Suicidal Pedestrian: Generation of safety-critical scenarios for autonomous vehicles

Yuhang Yang

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 17.4.2023

Supervisor

Prof. Alexander Ilin
Prof. Joni Pajarinen

Advisor

Dr. Amin Babadi
M.Sc. Kalle Kujanpää

Aalto University
School of Electrical Engineering

Aalto University
School of Electrical
Engineering

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Yuhang Yang

Title Suicidal Pedestrian: Generation of safety-critical scenarios for autonomous vehicles

Degree programme Automation and Electrical Engineering

Major Control, Robotics and Autonomous Systems **Code of major** ELEC3025

Supervisor Prof. Alexander Ilin
Prof. Joni Pajarinen

Advisor Dr. Amin Babadi
M.Sc. Kalle Kujanpää

Date 17.4.2023 **Number of pages** 53+2 **Language** English

Abstract

Autonomous driving is appealing due to its significant financial potential and positive social impact. However, developing capable autonomous driving algorithms faces the difficulty of reliability testing because some safety-critical traffic scenarios are particularly challenging to acquire. To this end, this thesis proposes a method to design a suicidal pedestrian agent based on the CARLA simulation engine that can automatically generate pedestrian-related traffic scenarios for autonomous vehicle testing. In this method, the pedestrian is formulated as a reinforcement learning agent that spontaneously seeks collisions with the target vehicle and is trained using a continuous model-free learning algorithm with two custom reward functions. Besides, by allowing the pedestrian freely explore the environment with a constrained initial distance to the vehicle, the pedestrian and autonomous car can be placed anywhere, rendering generated scenarios more diverse. Furthermore, four collision-oriented evaluation metrics are also proposed to verify the performance of the designed suicidal pedestrian and the target vehicle under testing. Experiments on two state-of-the-art autonomous driving algorithms demonstrate that this suicidal pedestrian is effective in finding autonomous vehicle decision errors when cars are exposed to such pedestrian-related traffic scenarios.

Keywords Autonomous Driving, Suicidal Pedestrian, Traffic Scenario Generation, Reinforcement Learning, CARLA Simulator

Preface

This thesis was conducted at the Long Term Decision Making and Transfer Learning group, Finnish Center for Artificial Intelligence, Aalto University. I want to thank Prof. Alexander Ilin and Prof. Joni Pajarinen for providing me with this project and for their kind help at each step. I am grateful for this opportunity to work on a thesis project that highly interests me. I would also like to especially thank my advisors M.Sc. Kalle Kujanpää and Dr. Amin Babadi for their valuable comments and guidance. Without their feedback and help, I would need much more time to finish this thesis.

Otaniemi, 17.4.2023

Yuhang Yang

Contents

Abbreviations

A3C	Asynchronous Advantage Actor-Critic
AD	Autonomous Driving
ALVINN	Autonomous Land Vehicle In a Neural Network
AV	Autonomous Vehicle
CARLA	Car Learning to Act
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q-Learning
GNSS	Global Navigation Satellite System
GTA V	Grand Theft Auto V
IA	Implicit Affordance
I2A	Imagination-Augmented Agent
IMU	Inertial Measurement Unit
InterFuser	Interpretable Sensor Fusion Transformer
LAV	Learning from All Vehicles
LBC	Learning by Cheating
LiDAR	Light Detection and Ranging
LQR	Linear Quadratic Regulator
MAD-ARL	Multi-Agent Driving with Adversarial Reinforcement Learning
MB-MPO	Model-Based Meta-Policy-Optimization
MDP	Markov Decision Process
ME-TRPO	Model-Ensemble Trust-Region Policy Optimization
MPC	Model Predictive Control
OU	Ornstein-Uhlenbeck
PID	Proportional-Integral–Derivative
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
SAC	Soft Actor-Critic
STEVE	Stochastic Ensemble Value Expansion
SUMO	Simulation of Urban Mobility
TD	Temporal Difference
TORCS	The Open Racing Car Simulation
TRPO	Trust Region Policy Optimization
WOR	World on Rail

1 Introduction

Autonomous driving (AD) is an appealing area of research with significant potential for improving household mobility, optimizing traffic efficiency, and ensuring safety. However, developing a capable AD system is not a trivial task. In order to cope with highly complex and diverse traffic scenarios, autonomous vehicles (AVs) need sufficient capacity to analyze the environment and make online decisions effectively, followed by performing low-level control commands. This requires many sub-systems to cooperate, including perception modules, prediction and planning modules, as well as control modules.

In recent years, with the development of modern artificial intelligence, especially in computer vision and reinforcement learning (RL) areas, AD has received much more attention, and rapid progress has been witnessed. Despite these advancements, AD is still challenging, and deploying it to scale is far from reality. One natural reason is the inability of AVs to tackle the complex, dynamic, and contingent scenarios of the real world, also known as long-tail distributions, such as vehicles running a red light, cargo falling from vehicles in front, or the sudden emergence of pedestrians from roadsides [1]. Traditional rule-based planning methods [2, 3] have utilized delicate hand-designed rules to solve such diverse safety-critical scenarios. Unfortunately, these methods suffer from generalizing unseen cases since it is impossible to enumerate all safety-critical situations. Nevertheless, learning-based methods [4, 5] directly learn behaviors from driving data without human-involved engineering. Since no hand-crafted rules are involved, these methods usually perform better in unfamiliar situations and indicate a promising direction.

Numerous studies have been developed through the second track, the learning-based one, to solve long-tail distributions. Based on the methodology used in these studies, they can be categorized into two distinct approaches: capacity enhancement approaches and scenario simulation approaches. Capacity enhancement approaches focus on improving the comprehensive scene-understanding ability and the sequential decision-making process of AVs by using more complex sensor configurations and more advanced learning algorithms [4, 6, 7]. In contrast, scenario simulation methods address the creation of safety-critical scenarios using simulators to imitate the real world and the generation of synthetic data [8, 9, 10, 11], which is one way that forces AVs to experience more diverse situations. Although many methods from both approaches are helpful, most of them either only consider the behaviors of vehicles while ignoring the other important traffic participant, the pedestrians [8, 10], or only model the motion of pedestrians in a simplistic and deterministic way [11, 12]. On the one hand, such constraints significantly degrade the capacity of vehicles to tackle pedestrian-related traffic scenarios, especially when those cars are in the urban environment. Another weakness is that they increase the complex and diverse gap between the simulation and the real world, rendering the simulated scenarios less general.

Therefore, the objective of this thesis is to design a new critical scenario with the participation of pedestrians via simulators that can be used to evaluate the capacity of AD algorithms dealing with such a case. Typically, this thesis is restricted to a

simplified urban traffic scene in which only two dynamic objects exist: one pedestrian and one vehicle. In this scene, the pedestrian behaves like a suicidal person, meaning that this pedestrian tries to spontaneously hit the running vehicle controlled by some AD algorithms when that vehicle drives close to the pedestrian. However, the development and training of driving algorithms that control the car will not be considered in this work since the vehicle only needs to respond to this situation rather than serve as a component of it. In order to accomplish this task, this thesis will separate the objective into two parts. The first part focuses on constructing the required urban traffic scene with simulators and modeling the behavior of pedestrians using RL methods. The second part will compare a series of state-of-the-art AD algorithms to evaluate their capacities in terms of avoiding collision with such a suicidal pedestrian.

The rest of this thesis is organized as follows. First, Chapter 2 lays a foundation to provide background knowledge on the Car Learning to Act (CARLA) [13] open-source urban simulation platform, ideas of RL methods, state-of-the-art AD algorithms, and trends in traffic scenario simulation. Then, Chapter 3 describes a thorough explanation of the developed method for this proposed traffic scenario design problem, including the scene construction, suicidal pedestrian modeling, and evaluation criteria proposal. Next, Chapter 4 presents the experiment results and corresponding performance analysis. Finally, Chapter 5 reviews this thesis, summarizing its main contributions, pointing out shortcomings, and prospecting possible extensions and improvements of this work.

2 Background

This chapter discusses the necessary theoretical knowledge for understanding the proposed methods described in the next chapter. First, the simulation platform used to implement the traffic scenario is explained. Second, ideas of RL are discussed. Third, some state-of-the-art AD algorithms are described. Finally, methods for traffic scenario simulation are argued.

2.1 Carla simulator

AD is data-hungry: in order to have sufficient capacity to deal with highly complex traffic situations in the physical world, AVs need to expose themselves to various scenes to collect useful driving logs. However, collecting data in the real world is not desired. On the one hand, driving an expensive test car with well-configured sensors to collect data is time-consuming and costly, especially when some logs in dangerous scenarios are needed. On the other hand, training and testing developed AVs in the physical world face logistical and management difficulties.

