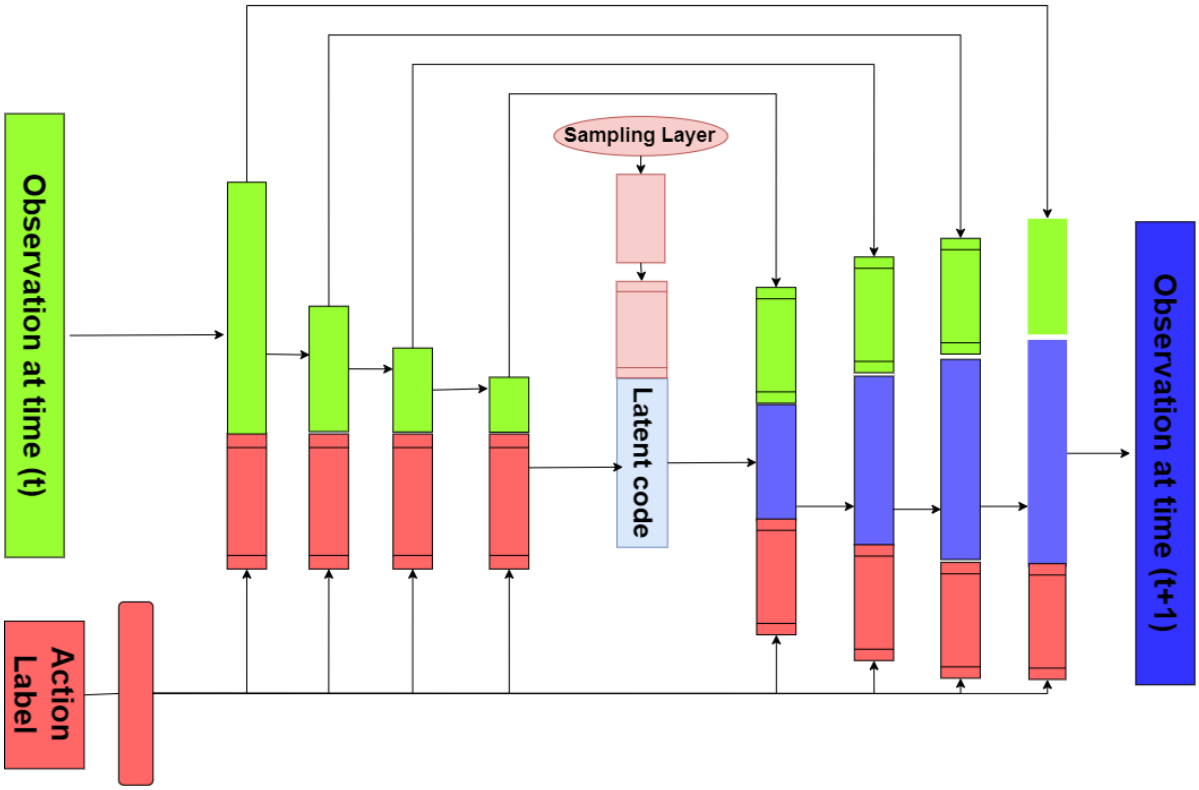


Figure 4.6: Generator Architecture

The decoder accepts the latent code and random noise as input, producing the observation for the next timestep O_{gt+1} as output. Comprising four fully connected layers, each layer is concatenated with the action label and the output of the corresponding encoder layer (see Figure 4.6). By implementing these skip connections, information from the inputs is preserved after passing through the bottleneck. ReLU activation functions and batch normalization follow each layer in both the encoder and decoder.

4.4.2 Discriminator

The discriminator is a fully connected network processing three inputs: observation at timestep t O_t, observation at timestep $t+1$ O_{t+1}, and the action label A_t,. Both observations propagate forward through the network, while the action label undergoes pass via a dense layer and is concatenated with every layer in the network. The output layer consists of a single neuron with a sigmoid activation function (Figure 4.7). This network outputs the probability that the given observation pair concerning the action label is real. Furthermore, the network evaluates the authenticity of generated data.