An alternative way is to collect data via simulation. There are many simulators to support AD research. CarSim is a powerful commercial software package developed by the Mechanical Simulation Company. It provides detailed and accurate dynamics models for different passenger vehicles and light-duty trucks. Furthermore, the commercial video game Grand Theft Auto V (GTA V) has been used to gather diverse three-dimensional environment data due to its large world, availability of numerous object models, and high-quality graphics [14, 15]. And the acquired data from GTA V is then used for training perception modules [16]. In contrast to commercial simulation platforms, there are also many open-source simulators. For example, The Open Racing Car Simulator (TORCS) [17] is a highly portable, modular racing simulator that could be used to verify the functions of new AD approaches. It provides more than 50 different types of cars, more than 20 racing tracks, and can support multi-player at the same time. In addition, the Simulation of Urban Mobility (SUMO) [18] provides a more complex traffic flow simulation platform with multi-modal sensor configurations and customized map generation, and a lot of improvements have been made since its publication [19]. Moreover, AutonoVi-Sim [20] is also a widely used simulator for AD data generation and algorithm verification.

Among those open-source simulators, a newly developed urban driving simulator called CARLA [13] attracts great interest from the AD community since it provides a detailed, flexible, modular, and high-quality urban driving platform. CARLA is grounded on Unreal Engine 4 [21] and uses a scalable server-client structure to control the interaction between the simulation and the user agent. The server is responsible for simulation-related tasks, such as physics computation, world-state updates, and scene rendering. The client consists of a series of modules controlling the world condition settings and logic of actors, which is implemented by Python API. Typically, the client sends instructions to the server to guide the simulation and receives sensor data in return. And instructions can be divided into two types: actor command and meta-command. Actor command controls the pedestrians, vehicles,

and sensors, while meta-command controls the server, including changing maps and weather of the simulation environment, resetting the simulation, and modifying the graphical quality.

Compared to other existing open-source simulators [17, 18, 20], CARLA [13] has three main advantages. First, it focuses on the urban environment, providing rich urban environmental objects such as shops, residential buildings, dustbins, and traffic lights, allowing AVs to be exposed to more complex and dangerous traffic scenarios. Second, a rich command set, flexible sensor configuration, and detailed control feedback are provided, thus supporting develop more complex AD approaches and simplifying the analysis of developed driving policies. Third, more types of vehicles and pedestrians are modeled in the simulator, which increases the simulation diversity.

Since the superiority for simulating urban environments of CARLA, and the thesis work objective of designing a dangerous traffic scenario with a suicide pedestrian that tries to hit AVs in one urban environment, the CARLA simulator is decided to serve as the implementation platform. Therefore, a detailed introduction to CARLA is needed. In this section, components of CARLA are provided. First, the static environment is discussed. Then, sensors and their attributes are discussed. Finally, pedestrians and vehicles are described.

2.1.1 Static environment

The static environment of CARLA is composed of two parts: the physical part and the virtual part. The physical part consists of various 3D models, which serve as basic blocks for constructing environment maps, while the virtual part is responsible for controlling environmental weather and illumination regimes.

The physical static 3D models, such as road lines, traffic signs, buildings, walls, traffic lights, vegetation, and other environmental objects, are all designed under the realism guideline to reflect their physical properties. By leveraging these models, CARLA builds eight towns covering urban, highway, and rural environments, which significantly increases the possibility of constructing highly complex and diverse traffic scenarios that are necessary for more advanced AD approaches.

Among these provided towns, Town 1 and Town 2 simulate similar residential areas with 2-lane roads and T-junctions. They are suitable for simulating pedestrian-related traffic situations due to the narrow roads and widespread sidewalks. Town 3 is the most complex map that simulates a central modern town with 5-lane junctions, roundabouts, lane merge and split, and other urban traffic networks. This map is typically used as a baseline to exhaustively evaluate the performance of AD algorithms in city environments. Town 4 and Town 6 contain long highways. However, Town 4 also simulates a small town surrounded by a highway, while Town 6 simulates many entrances and exits along the highway. Town 5 and Town 10 are similar to Town 3 but with different features, where the former has multiple lanes per direction that are preferred for testing the lane change behavior of AVs, and the latter is a combination of various city environments such as promenade and avenue. Finally, Town 7 simulates a rural area without traffic lights and signs. Three overviews of

Figure 1: The bird's-eye view of three CARLA towns in the same weather condition. From left to right: Town 2, Town 3, and Town 5. Town 2 simulates residential areas with 2-lane roads, while Town 3 and Town 5 simulate central areas with both 2-lane and multiple-lane roads.

these towns are illustrated in Figure 1.

In addition to eight town maps, CARLA also implements the control of environmental weather and illumination. Such functions provide more diversity to each town, further extending the capacity to simulate the changing natural environment of the real world. The weather controller can control the cloudiness of the sky, precipitation, wind intensity, fog intensity, and wetness. The illumination controller is responsible for the position and color of the sun, sunlight intensity, and penetration. Currently, CARLA provides three pre-defined light conditions: noon, sunset, and night and nine pre-defined weather conditions, resulting in 27 different combinations in total. However, user-defined weather-illumination combination beyond this set is still allowed.

2.1.2 Dynamic objects

The dynamic objects provided in CARLA can be divided into three categories: pedestrians, vehicles, and sensors. However, sensors must be attached to one pedestrian or vehicle, while pedestrians and vehicles could be spawned alone.

Vehicles are one of the most important entities in CARLA. Currently, 27 different vehicle models are provided, ranging from motorcycles to cars and light trucks, and rich commands from different levels can be applied to control their behavior. The low-level commands, such as *throttle*, *steer*, and *brake*, directly control the vehicle's motion. The high-level commands, including *turn-left*, *turn-right*, *lane-follow*, *go-straight*, *change-lane-to-left*, and *change-lane-to-right*, are used to describe the behavior of vehicles in a more human-kind and general way. Such a hierarchical command structure provides sufficient command information, making it easier to train various AD algorithms that use different control commands as supervision. Moreover, it also provides a chance to flexibly develop various sub-modules according to the control command, such as designing a more robust controller using the low-level command.

In addition to the control command, CARLA also provides a path-planning

module and a controller module to support navigation, where the planning module utilizes the well-known A* algorithm and the controller module utilizes the proportional–integral–derivative (PID) method. Based on these two supplementary modules, rule-based driving policies [3, 22] can be used for non-player vehicles, thus decreasing the workload of creating a simulation world with multiple cars.

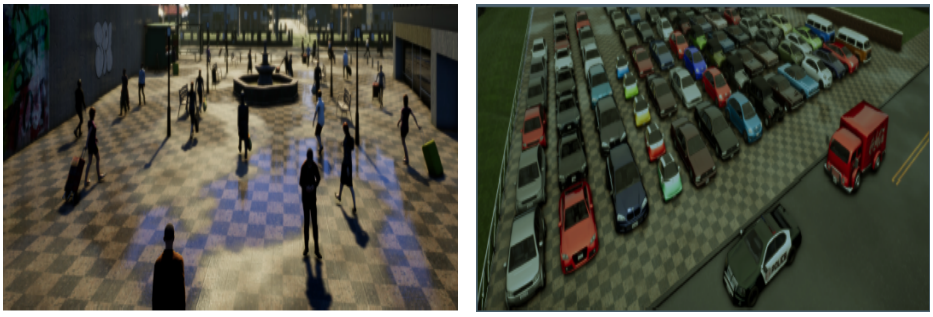


Figure 2: Some vehicles and pedestrians available in CARLA [13].

Pedestrians are the second dynamic objects in CARLA that can be spawned alone. Similar to vehicles, the control of pedestrians can be achieved by commands and pre-defined controllers. The pedestrian control commands include *direction*, *speed*, and *jump*. In order to move pedestrians, commands should be updated and applied during every simulation step. In contrast, the pre-defined pedestrian controller will automatically navigate the walkers as long as their destinations are set. And walkers are encouraged to walk along marked road crossings and sidewalks, as well as avoid collision with vehicles once the controller rules them.

Sensors are essential for AVs to perceive the surrounding environments and their inner states. CARLA provides different kinds of sensor modalities to allow flexible configurations for autonomous cars. Sensors that perceive the environment include light detection and ranging (LiDAR), camera, and radar. Moreover, To simplify the process of training and evaluating AD approaches, cameras have three variants of output: RGB image, ground-truth depth image, and ground-truth semantic image, and LiDARs provide two variants of output: point cloud and ground-truth semantic point cloud. An example of these outputs is shown in Figure 3. Sensors that estimate the inner states of AVs are limited to the global navigation satellite system (GNSS) and inertial measurement unit (IMU). And measurements of the inner states include location and orientation concerning the world coordinate, speed, and acceleration.

In addition to these environmental perception and inner state estimation sensors that send readings at every simulation step, CARLA also provides sensors associated with collision and traffic rules that send readings only when they are triggered by events. The collision sensor detects collision events between its parent vehicle and objects in the world. The lane invasion sensor detects the behavior that vehicles invade sidewalks or wrong-lane. Finally, the obstacle sensor detects objects ahead of its parent vehicle. These sensors provide more detailed signals for evaluating AVs.