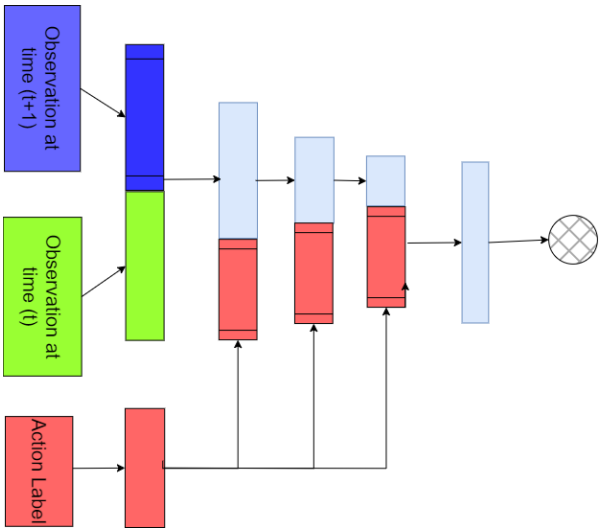


Figure 4.7: Discriminator Architecture

4.4.3 Training Loop

The GAN's training occurs in two distinct steps: the individual training of the Generator and the Discriminator. A batch of data is fetched from the training corpus. The observation at the previous time step O_t and action label A_t are propagated through the generator, resulting in the generated observation for the next timestep O_{gt+1}. This output, combined with generator inputs (O_t, A_t), is then fed into the discriminator. The discriminator evaluates the probability of the authenticity of O_{gt+1} for given O_t and A_t. Generated observations are assigned a 'fake' label ('0'), prompting computation of cross-entropy loss between these labels and the discriminator output. Additionally, mean square error is calculated between the generated observation O_{gt+1} and the actual observation O_{t+1}. The generator gradients are computed with respect to the combined weighted loss, as expressed in the equation, and the generator gradients are updated:

$$Generator\ loss, L_g = L_d + \alpha \times MSE$$

where α is the weight corresponds to mean square error. For the discriminator, actual data (O_t, O_{t+1}, A_t) and generated data $(O_t, O_g t + 1, A_t)$ are utilized, compared against the 'real' label ('1') and 'fake' label ('0') respectively. The sum of discriminator loss from both generated and actual data is calculated. Subsequently, the discriminator gradients are computed and updated concerning this combined discriminator loss: -

$$Discriminator\ Loss = L_{dreal} + L_{dgenerated}$$

During generator training, discriminator weights are frozen, and vice versa, preventing either component from becoming overpowering. This alternating training of the generator and discriminator occurs for each epoch, facilitated by the use of the ADAM optimizer at adaptive learning rate

5 Results and Discussions

5.1 Comparative metrics study

The methodology proposed for assessing simulator suitability necessitates inputs from both simulator developers and users to validate the applicability of the derived scores. While the base score of simulators for proposed parameters can mostly be populated from official documentation, subjective parameters like "Photorealism" require alternative evaluation methods. To assess such subjective parameters, a collective perspective was gathered from various individuals. This included an average evaluation from multiple sources to ensure a well-rounded perspective. To assess user weights, four distinct user personas were identified, each with unique use cases for simulator:

User 1: An University student primarily focused on sensory data, feedback, and training deep learning models for projects. While realism holds importance, it's not a primary criterion.

User 2: A software tester at an automobile company dedicated to real-world model performance validation.

User 3: A research scholar dedicated to achieving autonomous driving within a confined environment, endeavoring to recreate customized real-world scenarios within the simulator.

User 4: Emphasizes extensive data generation, resembling a substantial data collector assembling datasets for various purposes, extending beyond autonomous driving or machine learning tasks.

Three simulators CARLA [11], Summit [5], and LGVSL [39] were considered for evaluation, with their base scores sourced from official documentation (7).

The derived user weights for the identified personas (found in the appendix) were utilized to calculate final scores, summarized in Table 5.1:

	CARLA	Summit	LGVSL
User 1	18.5	**21**	19.7
User 2	22.5	34	**36**
User 3	16.65	**23.4**	22.4
User 4	18.4	29.2	**29.7**

Table 5.1: Final scores of comparisons

Interpreting the results, LGVSL emerges as a suitable choice for User 2, emphasizing the integration of real-world components, an essential criterion for their use case. Conversely, Summit resonates with User 1 due to its adeptness in simulating realistic traffic behaviours and leveraging CARLA's platform for training ML algorithms. User 3 requires more customization options. SUMMIT simulator, a derived version, includes CARLA's features along with its own proven enhancements, making it the preferred choice. On the other hand, LGVSL offers various data related to ego vehicle data and enables efficient logging, which appeals to User 4.

A critical suggestion for refinement involves the introduction of a "mandatory parameter" criterion. This criterion would empower users to define indispensable parameters, and if a simulator under evaluation fails to offer them, it should be excluded from the assessment. This approach ensures that users can establish essential criteria tailored to their specific needs. This comparative analysis showcases the varied suitability of simulators based on distinct user perspectives, emphasizing the importance of tailored assessments for specific use cases.

5.2 Generative Model-Based Simulators

The proposed architecture underwent training for 50 epochs utilizing a mini-batch size of 16. Figure 5.1 illustrates the dynamic trends in generator and discriminator loss during the training phase.

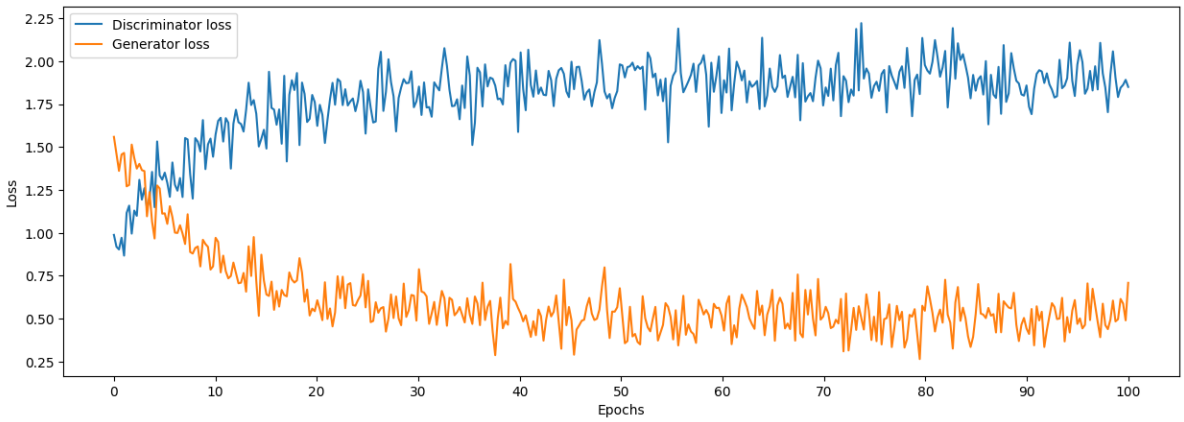


Figure 5.1: Generator and Discriminator Loss

To optimize learning, the mean square loss weight parameter alpha was systematically reduced by half every 10 epochs. This strategy allowed the generator to initially focus on data reconstruction, progressively advancing to generate novel features.

Performance evaluation of the model involved testing with various customized action commands. Figure 5.2 portrays the generation of subsequent observation $O_{g(t+1)}$ based on randomly chosen action vectors at a preceding time step (O_t).

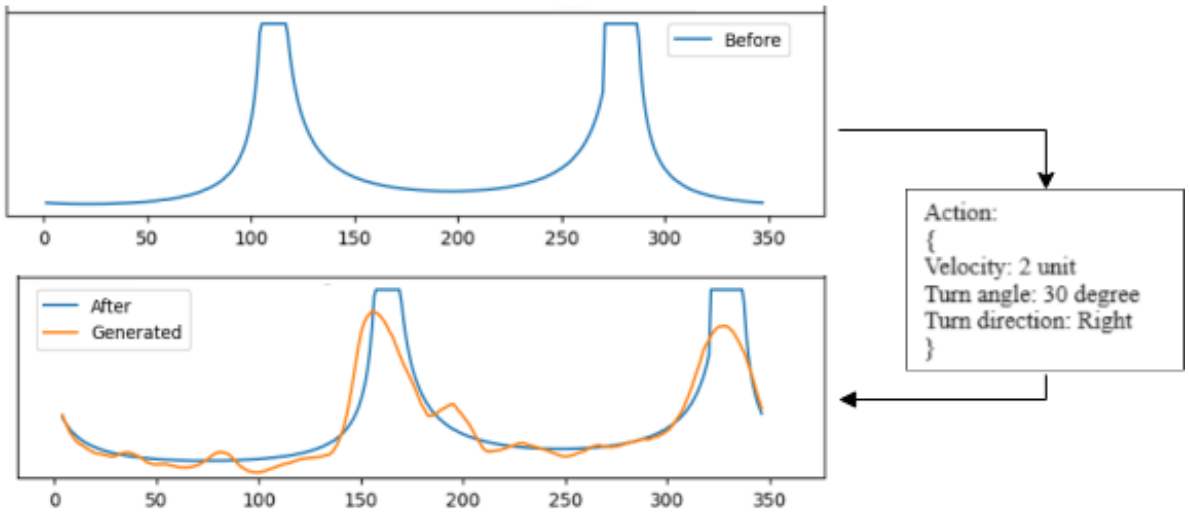


Figure 5.2: Transition of Observation for the Action Command *Turn angle: 30 degrees, Turn direction: Right, Velocity: 2 units*

Evident shifts within the signal patterns validate the successful transition of observations in concordance with the designated action commands. Figure 5.2 provides a comparative analysis between the generated output and the expected output for identical action commands. While the generated output exhibited discernible noise and lacked the pristine

quality of target output from the training corpus, no instances of new object generation were identified within the input. The failure to generate new objects can be attributed to the limited diversity of objects within the training dataset. Furthermore, visually discerning the generation of plausible new objects within 2D lidar data posed significant challenges. Recursive generation of outputs, wherein the generated output $O_{g(t+1)}$ serves as input for subsequent time steps, led to a cumulative increase in noise. Subsequently, this cumulative noise adversely impacted the fidelity of the environment's representation over time, as depicted in Figure 5.3, displaying observations generated recursively for 20 time steps.