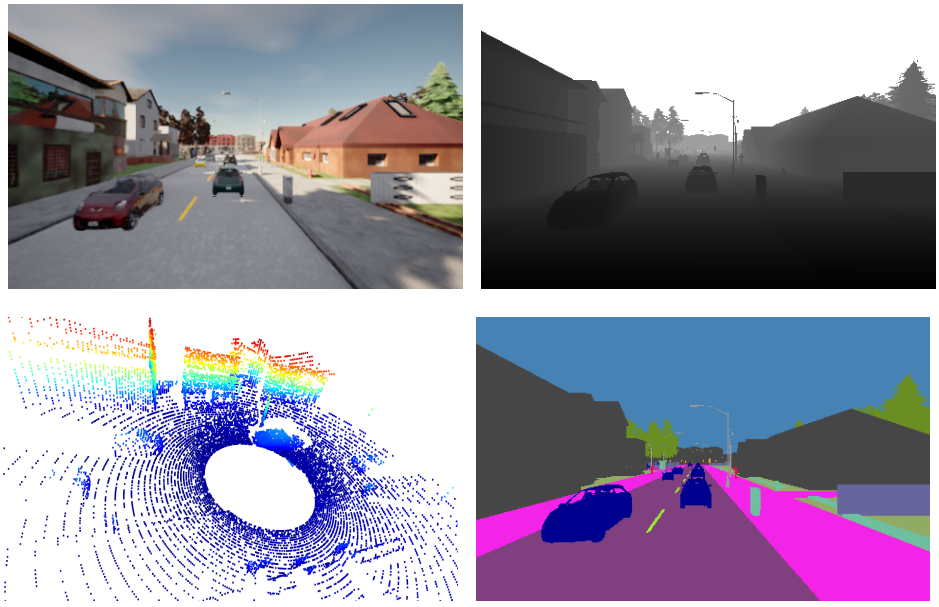


Figure 3: Four of the sensing modalities in CARLA. Clock-wise from top left: normal camera, depth camera, semantic segmentation camera, and 3D LiDAR.

2.2 Reinforcement learning

RL is a subfield of machine learning that learns behavior through a trial-and-error style without explicit human supervision [23], and it is widely used in AD area [6, 7, 13, 24]. Its typical feature is that the RL agent learns by actively interacting with the environment throughout the process. At each time step t, the agent observes a particular state s_t from the environment, and it should choose an action a_t based on the current state and its policy $\pi(a_t|s_t)$, a function that maps states to actions, to transition to a new state s_{t+1}, followed by receiving a scalar reward $R(s_{t+1}, s_t, a_t)$. This perception-action loop is illustrated in Figure 4.

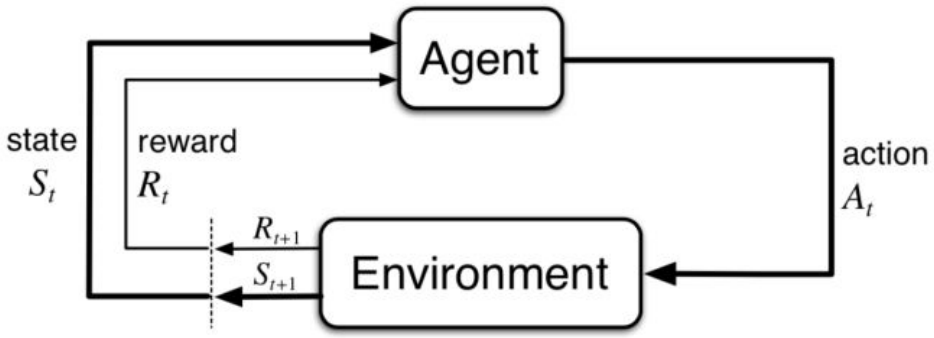


Figure 4: The interaction process between an RL agent and environment.

According to the interaction, different policies will determine different sequences of actions, thus resulting in different rewards. Therefore, the goal of the agent is to

find an optimal policy, denoted as π^*, that maximizes the expectation of cumulative discounted reward. Typically, the cumulative discounted reward is called the return, and it is defined as:

$$G_t = \sum_{k=0}^{T} \gamma^k R(s_{t+k+1}, s_{t+k}, a_{t+k}), \tag{1}$$

where t is the time step the agent currently stays in, $\gamma \in [0,\ 1]$ is the discount factor, and T is the time horizons. However, learning an optimal policy is still very difficult even though the return is formally formulated. The main challenges come from two aspects. First, since the state transition dynamics are not available to the RL agent, it needs to learn about policies by trial and error. Second, in-depth evaluation criteria during the interaction are unknown. This leads to a situation in which the agent learns without indicating whether the agent is in the right direction. Considering these problems, some mechanisms are proposed, which will be discussed in this section.

Markov decision process (MDP) is a mathematical framework to describe discrete-time stochastic control processes [23, 25], which could be used to describe the interaction between RL agents and environments. Formally, an MDP is denoted as a 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where:

- $\mathcal{S}$ is the set of states, called state space
- $\mathcal{A}$ is the set of actions, called action space
- $\mathcal{P}$ is the transition probability function: $\mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$, which describes the probability distribution of entering a new state $s_{t+1} \in \mathcal{S}$ at time $t+1$ from a state-action pair $(s_t, a_t) \in (\mathcal{S} \times \mathcal{A})$ at time t
- $\mathcal{R}$ is the immediate reward function: $\mathcal{S} \times \mathcal{S} \times \mathcal{A} \to R$, which describes the received scalar reward $R(s_{t+1}, s_t, a_t)$ after transitioning from state s_t to state s_{t+1} due to the performed action a_t
- γ is the discount factor satisfying $0 \leq \gamma \leq 1$, representing the importance of considering long-term rewards

Most RL problems could be formulated as MDPs, and a key concept underlying each MDP is the Markov property that the future state depends only on the current state and action, while the past could be thrown away. This means the current state s_t is a sufficient statistic for the environment that comprises all necessary information for making an action a_t, or in other words, any actions made at time step t can be based solely on the current state s_t rather than the sequence of previous states $\{s_1, s_2, ..., s_t\}$.

Value functions estimates the goodness of each state or each state-action pair in terms of return. Formally, two value functions are utilized: the state-value function

and the action-value function. The state-value function $V_\pi(s)$ is the expected return from state s when following policy π:

$$V_\pi(s) = E_\pi\Big[G_t \;\Big|\; \mathcal{S}_t = s\Big]. \tag{2}$$

Similar to the state-value function, the action-value function is the expected return for taking action a at state s and then following policy π:

$$Q_\pi(s,a) = E_\pi\Big[G_t \;\Big|\; \mathcal{S}_t = s, \mathcal{A}_t = a\Big]. \tag{3}$$

By exploiting the Markov property, the above two functions can be rewritten into a recursive form, and the so-called Bellman equations [26] are obtained:

$$\begin{cases} V_\pi(s) & = E_\pi\Big[R\Big(s', s, \pi(s)\Big) + \gamma \cdot V_\pi(s') \;\Big|\; \mathcal{S}_t = s\Big] \\ Q_\pi(s,a) & = E_\pi\Big[R(s', s, a) + \gamma \cdot Q_\pi\Big(s', \pi(s')\Big) \;\Big|\; \mathcal{S}_t = s, \mathcal{A}_t = a\Big] \end{cases} \tag{4}$$

One significant advantage of equation 4 is that it simplifies the multi-step sequential summation problem into a two-step summation problem with two distinct parts, where the first part is an immediate reward received by selecting an action at current time step, and the second part is a value estimate of the next state. This means the current estimations are simple combinations of the immediate reward and the discounted future measures, and thus step-by-step updates are allowed to approximate value functions.

As mentioned before, RL algorithms target finding an optimal policy π^* that receives a maximal expected return. Therefore, the optimal policy should have a corresponding optimal state-value function and action-value function:

$$\begin{cases} V_*(s) & = \max\limits_\pi V_\pi(s), & \forall s \in \mathcal{S} \\ Q_*(s,a) & = \max\limits_\pi Q_\pi(s,a), & \forall s \in \mathcal{S}, a \in \mathcal{A} \end{cases} \tag{5}$$

Exploration and exploitation are two fundamental but contradictory cores in RL [23, 27]. On the one hand, the agent should try out more non-optimal actions to obtain comprehensive information about the environment to reduce its uncertainty regarding transition probabilities and the reward function. On the other hand, in order to avoid inefficiency and make useful progress, the agent should exploit its current estimation of the environment to choose the best action with the most reward. One simple but effective method to deal with this exploration-exploitation trade-off problem is to use randomized methods as policies, such as epsilon-greedy policy and the upper confidence bound (UCB) [28] algorithm, or to add random noise to actions, such as Gaussian noise and Ornstein-Uhlenbeck (OU) noise [29]. Moreover, advanced methods that design intrinsic rewards [30, 31] to motivate exploration are applied to larger, more challenging environments with sparse rewards and they have achieved better performance.