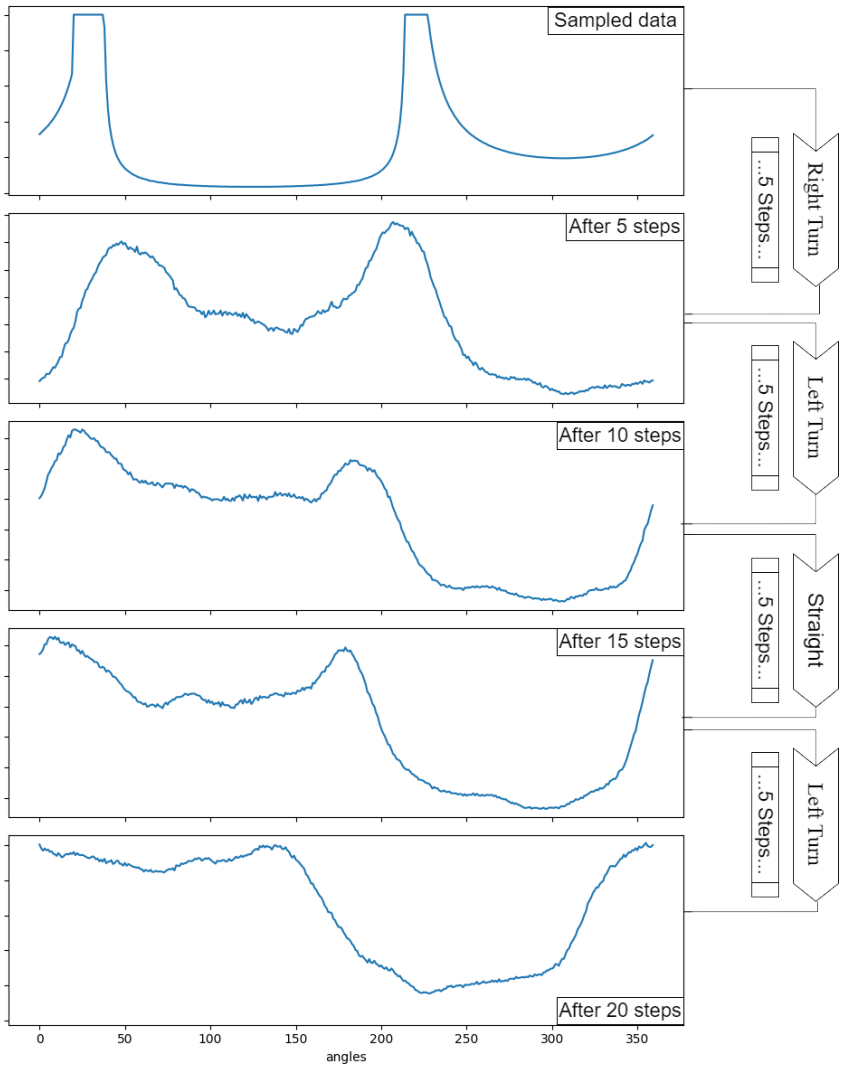


Figure 5.3: Recursive Generation of Sensor Observations

Initial outputs exhibited commendable quality; however, the quality notably degraded after 5 steps.

6 Conclusion

This research includes two primary phases. The first segment aimed to establish a unified metric for evaluating the useability of diverse autonomous vehicle simulators. The objective was to quantify a simulator's usability based on user-specific requirements. An extensive study of available simulators and their respective use cases led to the identification of 73 distinct parameters for comparing these platforms. A scoring system was devised, allowing each parameter to be rated on a scale of 0 to 1.

The scoring system comprises two sets of scores: one assesses the simulator's capabilities (referred to as the Base score), while the other measures the significance of each parameter for a user and their intended application (referred to as user weight). The final score results from the combination of these two sets, offering a comprehensive representation of both simulator capability and user preferences. These final scores, generated for different evaluated simulators, serve as a basis for informed decision-making. A higher score signifies a better fit for the user's needs, facilitating a systematic comparison among various autonomous driving simulators. This approach streamlines the process of evaluating simulators, significantly reducing the time users spend navigating through documentation, installing simulators, and assessing their applicability. Moreover, it enables the creation of a precise overview of a simulator's functionalities.

The second segment of the thesis concentrated on developing a prototype simulator driven by a deep generative model. This approach aimed to replicate two fundamental functionalities of an autonomous driving simulator. Firstly, it focused on simulating the transition of sensory data to a subsequent state based on the executed action. Secondly, it aimed at generating a spectrum of novel scenarios crucial for training machine learning models to ensure their ability to generalize effectively.

The generative model's output was proposed to drive the simulator iteratively by using its own generated output as subsequent input in consecutive time steps. The scenario generation holds potential in aiding algorithms trained on this simulator to generalize more effectively, potentially addressing the issue of limited environmental diversity observed in certain simulators like Carla [11].

A conditional generative adversarial network was devised and trained using data obtained from a mathematical model-based simulator. Specifically, the network was trained on sensor data derived from a simulated 2D Lidar. The objective was to learn the tran-

sition of sensor data across multiple time steps in accordance with an action command represented by a 3-dimensional vector (Turn angle, Turn direction, Velocity). Furthermore, the trained model was intended to generate instances of such data representing plausible new scenarios.

The results indicated that the model effectively learned the physics underlying sensor and effectively modelled their data transitions in response to input action commands. However, limitations surfaced as the generated data did not introduce new objects and their representations in the 2D Lidar map. This shortcoming might due to the lack of diversity in the training data. Additionally, noise was observed in the generated data, presenting challenges in identification and assessing the realism of newly generated objects within the 2D Lidar signal.

7 Future works

Future works regarding the comparative metric includes the potential introduction of a system that mandates parameters which are utmost crucial to users. Such mandatory parameters would signify critical criteria, and simulators failing to meet these standards could be excluded from evaluation rather than receiving higher weights. Additionally, considering that base scores remain constant for a specific simulator version, pre-establishing a list of base scores for widely used simulators would streamline the application of this evaluation system, ensuring user-friendly implementation. Moreover, the inclusion of new parameters derived from user feedback and reviews could enrich the evaluation framework, enhancing its relevance and comprehensiveness over time.

Regarding sensory data generation, there are several areas that can be improved. The methodology employed for learning the transition of 2D Lidar sensor data concerning given action commands could be extended to encompass other sensors prevalent in autonomous driving systems. Specifically, this approach could be extrapolated to generate sequential images based on prior image time steps and their respective actions. Accumulating diverse datasets across various scenarios could facilitate learning its distributions and the generation of diverse scenarios with novel objects integrated into the generated images. Evaluating and quantifying the realism of generated output is comparatively more straightforward in images than in Lidar data. Moreover, integrating specific filters such as low pass filters could diminish noise in the generated output (2D Lidar signals), preventing its accumulation across successive time steps. Training the model using datasets containing synchronized data from multiple sensors holds potential to enable the model to learn the correlations among sensors concerning their variations in response to actions, ultimately leading to the generation of synchronized multisensory data.

List of Figures

List of Tables

Appendix

The table 1 shows the scores of CARLA[6], SUMMIT [46], and LGVSL[30] simulators for each criterion in the proposed system for comparison. These scores are calculated based on the information from the official websites and documentation of the software.

			CARLA	SUMMIT	LGVSL
1. Sensors					
	Ready to use Sensor				
	i	RGB Camera	1	1	1
	ii	Depth Camera	1	1	0
	iii	3D Lidar	0	0	1
	iv	Radar	1	1	1
	v	IMU Sensor	1	1	1
	vi	Semantic info Sensor	1	1	1
	vii	Force Impact Sensor	0	1	1
a	viii	Lane detection/ infringement sensor	1	1	0
	ix	Fuel/ Battery sensor	0	0	0
	x	Vehicle speed sensor	0	1	1
	xi	GPS	1	0	1
	xii	Visiblity Sensor	0	0	0
	xiii	Other vehicle sensors related to vehicle dynamics	0	0	0
	xiv	Other environment sensors related to perception	1	0	1
	Xv	Macroscopic Global Sensor	0	0	0
b	Multiple Synchronous Sensors		1	1	1
c	User defined position and orientation of sensors		1	1	1
d	Precision of the Sensor readings		0	0	0
e	Availability of real time sensor models		0	0	1
f	User defined Sensors and Observers		0	0	1
2. Actors					
	Geometry				
	i	Pre defines Scenarios	1	1	1

a

			CARLA	SUMMIT	LGVSL
	ii	Predefines models for Actors	1	1	1
	iii	User-Defined Actors	0	0	0
	iv	No. of Actors and Spawning Location	1	0	0
	v	Recreation of Real time models	0	0	1
	vi	Path/ destination of Actors	0	1	0
	vii	Context aware spawning	0	0	0
	Behaviour				
	i	Pre defined Rules	1	1	1
	ii	User defined Rules	0	1	1
b	iii	Distinguishable Behavioural pattern	0	1	0
	iv	Sensitivity	1	1	1
	v	Level of aggressiveness in driving	0	0	0
	vi	Reproducibility	0	0	1
3. Environment					
a		Pre defined libraries	1	1	1
b		User defined environments	0	1	1
c		Real world maps	0	1	1
d		Photorealism	0	1	1
e		Variable resolution	0	0	0
f		Context aware map-planning	0	1	0
g		User defined scenarios	0	1	1
h		Material defenition	0	0	1
i		Variable levels of sun shades	1	1	1
j		Special optical feautres	0	0	1
k		Data visualization and analytical tools	0	1	1
l		Rendering	0	1	1
m		Recording	0	1	1
4. Ego Vehicle					
a		Vehicle Geometry	1	1	1
b		Wheel Geometry	0	1	1
c		Real world models	0	1	1
d		Real world physics	0	1	1
e		Input commands	1	1	1
f		Other components	0	0	0
5. Platform					
a		Cross platform Compatability			
	i	Linux	1	1	1
	ii	ROS	1	0	1
	iii	Windows	1	1	0

			CARLA	SUMMIT	LGVSL
	iv	MacOS	1	1	0
	v	Other	1	0	0
b	Open Source		1	1	0
c	Scripting Language support				
	i	Python	1	1	1
	ii	C++	1	1	0
	iii	R	0	0	0
	iv	Java	0	0	0
	v	Others	0	0	0
d	Scalability		0	1	0
e	Computation resource evaluation		0.5	0.5	0.7
f	Community support				
	i	Structured documentation	1	1	1
	ii	Tutorials	1	1	0
	iii	Discussion Forums	0	0	0
g	Distributed Computing		1	0	0
6. Driving Algorithm					
a	Data Exchange		1	1	1
b	Data Logging		1	1	0
c	Hardware in Loop (HiL)		0	0	1
d	ML Libraries		0	0	0