2.2.1 Model-free methods

Model-free RL is a general category in the RL community that contain a series of algorithms. It assumes that the environment is a black box that produces rewards for agent actions. By this means, the RL agent only needs to learn a policy to maximize the received rewards without knowing the environment, in other words, the transition probabilities and the reward function. The main advantage of leaving an environment model is that it is easier to implement and allows agents to learn optimal policies with zero bias due to the absence of environment modeling errors. According to Arulkumara et al. [27], a lot of model-free RL algorithms have been developed, and the most notable state-of-the-art ones are DQN [32], A3C [25], PPO [33], DDPG [29], and SAC [34].

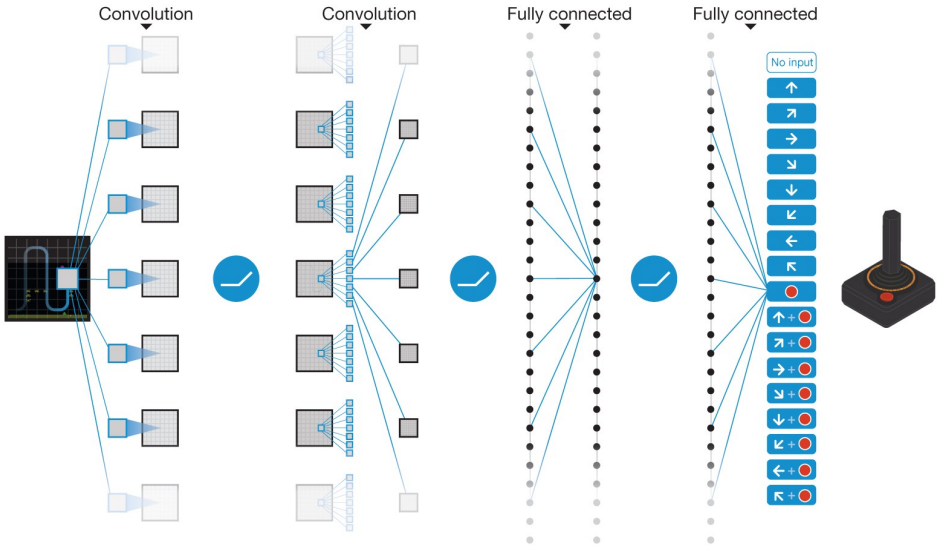


Figure 5: Schematic illustration of the convolutional neural network used in DQN for image inputs [35]. The input consists of an image in the shape of $84 \times 84 \times 4$, followed by three convolutional layers and two fully connected layers.

DQN: Mnih et al. [32] proposed the first practical RL algorithm involving deep neural networks. The idea of DQN arises from the fact that deep convolutional neural networks are powerful for compact information representation and have strong function approximation abilities: with high-dimensional images as inputs, the network, as shown in Figure 5, can be trained with a variant of Q-learning to directly estimate Q values of all possible state-action pairs. This is done through two techniques: experience replay and target networks. Experience replay is a cyclic buffer that stores historical transitions in the form of $(s_t, a_t, s_{t+1}, R(s_{t+1}, s_t, a_t))$, enabling the RL agent to sample training data offline uniformly. Such a design not only breaks down the temporal correlations between transitions that cause instability of algorithm training but also massively improves the data utilization efficiency so that the same data can be used several times. The target network is a copy of the main network enacting the policy, but with frozen weights for a long time. During training, the policy network calculates the temporal difference (TD) error based on Q values from the fixed target

network instead of its own, avoiding divergence problems because of rapid changes in TD errors. Furthermore, in order to match the policy network, the weights of the target network are updated after a fixed number of iterations.

A3C: Mnih et al. [25] proposed the Asynchronous Advantage Actor-Critic (A3C) algorithm, an asynchronous, multi-agent, distributed version of the Advantage Actor-Critic. The A3C follows the actor-critic formulation and updates the advantage estimation in parallel. It consists of several actor-critic networks with the same architecture. The global network is a weight center in which the most recently updated weights are stored. Other transcript networks independently control their own agents to interact with the environments and update weights asynchronously once enough data is collected, followed by uploading the latest weights to the global network. This framework is illustrated in Figure 6. With the help of multiple agents and independent environments, A3C allows more exploration and converges fast to optimal policies.

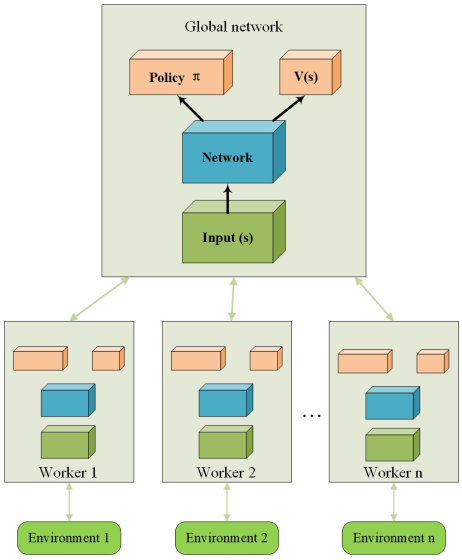


Figure 6: The framework of A3C. Each network shares the same architecture. The global network does not interact with the environment but merely stores the latest weights from all worker networks. The worker networks interact with the environments independently and improve their parameters asynchronously. During training, each worker network firstly copies weights from the global network and then improves parameters based on the collected data, followed by updating weights back to the global network.

PPO: Schulman et al. [33] considered the calculation difficulty of the second-order gradient in Trust Region Policy Optimization (TRPO) [36] and then developed Proximal Policy Optimization (PPO) to approximate the calculation. Compared to directly calculating a constrained quadratic problem requiring the second-order gradient, PPO modifies the objective into an unconstrained problem with only the first-order gradient information needed. Furthermore, it uses two main variants for approximation: the heuristic objective clip and the adaptive penalty on the Kullback-

Leibler (KL) divergence. Since PPO dramatically decreases computation complexity while retaining competitive performance and contains few hyperparameters to tune, it is becoming more popular in many RL tasks.

DDPG: Lillicra et al. [29] extended the policy gradient theorems from stochastic policies to deterministic policies and trained them with actor-critic architecture in the offline mode. Similar to DQN [32], DDPG also utilizes experience replay and target networks to guarantee stability during training. Moreover, DDPG addresses exploration ability by adding random noise to actions. Typically, Gaussian noise and OU noise are two suitable choices. The Gaussian noise is not temporally correlated and widely spreads in the natural world; therefore, it is a good starting point for many tasks. In contrast, the OU noise is temporally correlated and can give linearly negative feedback around the mean, leading it suitable for physical control problems with inertia.

SAC: Haarnoja et al. [34] proposed another offline algorithm with the actor-critic architecture called Soft Actor-Critic (SAC) that provided for both data efficiency and stability. The central idea of SAC is entropy regularization: in order to handle very complex and high-dimensional tasks, the policy entropy needs to be maximized to force the policy to be more stochastic; meanwhile, a high return is still desired for finding a good policy, and thus the entropy should be limited. This is done by adding an entropy term to the actor-network and to the calculation of TD errors with an adaptive entropy coefficient. In addition, another key idea in SAC is that it uses two critic networks to avoid overestimated Q values. During training, the SAC will always choose one critic network with a smaller estimated Q to calculate the TD error. Finally, offline techniques, such as experience replay and target networks, also play essential roles in SAC.

2.2.2 Model-based methods

All algorithms described above are model-free, where the RL agent directly learns optimal value functions or policies without knowing the environment dynamics. Although they have achieved outstanding performance in many tasks, applying them to practical problems in the real world is still a challenge. The main reason lies in the requirement for a massive amount of data. For example, in order to learn to walk on uneven ground, the legged robot should try thousands of episodes to collect enough data in the simulation, which is quite time-consuming and costly in the real world. Therefore, another type of RL algorithm, model-based RL, is proposed to reduce the number of required interactions. The key idea of model-based RL is to learn a transition function and a reward function that can describe how the environment changes. Furthermore, once the agent knows the environment model, it is straightforward to find an optimal policy using some online methods without interacting with the environment directly.

However, modeling the environment takes much work. The biggest challenge is that the model should be zero bias; otherwise, the agent will find policies that perform well with respect to the learned model but behave terribly in the real world. Recently, a series of model-based RL algorithms with excellent performance have

been developed, and some representative methods are I2A [37], ME-TRPO [38], MB-MPO [39], and STEVE [40].

Imagination-Augmented-Agent (I2A) [37] is a Deep-RL architecture that combines model-free and model-based aspects. It assumes the learned environment is imperfect and the direct interaction with the environment plays an important role in optimal policy search even though the model is known. I2A consists of three key components: environment modeling, imagination policy rollout, and policy combination. The environment modeling part approximates the transition probabilities and reward functions based on the data obtained by interacting with the true environment. The imagination policy rollout part trains a rollout policy to generate different trajectories using the trained environment model. Given a state, the rollout policy will choose actions to transition to new states inside the environment model. This process continues for several steps and thus generates some imaginary trajectories. Although these trajectories may not give high rewards, they can provide more information than true data, since they are sampled from much more diverse distributions. Finally, the policy combination part utilizes both the true and imagined data to generate an optimal policy using some model-free RL algorithms mentioned above. The overall architecture of I2A is illustrated in Figure 7.