Table 1: Base Score for Carla, Summit. LGVSL

The table displays the criteria weights provided by four distinct users within the comparison system. Here's a brief description of each user and their respective use case: **User1:** A university student primarily focused on projects involving sensory data generation, feedback, and training deep learning models. While realism isn't a top priority, it would be advantageous. **User2:** A software tester at an automobile company concentrating on validating model performance in real-world scenarios. **User3:** A research scholar striving to achieve autonomous driving within a confined environment, aiming to replicate real-world custom environments within the simulator. **User4:** Emphasizes extensive data generation, resembling a large-scale data collector preparing datasets for various tasks, not limited to autonomous driving or machine learning. Hence, customization options and distributed computing hold significant importance.

	User 1	User 2	User 3	User 4
1. Sensors				
Ready to use Sensor				

a

			User 1	User 2	User 3	User 4
	i	RGB Camera	0	1	1	0.6
	ii	Depth Camera	1	1	0.5	1
	iii	3D Lidar	0	1	1	1
	iv	Radar	0	0	0.7	1
	v	IMU Sensor	0	1	0.5	1
	vi	Semantic info Sensor	1	0	0.5	0.6
	vii	Force Impact Sensor	0	1	0	1
	viii	Lane detection/ infringement sensor	1	1	0.1	1
	ix	Fuel/ Battery sensor	0	1	1	1
	x	Vehicle speed sensor	0	1	1	1
	xi	GPS	0	1	1	1
	xii	Visiblity Sensor	0	0	0.1	0.6
	xiii	Other vehicle sensors related to vehicle dynamics	0	1	0.3	0.6
	xiv	Other environment sensors related to perception	1	0	0.2	0.5
	Xv	Macroscopic Global Sensor	0	0	0.5	0.1
b	Multiple Synchronous Sensors		1	1	1	1
c	User defined position and orientation of sensors		0	1	1	1
d	Precision of the Sensor readings		0	1	1	1
e	Availability of real time sensor models		0	1	0.9	1
f	User defined Sensors and Observers		1	0	0.5	0.9
2. Actors						
	Geometry					
	i	Pre defines Scenarios	1	1	1	0.2
	ii	Predefines models for Actors	1	1	1	0.6
	iii	User-Defined Actors	0	1	0.7	1
a	iv	No. of Actors and Spawning Location	0	1	1	1
	v	Recreation of Real time models	0	1	0.7	1
	vi	Path/ destination of Actors	0	1	1	1
	vii	Context aware spawning	0	1	1	0.6
	Behaviour					
	i	Pre defined Rules	0.5	1	1	0.6
	ii	User defined Rules	0.5	1	1	1
b	iii	Distinguishable Behavioural pattern	0	1	0.1	0.6
	iv	Sensitivity	0.5	1	0.3	0.3
	v	Level of aggressiveness in driving	0	1	0.2	0.6

			User 1	User 2	User 3	User 4
	vi	Reproducibility	1	0	1	0.5
3. Environment						
a		Pre defined libraries	1	1	1	0.7
b		User defined environments	0	0	1	1
c		Real world maps	0	1	0.7	0.5
d		Photorealism	0.5	1	0.5	0.9
e		Variable resolution	0	0	0.6	0.6
f		Context aware map-planning	0	1	0.4	0.9
g		User defined scenarios	1	1	0.7	1
h		Material defenition	0	0.5	0.2	0.8
i		Variable levels of sun shades	0	1	0.6	1
j		Special optical feautres	0	1	0.1	0.7
k		Data visualization and analytical tools	1	1	0.6	0.5
l		Rendering	0	1	0.1	0.8
m		Recording	1	0.5	0.7	1
4. Ego Vehicle						
a		Vehicle Geometry	0	1	0.9	0.6
b		Wheel Geometry	0	1	0.2	0.6
c		Real world models	0	1	0.8	0.9
d		Real world physics	0	1	0.9	1
e		Input commands	1	1	0.6	1
f		Other components	0	1	0	0.3
5. Platform						
a		Cross platform Compatability				
	i	Linux	1	1	1	1
	ii	ROS	1	1	1	0.5
	iii	Windows	0.5	0	0.5	0
	iv	MacOS	0	0.5	0	0
	v	Other	0	0	0	0
b		Open Source			0.75	0.5
c		Scripting Language support				
	i	Python	1	1	1	0
	ii	C++	0	1	0.2	1
	iii	R	0	0	0	0
	iv	Java	0	0	0	0
	v	Others	0	0	0.1	0.5
d		Scalability	1	0	0.1	1
e		Computation resource evaluation	1	0	1	1
f		Community support				

			User 1	User 2	User 3	User 4
	i	Structured documentation	1	1	1	1
	ii	Tutorials	1	0	0.7	0.1
	iii	Discussion Forums	1	0.5	0.5	0.1
g	Distributed Computing		0.5	0	0.2	1
6. Driving Algorithm						
a	Data Exchange		1	0	0.9	0.2
b	Data Logging		1	1	1	1
c	Hardware in Loop (HiL)		0	1	1	0.1
d	ML Libraries		1	0	1	0.5

Table 2: User weights table

Bibliography

[1] Md Salman Ahmed, Mohammad Asadul Hoque, and Phil Pfeiffer. "Comparative study of connected vehicle simulators". In: *SoutheastCon 2016*. IEEE. 2016, pp. 1–7.

[2] Niki Aifanti, Christos Papachristou, and Anastasios Delopoulos. "The MUG facial expression database". In: *11th International Workshop on Image Analysis for Multimedia Interactive Services WIAMIS 10*. IEEE. 2010, pp. 1–4.

[3] Moustafa Alzantot, Supriyo Chakraborty, and Mani Srivastava. "Sensegen: A deep learning architecture for synthetic sensor data generation". In: *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE. 2017, pp. 188–193.

[4] Davide Anguita et al. "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine". In: *Ambient Assisted Living and Home Care: 4th International Workshop, IWAAL 2012, Vitoria-Gasteiz, Spain, December 3-5, 2012. Proceedings 4*. Springer. 2012, pp. 216–223.

[5] Panpan Cai et al. "SUMMIT: A Simulator for Urban Driving in Massive Mixed Traffic". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 4023–4029.

[6] CARLA Team. *CARLA Documentation*. `https://carla.readthedocs.io/en/latest/`. 2023.

[7] Marius Cordts et al. "The cityscapes dataset for semantic urban scene understanding". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 3213–3223.

[8] Thomas M Cover. *Elements of information theory*. John Wiley & Sons, 1999.

[9] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[10] Janosch Delcker. "The man who invented the self-driving car (in 1986)". In: *Politico, July* (2018).

[11] Alexey Dosovitskiy et al. "CARLA: An Open Urban Driving Simulator". In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 1–16.

[12] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of machine learning research* 12.7 (2011).

[13] Jakob Geyer et al. “A2d2: Audi autonomous driving dataset”. In: *arXiv preprint arXiv:2004.06320* (2020).

[14] Ian Goodfellow et al. “Generative adversarial networks”. In: *Communications of the ACM* 63.11 (2020), pp. 139–144.

[15] Lena Gorelick et al. “Actions as space-time shapes”. In: *IEEE transactions on pattern analysis and machine intelligence* 29.12 (2007), pp. 2247–2253.

[16] Sonam Gupta, Arti Keshari, and Sukhendu Das. “Rv-gan: Recurrent gan for unconditional video generation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 2024–2033.

[17] Joyce Henry, Terry Natalie, and Den Madsen. “Pix2Pix GAN for Image-to-Image Translation”. In: *Research Gate Publication* (2021), pp. 1–5.

[18] Martin Heusel et al. “Gans trained by a two time-scale update rule converge to a local nash equilibrium”. In: *Advances in neural information processing systems* 30 (2017).

[19] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[20] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural networks* 2.5 (1989), pp. 359–366.

[21] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *International conference on machine learning*. pmlr. 2015, pp. 448–456.

[22] Phillip Isola et al. “Image-to-image translation with conditional adversarial networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1125–1134.

[23] Phillip Isola et al. “Image-to-image translation with conditional adversarial networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1125–1134.

[24] Rui Jin and Qiang Niu. “Automatic Fabric Defect Detection Based on an Improved YOLOv5”. In: *Mathematical Problems in Engineering* 2021 (Sept. 2021), pp. 1–13.

[25] Arti Keshari, Sonam Gupta, and Sukhendu Das. “V3GAN: Decomposing Background, Foreground and Motion for Video Generation”. In: *arXiv preprint arXiv:2203.14074* (2022).

[26] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[27] Nathan Koenig and Andrew Howard. “Design and use paradigms for gazebo, an open-source multi-robot simulator”. In: *2004 IEEE/RSJ international conference on intelligent robots and systems (IROS)(IEEE Cat. No. 04CH37566)*. Vol. 3. IEEE. 2004, pp. 2149–2154.

[28] Solomon Kullback and Richard A Leibler. "On information and sufficiency". In: *The annals of mathematical statistics* 22.1 (1951), pp. 79–86.

[29] G Ajay Kumar et al. "LiDAR and Camera Fusion Approach for Object Distance Estimation in Self-Driving Vehicles". In: *Symmetry* 12.2 (2020). URL: https://www.mdpi.com/2073-8994/12/2/324.

[30] LGVSL Team. *LGVSL Simulator Documentation*. https://www.svlsimulator.com/docs/. 2023.

[31] J Liu, X Huang, and Z Li. "Baidu apollo autonomous vehicles platform". In: *Communications of the CCF* 14.008 (2018), pp. 63–69.

[32] Time Magazine. "Science: Radio auto". In: *Time Magazine* 10 (1925).

[33] Thomas Mauthner, Peter M Roth, and Horst Bischof. *Action recognition from a small number of frames*. na, 2009.