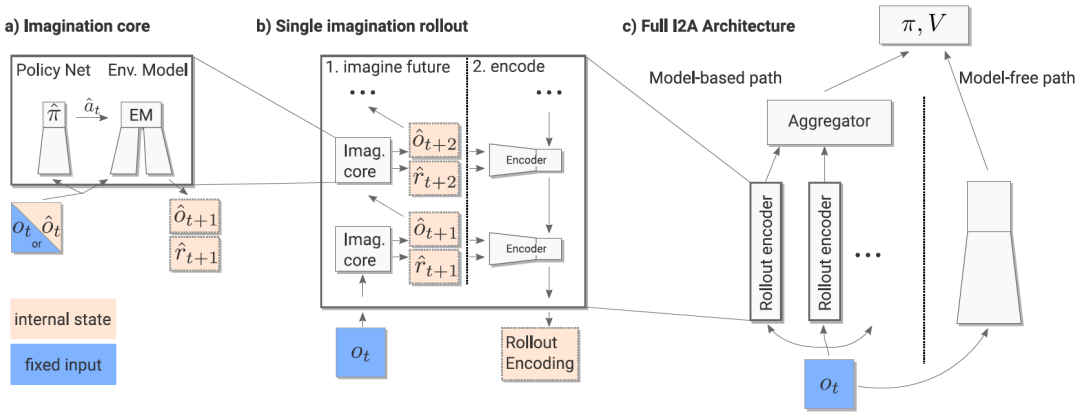


Figure 7: The overall framework of I2A [37].

Model-Ensemble TRPO (ME-TRPO) [38] assumes that an ensemble of environment models can help reduce the model bias and policy can be trained with model-free algorithms purely on imaginary data. It inherits from the vanilla method but with some modifications. Firstly, ME-TRPO uses a model ensemble, a set of environment models learned using the same real data, to alleviate potential sparse distributions that cause bias, and these models only differ from the initial weights and the order in which mini-batches are sampled during training. Additionally, the TRPO [36] algorithm is applied to optimize the policy over the model ensemble. During optimization, each model inside the model ensemble will generate a trajectory so that the policy will not overfit a particular model. Finally, the policy's performance is monitored on validation data generated by the model ensemble. Once there are few performance improvements for the policy, ME-TRPO will stop and return to the current model ensemble and policy.

Model-Based Meta-Policy Optimization (MB-MPO) [39] is another deep model-based RL architecture with the idea of using an ensemble in an orthogonal way: the learned environment models do not need to be sufficiently accurate since policies optimized by the meta-learning technique can quickly adapt to any of the fitted environment models with one gradient step. In MB-MPO, the environment models are trained via a standard supervised learning framework, while the policy optimization is defined as a meta-learning problem. First, the target policy generates new policies with adapted parameters to each model dynamics. Then, those adapted policies will collect trajectories by interacting with their models and be optimized. Finally, the target policy is trained with optimized policies. This process continues until the target policy does not improve further. With the help of meta-learning, MB-MPO provides a solution to the policy optimization problem with inaccurate model dynamics and significantly enhances the policy robustness to model imperfections.

Stochastic Ensemble Value Expansion (STEVE) [40] further extends the ensemble idea to both the models and action-value functions. The key idea behind STEVE is to interpolate between different horizon lengths so that the imaginary trajectory length can be adjusted dynamically to fit various tasks or environments. In STEVE, each model and action-value function will generate one imaginary trajectory with a fixed rollout length. By approximating the uncertainty of each trajectory with different lengths, there is only one segment with the lowest uncertainty could be chosen as the target, while others need to be trained to minimize the difference to that target. Compared to other model-based RL algorithms, such as I2A [37] and ME-TRPO [38], STEVE achieves better performance in more noisy and complex environments.

2.3 Autonomous driving algorithms

AD is a highly complex system composed of many subsystems working together to achieve autonomy in diverse, dynamic, and contingent traffic scenarios of the real world. These subsystems include perception, planning, and control. The perception system aims to help AVs understand the surrounding environment. It takes raw data from different sensors, such as cameras, LiDARs, Radars, and GNSS, outputting the ego-vehicle location and velocity, poses and motions of obstacles, states of traffic lights, and road maps. The planning system decides and plans the behavior of AVs with the information obtained from the perception module to ensure safe and comfortable driving. Lastly, the control system is responsible for converting the trajectory generated by the planning system into accurate execution commands to control the vehicles to reach specific targets in a finite time. These subsystems and their tasks are illustrated in Figure 8.

Considering the functions of different modules, many researchers have conducted in-depth studies into these three subsystems separately in the last decades and formed one category of AD algorithms: the modular pipeline. However, there is another mainstream in the AD community called the end-to-end driving method. In contrast to the modular pipeline method that designs and trains each module in separation, the end-to-end driving method believes all modules can be integrated without individual

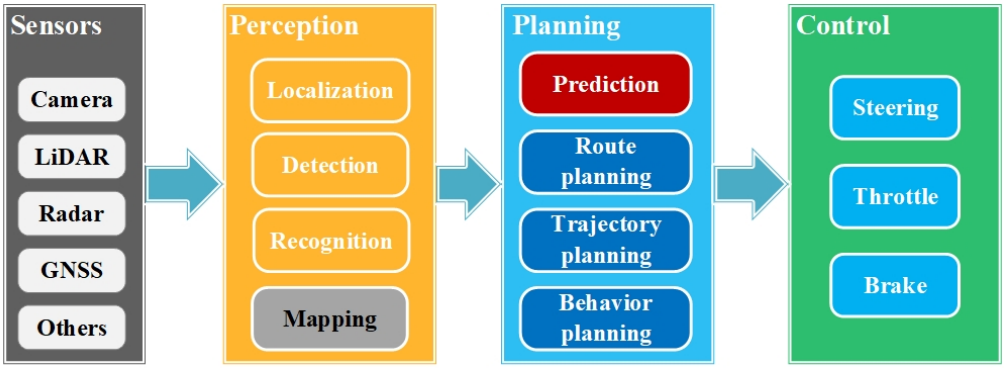


Figure 8: High-level autonomous driving stack. The mapping task in the perception module is optional since HD maps can be provided. The prediction task is painted red because it is the basis of other tasks.

design and be trained together, thus resulting in a more straightforward training and evaluation procedure for AVs. Since both methods are popular and provide state-of-the-art performance nowadays, this section will discuss them in the following, separately.

2.3.1 Classical modular pipeline

Environment perception provides a mediated representation of the scene by detecting a self-drive vehicle's nearby objects and road layouts. This system is mainly driven by computer vision, and many researchers have further separated the whole task into smaller pieces: detection, mapping, and localization.

The detection task involves two key components: lane detection and dynamic object detection. Typical lane detection algorithms take RGB images as inputs and output splines of detected lane markings [41, 42]. Huval et al. [43] detected the lanes on the highway environment using a regression method with a six-dimension vector, where the first four dimensions indicated the positions of two endpoints of a local lane segment, and the remaining two dimensions provided the relative depth of the endpoints toward the camera. Chen et al. [44] predicted a binary segmentation image to offer rich semantic information for lanes based on VGG architecture [45]. Similar to lane detection, dynamic object detection algorithms aim to detect cars and pedestrians, outputting bounding boxes to specify their positions, orientations, classes, and confidences from single or multi-modal sensors [46, 47, 48]. For example, Lang et al. [46] proposed their LiDAR-based PointPillars network in 2019 to extract vehicles and pedestrians using only 2D convolutional networks. Moreover, Liang et al. [49] fused LiDAR point cloud and RGB images to obtain a more accurate detection result in a 3D environment.

The mapping task either generates high-resolution maps offline or predicts road structure online. In offline settings, the maps are generated iteratively. By passing through the same scene several times with a specific data collection car, mapping algorithms gradually modify their wrong estimations, thus producing high-resolution maps step by step [50, 51]. Notably, one key advantage of offline mapping is that

the generated maps contain rich semantic information about the topology of roads, traffic lights, and crossroads in the global area, which can significantly increase the safety of trajectory planning. However, building high-resolution maps usually needs human experts' manual labeling, and maintaining them is a tedious task. On the contrary, online mapping directly predicts map elements around the ego-vehicle without explicitly outputting a final map [52, 53]. Homayounfar et al. [53] introduced a hierarchical recurrent neural network to predict the road area and lane boundaries in highway scenes. Similarly, Chen et al. [5] utilized image inputs to predict road affordances in which vehicles can drive safely.