[34] Marvin Minsky and Seymour Papert. "An introduction to computational geometry". In: *Cambridge tiass., HIT* 479.480 (1969), p. 104.

[35] NVIDIA Corporation. *NVIDIA Self-Driving Cars Simulation*. 2023. URL: https://www.nvidia.com/en-us/self-driving-cars/simulation/.

[36] Open Dynamics Engine. *Open Dynamics Engine*. https://opende.sourceforge.net/. Accessed: 2023.

[37] Lucas Pinheiro Cinelli et al. "Variational autoencoder". In: *Variational Methods for Machine Learning with Applications to Deep Networks*. Springer, 2021, pp. 111–149.

[38] A Pranklin. *Introduction to the Theory of Statistics*. 1974.

[39] Guodong Rong et al. "Lgsvl simulator: A high fidelity simulator for autonomous driving". In: *2020 IEEE 23rd International conference on intelligent transportation systems (ITSC)*. IEEE. 2020, pp. 1–6.

[40] *SAE Updates J3016 Automated Driving Graphic*. https://www.sae.org/news/2019/01/sae-updates-j3016-automated-driving-graphic. Accessed on [17.11.2023].

[41] Masaki Saito et al. "Train sparsely, generate densely: Memory-efficient unsupervised training of high-resolution temporal gan". In: *International Journal of Computer Vision* 128.10-11 (2020), pp. 2586–2606.

[42] W Shanshan. "Panosim: a new generation of advanced vehicle intelligent driving simulation system (in chinese)". In: *IEEE Spectrum* 10 (2015), pp. 68–71.

[43] Ilya Shimchik et al. "Golf cart prototype development and navigation simulation using ROS and Gazebo". In: *MATEC Web of Conferences*. Vol. 75. EDP Sciences. 2016, p. 09005.

[44] Yale Song, David Demirdjian, and Randall Davis. "Tracking body and hands for gesture recognition: Natops aircraft handling signals database". In: *2011 IEEE International Conference on Automatic Face & Gesture Recognition (FG)*. IEEE. 2011, pp. 500–506.

[45] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. "UCF101: A dataset of 101 human actions classes from videos in the wild". In: *arXiv preprint arXiv:1212.0402* (2012).

[46] Summit Team. *SUMMIT Documentation*. `https://adacompnus.github.io/summit-docs`. 2023.

[47] Pei Sun et al. "Scalability in perception for autonomous driving: Waymo open dataset". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 2446–2454.

[48] Shan Sun et al. "Taichi: A fine-grained action recognition dataset". In: *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*. 2017, pp. 429–433.

[49] Murat Taskirar et al. "Face recognition using dynamic features extracted from smile videos". In: *2019 IEEE International Symposium on INnovations in Intelligent SysTems and Applications (INISTA)*. IEEE. 2019, pp. 1–6.

[50] The Khronos Group. *OpenGL*. `https://www.opengl.org/`. Accessed: 2023.

[51] Sergey Tulyakov et al. "Mocogan: Decomposing motion and content for video generation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 1526–1535.

[52] *Unity*. `https://unity.com/`. Accessed: 31/12/2023.

[53] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. "Generating videos with scene dynamics". In: *Advances in neural information processing systems* 29 (2016).

[54] Yaohui Wang et al. "Imaginator: Conditional spatio-temporal gan for video generation". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2020, pp. 1160–1169.

[55] *Waymo*. `https://waymo.com/`. Accessed: 31/12/2023.

[56] Bernhard Wymann et al. "Torcs, the open racing car simulator". In: *Software available at http://torcs. sourceforge. net* 4.6 (2000), p. 2.

[57] Pengchuan Xiao et al. "Pandaset: Advanced sensor suite dataset for autonomous driving". In: *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. IEEE. 2021, pp. 3095–3101.

[58] Guang Yang et al. "Survey on autonomous vehicle simulation platforms". In: *2021 8th International Conference on Dependable Systems and Their Applications (DSA)*. IEEE. 2021, pp. 692–699.

[59] Matthew D Zeiler. "Adadelta: an adaptive learning rate method". In: *arXiv preprint arXiv:1212.5701* (2012).

Declaration on oath

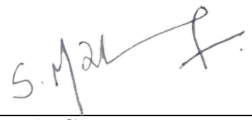
I hereby certify that I have written my master thesis independently and have not yet submitted it for examination purposes elsewhere. All sources and aids used are listed, literal and meaningful quotations have been marked as such.

Mahesh Saravanan, 17.01.2024

Consent to plagiarism check

I hereby agree that my submitted work may be sent to PlagScan (www.plagscan.com) in digital form for the purpose of checking for plagiarism and that it may be temporarily (max. 5 years) stored in the database maintained by PlagScan as well as personal data which are part of this work may be stored there.

Consent is voluntary. Without this consent, the plagiarism check cannot be prevented by removing all personal data and protecting the copyright requirements. Consent to the storage and use of personal data may be revoked at any time by notifying the faculty.



Mahesh Saravanan, 17.01.2024