The localization task aims to localize AVs in the global coordinate with centimeter-level precision. To achieve this goal, existing localization algorithms usually fuse different methods with two-level precision: global localization and relative localization. Global localization first uses classical algorithms, such as Particle filter, Kalman filter, and its variants [54, 55], to estimate the position of LVs with data obtained from IMU, GNSS, and wheel odometry, generating a rough result with errors in meters. Then, relative localization uses landmark features extracted from visual sensors [56, 57] to tune the result to the desired precision.

Motion planning uses information from the perception layer to plan future trajectories. As is shown in Figure 8, this module includes three different tasks: Route planning, Trajectory planning, and Behavior planning.

Route planning defines an optimal and safe path from the starting position to the target position with the help of road maps. Traditionally, Dijkstra and A* and their variants are applied to find such paths. However, they are incapable of pathfinding in large-scale maps because of the exhaustive search. Considering the high time complexity, recent algorithms [58] either represent the path as a function with a finite-dimensional parameter vector to accelerate the convergence to a local optimal or discretize the configuration space of AVs as a graph and incrementally sample and build a path toward the lowest cost direction.

Trajectory planning is a generalization form of route planning and has been shown to be more complicated than route planning. It involves reaching a fixed position at a particular time in dynamic environments. This requires the computation to consider the dynamics of AVs and the feasibility of actuation commands, as well as the effects of possible dynamic obstacles. Moreover, high computation complexity makes it impossible to obtain exact solutions, and therefore, numerical approximation methods directly in the time domain or in the configuration space with a time dimension [59] have become a popular choice.

Behavior planning determines how the AV interacts with the environment and other dynamic objects. By considering the planned route, traffic rules, and other vehicles, the behavior planning layer selects a short-term strategy to guide the self-drive car in a high-level maneuver, such as lane-following, overtaking, lane-changing, and braking.

Vehicle control module converts the planning into actual commands, such as steer, throttle, and brake, to transition the vehicle from its current state to another desired state. In order to precisely move the vehicle according to the planning results, the controller should satisfy two essential attributes: stability and robustness.

Therefore, closed-loop controllers with feedback and others that have been proven their stability and robustness using complete mathematical theorems are preferred. Among them, the PID controller and its variants are the most widely used due to their low cost and simple architecture. Furthermore, more advanced control methods, such as Sliding Mode Control and Fuzzy Logic Control [60] that can handle uncertainty about the vehicle state, are also utilized to control AVs. Due to their superior performance, the AVs are under a precise control process with a fast response frequency. Lastly, some optimal control methods, such as Linear Quadratic Regulator (LQR) and Model Predictive Control (MPC) [61], are playing a more important role in the field of controlling AVs.

2.3.2 End-to-end autonomous driving

End-to-end autonomous driving aims to directly map sensor signals to control commands for steering, throttle, and brake without separately fine-tuning any subsystems as it does in the modular pipeline. Its key advantage is that all parameters of a whole driving model can be optimized jointly with respect to a shared end goal using differentiable optimizers. Furthermore, based on their training methodology, these end-to-end driving algorithms can be divided into imitation learning-based ones and Deep-RL-based ones.

Imitation learning-based AD learns driving policies via supervised training from trajectories collected by human experts, aiming at reproducing experts' behavior in different traffic scenarios. Since Pomerleau [62] first pioneered this technique in the real world with Autonomous Land Vehicle In a Neural Network (ALVINN), many works have extended imitation learning to more complicated and more challenging environments, some of which are LBC [63], LAV [64], and InterFuser [65].

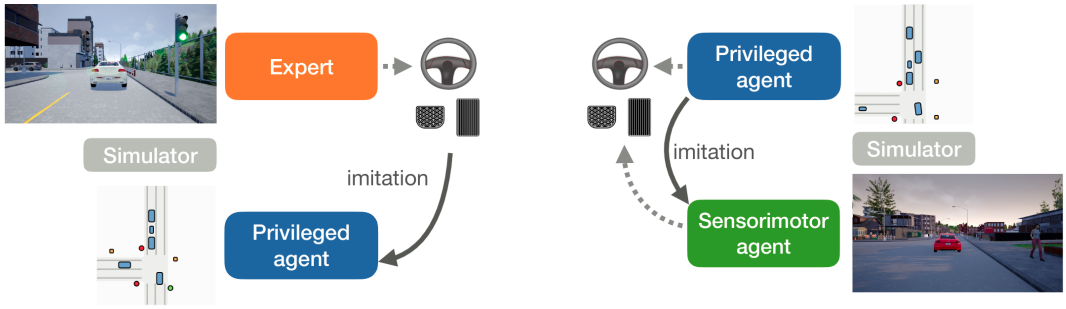


Figure 9: Overview of LBC [63]. The left shows the privileged agent directly learns to imitate expert demonstrations with access to ground-truth data. In this scene, the privileged agent is cheating. The right shows the vision-based agent learns to mimic the privileged agent without access to ground-truth data. The privileged agent acts as a teacher and provides high-capacity on-policy supervision. Thus, the vision-based agent does not cheat.

LBC: Chen et al. [63] decomposed the problem of perceiving and acting in

the world into two stages and proposed the vision-based driving model, Learning by Cheating (LBC). As illustrated in Figure 9, the first stage trains a privileged behavior cloning agent that can access all ground-truth data, such as the position of all traffic participants and the environment layout. The second stage trains another purely vision-based to mimic the privileged agent. Obviously, such an inverse acting-perceiving procedure has three advantages. First, the learning to see and learning to act problems are decomposed and can be solved easily. Second, the privileged agent accesses and operates on a compact bird's-eye view representation of the environment, enabling efficient data augmentation and fast generalization. Finally, the trained privileged agent is a white box, allowing query states not only visited in the original trajectories but also rolled out.

LAV: Chen et al. [64] introduced their driving model, Learning from All Vehicles (LAV), which utilized three RGB cameras and one 3D LiDAR in 2022. The basic idea is that driving logs experienced by other vehicles may contain safety-critical scenarios and thus can help the ego-vehicle increase the chance to see more situations. This is done through a vehicle-centered map representation that does not distinguish the ego-vehicle and other vehicles. In LAV, the sensor data is fused via point-painting [48] and each vehicle is extracted using CenterPoint [66] with PointPillars [46] backbone, followed by a rotated ROI extraction to obtain fixed-size representations. Once the features of vehicles are obtained, they are fed into motion planner equally to mimic their trajectories with supervised training. This is illustrated in Figure 10. An evaluation of several testing criteria presented by Chen et al. [64] shows that LAV can get great results in both training conditions and unknown environments.

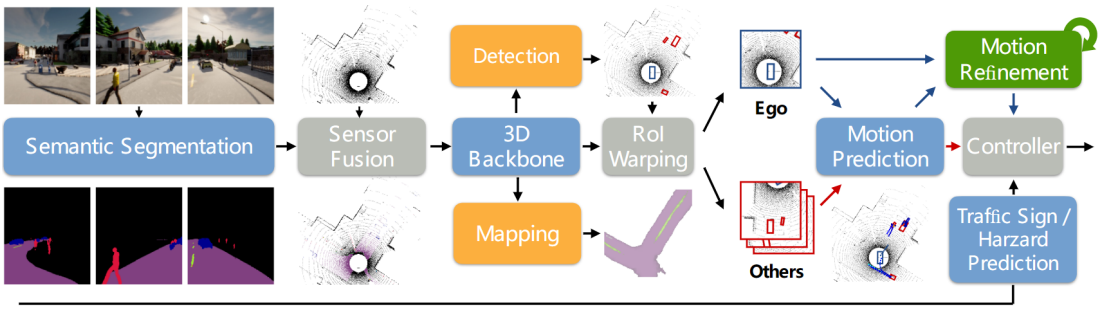


Figure 10: Overview of LAV [64]. The images and point cloud data are fused via point-painting [48] and then fed into PointPillars [46] 3D backbone to perform detection and mapping. Once vehicles are detected, ROI extraction is applied to fetch fixed-size features, followed by predicting their future trajectories separately. Lastly, refined trajectories are sent to two PID controllers to generate low-level control commands. Additionally, another convolutional network is responsible for hazard brake prediction directly from image inputs.

InterFuser: Shao et al. [65] considered the importance of comprehensive scene understanding and the interpretability of control command generation, proposing their Interpretable Sensor Fusion Transformer (InterFuser) driving model. As illustrated in Figure 11, the InterFuser is composed of three parts. The first part is a transformer encoder that takes multi-modal multi-view sensor inputs, outputting a compact

information integration. The second part is another transformer decoder with three parallel prediction headers to predict future waypoints, object density maps, and traffic rules, respectively. Furthermore, different queries and positional embeddings are also utilized to distinguish the outputs. The last part is a safety controller, which utilizes information from the decoder and historical trackers to forecast the motion of other vehicles, avoiding potential dangerous driving behaviors of the ego-vehicle. Notably, in order to ensure the safety controller can consider all participants in the recovered object density maps, InterFuser uses a two-threshold trick on the object density map to maintain uncertain objects and remove repeated detection: the lower threshold detects as many as objects, while the higher threshold removes repetition.

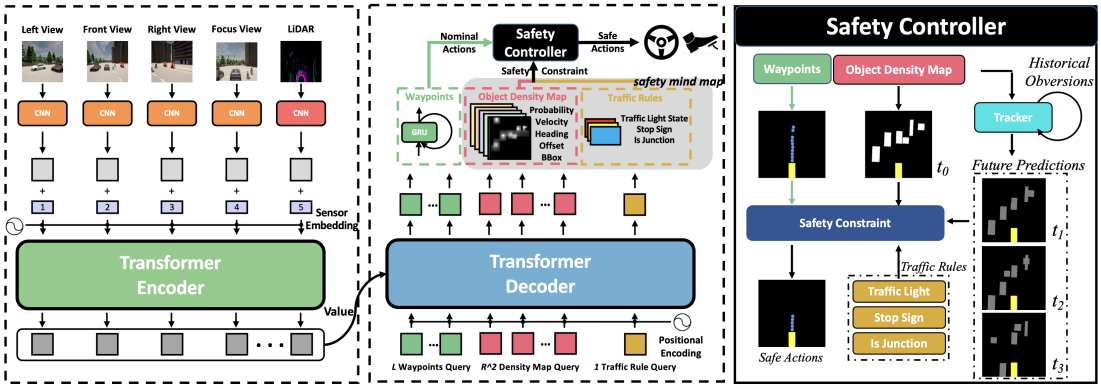


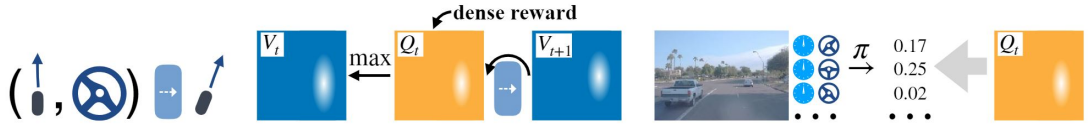
Figure 11: Overview of InterFuser [65]. The encoder fuses multi-modal sensor data into a compact representation. Next, the decoder predicts future waypoints, the object density map, and traffic rules with the help of queries. Lastly, the safety controller recovers the scene with the density map and predicts other vehicles' motion from the historical trackers, producing safe low-level control commands that can avoid the collision.

However, imitation learning-based driving algorithms suffer from heavily long-tail distributions and upper-bound capacity limitations. On the one side, since the behavior cloning agents try to mimic the experts, they are naturally limited to the expert actions, which means that the behavior cloning agents can at most perform as well as the expert demonstration. On the other side, obtaining expert demonstrations in every traffic scenario is an impossible task, and therefore the imitated policy has a strong confidence to example situations but generalizes poorly to unseen cases. Although excellent works like LAV [64] and InterFuser [65] have attempted to address these issues and achieved remarkable results, they have not been solved yet. Hence, many researchers have turned to Deep-RL-based methods.

Deep-RL is a combination of deep learning and RL, where deep neural networks are used to approximate value functions in the RL problem framework. Different from imitation learning that guides the agent to mimic an expert, Deep-RL teaches the agent to explore the environment itself and drive safely. Thus, the agent will not have a bias in terms of encountering different scenarios and can have a much higher capacity boundary. Nevertheless, as other RL problems need a massive amount of data to train the agent, Deep-RL-based autonomous car also requires a more

considerable amount of data than supervised imitation learning to converge, which is why many studies are conducted in simulation. Kendall [67] first introduced Deep-RL into AD in urban situations and showed great potential. Since then, a lot of impressive works have been completed, some of which are WOR [6], IA [24], and Latent DRL[68].

WOR: Chen et al. [6] proposed the model-based RL method World on Rails (WOR) in 2021. They assume the agent's action influences only its own state while the environment will not change, factorizing the world dynamics into a low-dimensional ego-vehicle model and a non-reactive world model. Furthermore, the transition of the world model is deterministic and purely depends on its previous state, meaning that one whole trajectory is confirmed once its initial state is known. About the driving policy, WOR first estimates the action-value function using the Bellman equation with a tabular dynamic programming evaluation. This is possible since the forward world model can roll out all trajectories based on selected actions. Then, the estimated action-value function is used to supervise the desired driving policy that takes a single image as input. This pipeline is illustrated in Figure 12. As Chekroun [7] argued, although the assumption of a static world was not realistic both in simulation and the real world, it showed a way to decompose a complex urban driving task into an easy deterministic environment model learning task and a more straightforward policy search task, and such decomposition could achieve state-of-the-art performance.



(a) Forward model. (b) Bellman update. (c) Distillation.

Figure 12: Overview of World on Rails [6]. First, the forward model is learned based on all trajectories and the world. Then, the action-value function is updated using dynamic programming of the Bellman equation. Finally, the policy distillation is performed for a single image.

IA: Toromanoff et al. [24] considered the difficulty of training a large Deep-RL network with raw image inputs, introducing the Implicit Affordances (IAs) method in dealing with urban driving problems. They argue, for urban driving, the difficulty of applying Deep-RL approaches comes from two aspects: the convergence problem of training a larger network with larger images and the limitation of replay buffer size. Therefore, it is better to encode the raw images into a more compact representation to serve as the RL state. To do so, they split the driving system into two subsystems. First, a visual encoder is trained on some auxiliary tasks, such as semantic segmentation, traffic light detection, and road classification, to output some affordance features of the environment. Then, the encoder is frozen, and a Deep-RL framework takes the output features of this encoder to produce an optimal driving policy. The experiment provided by Toromanoff [24] shows that their method takes 20 times less memory and converges 5 times faster rather than inputting raw images.

Latent DRL: Chen et al. [68] further extended the idea of encoding raw inputs to reduce the sample complexity. Their work divides the urban driving task into two smaller but highly correlated subtasks: environment modeling and driving policy search. Due to their sequential observation property, both subtasks can be formulated as a probabilistic graphical model with latent variables, as shown in Figure 13. For the environment modeling, they use a sequential variational auto-encoder to embed the observations that contain one RGB image and one LiDAR image, and produce an immediate bird's eye view mask for reasoning interpretation and driving policy search, where the generated mask includes road maps, routing, other road participants, and the position of the ego-vehicle. For driving policy search, the SAC [34] algorithm is selected because it is an extension of entropy maximization. The comparison result to other driving baselines, such as DQN [32] and DDPG [29], shows that the Latent DRL method has great potential for complex urban driving.

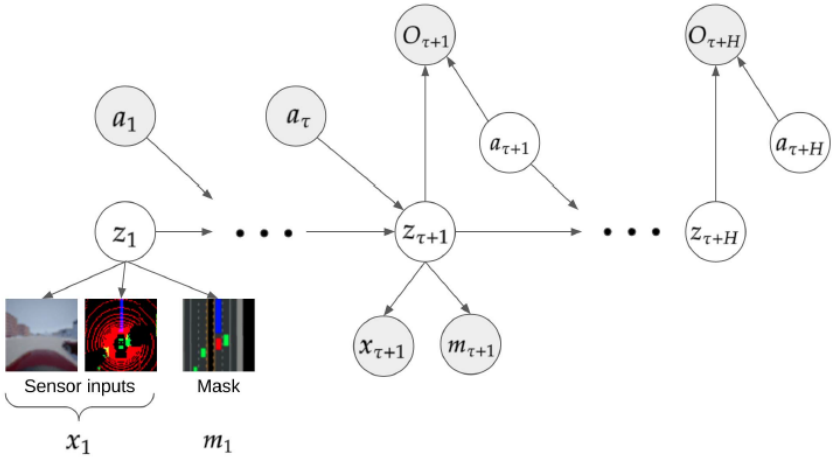


Figure 13: A probabilistic graphical model of Latent DRL [68]. The latent variable z contains the sensor input x and can be decoded into the bird's eye view mask m.

2.4 Traffic scenario simulation

Traffic scenario simulation aims to construct diverse traffic situations using simulators, reproducing some scenes that are dangerous or rare in the real world, such as vehicles running a red light, falling cargo from front driving tracks, or the sudden emergence of pedestrians from roadsides. Typically, its development arises from two reasons. First, simulation is cheap and can cover all potential situations. According to a recent survey [69], one difficult challenge that hinders the development of AD is the requirement for a considerable amount of driving data. Because it is extremely costly to drive an expensive test car with well-configured sensors to collect drive logs in the real world, only some big companies with a fortune can afford it and obtain enough data, which significantly limits the AD community from exploring more practical driving algorithms. However, with the help of simulation, any research group can build diverse traffic scenes to collect driving logs without buying a real

car, thus breaking through the harsh data threshold. Second, simulation is easy to evaluate AD algorithms in some edge cases. Since AD is a safety-critical application, it must be fully guaranteed that it can handle all possible situations encountered safely. However, testing a developed AD algorithm in the real world is a dangerous task, and it is often impossible to validate the AV in all traffic scenarios. Therefore, a new method for safe and comprehensive testing is to design rich traffic simulation scenarios in simulation. By doing this, the performance of AV can be analyzed in any critical case without being deployed in practice, thus not only improving safety during tests but also saving costs.

A lot of research has been proposed on traffic scenario simulation [10, 11, 12, 70, 71]. For example, Abeysirigoonawardena et al. [11] propose an automatic adversarial driving scenario generation method for testing AVs. Their work involves training adversarial cars and pedestrians using Bayesian optimization and modeling the unknown cumulative performance of the test agent as a Gaussian process. However, their work exposes some drawbacks. First, the behavior of adversarial pedestrians is restricted. In order to avoid wasteful exploration of adversarial agents, they manually define reachable areas of pedestrians and use a graph to describe those areas, in which nodes are possible targets that pedestrians can reach, and edges are trajectories that pedestrians must follow. Such human specification renders pedestrian behavior simple and easy to predict. Second, the generated scenario is limited to intersections. This is because the adversarial car is not allowed to cross lanes, resulting in a situation that it can only try to collide with the test vehicle when it changes its directions at intersections.

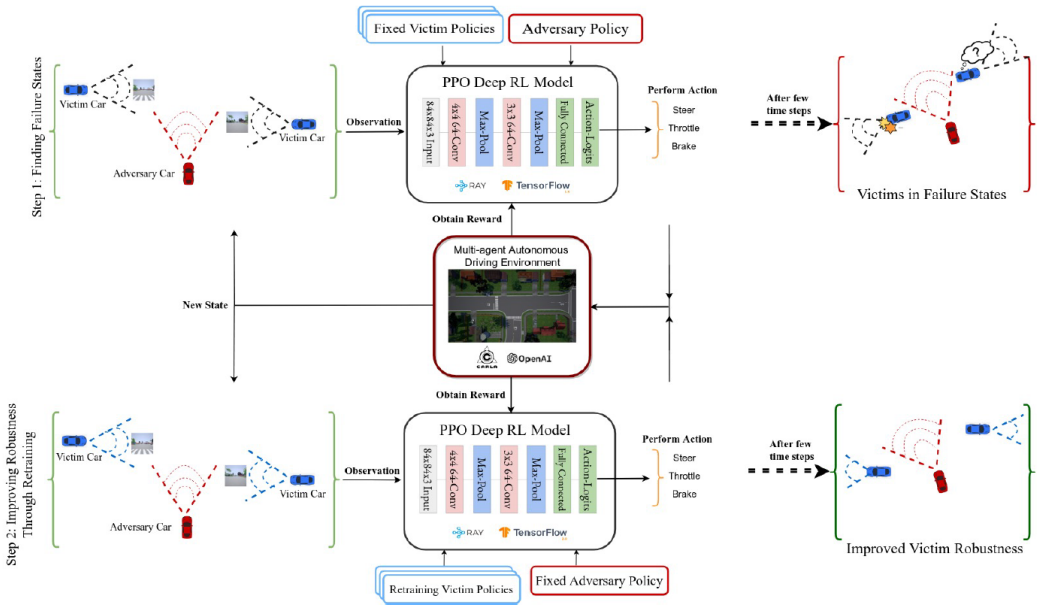


Figure 14: The framework of MAD-ARL [10]. Each agent uses a PPO-based model to generate driving policies. The top row is the first step to finding failure scenarios, while the bottom row is the second step to retraining victims based on those failures.

Another similar work proposed by Sharif et al. [10] introduces a Multi-Agent Driving with Adversarial Reinforcement Learning (MAD-ARL) framework to not only test AVs for finding decision errors but also improve the performance of AVs to those errors. As is shown in Figure 14, this framework consists of two steps. In the first step, one adversarial car, called the adversary, is trained with the PPO [33] algorithm to create natural and adversarial observations to degrade the performance of the AV under test. Moreover, the corresponding adversarial policy is guided either to maximize the collision rate with the test AV, or to maximize the offroad steering rate. In the second step, the test AV, also called the victim, is retrained with the adversarial car to defend against those attacks by freezing the adversarial policy. Similar to the training procedure of the adversary, the victim also uses the PPO [33] algorithm to improve its driving policy. The experiment provided by Sharif et al. [10] shows that the MAD-ARL framework is beneficial for detecting driving errors and improving against those errors in terms of collisions and offroad steering.

More recent works have further studied the behavior of pedestrians. Karunakaran et al. [12] train pedestrians to cross roads through the crosswalk when the test vehicle is approaching. However, the pedestrian trajectory is scripted, constraining the proposed method from being generalized to other environments. Inspired by existing various pedestrian models [71, 72, 73], Priisalu et al. [70] propose to train a model to learn to place pedestrians such that they are prone to collide with a test AV rather than to control their actions or trajectories. Their method consists of three parts. The first two parts are the AV algorithm and the pedestrian behavior model, which are both frozen and will not be updated. The last part is the learnable adversarial test synthesizer that learns to initialize the pedestrian in a proper place according to the selected pedestrian behavior model, occlusions, and scene semantics. This is illustrated in Figure 15. Compared to other works [12, 71], this method offers two main advantages. Firstly, the adversarial test synthesizer is independent of the pedestrian behavior model, and therefore it is free to select different pedestrian models according to the situation. Secondly, the generated traffic scenario is more challenging and realistic since the pedestrian is initialized in occluded spaces, leading the test vehicle difficult to perceive it in advance.

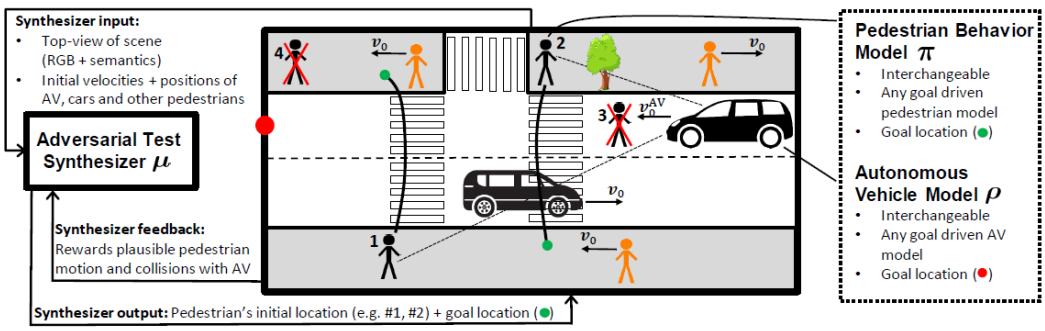


Figure 15: Overview of the safety-critical scenario generating model with pedestrians [70]. The adversarial test synthesizer learns a distribution μ to select the initialization position of a pedestrian with a given behavior model π.

3 Methods

This chapter describes the proposed method used in this thesis. First, the overall framework is introduced. Second, the methodology for designing the suicidal pedestrian is described, including input and output representations, reward function formulation, and policy search mechanism. Third, the evaluation metrics are defined to verify the effectiveness of the designed pedestrian agent.

3.1 System description

This thesis addresses the problem of generating a pedestrian-related safety-critical traffic scenario that can be used to test the capacity of AVs. Specifically, the proposed traffic scene contains two agents: the pedestrian and the AV. The pedestrian observes the location and motion of the AV and tries to hit the car from an unpredictable direction, causing the AV failure. On the other hand, The AV is controlled by some state-of-the-art driving algorithms, which take sensor observations as inputs and produce low-level commands to drive the car without causing any collision. Furthermore, no prior assumptions about the vehicle model have been made. Hence, theoretically, any AD algorithm can be used to cooperate with the suicidal pedestrian. This is illustrated in Figure 16.

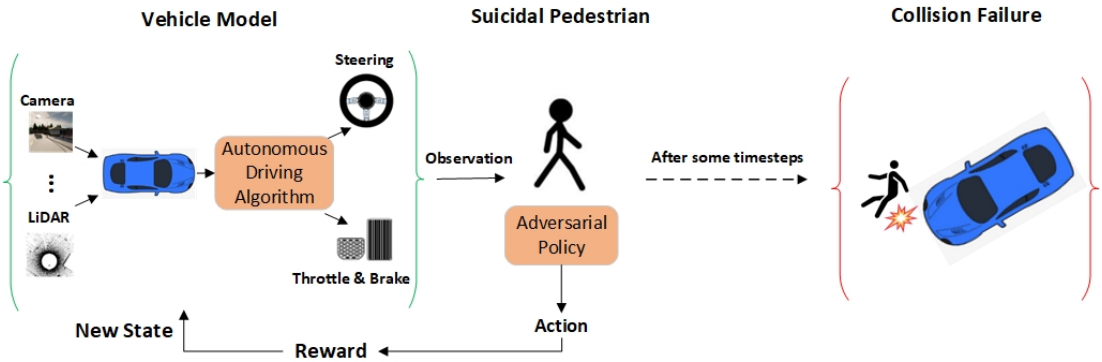


Figure 16: Overview of the proposed suicidal pedestrian traffic scenario. The AV is driven by some AD algorithms with fixed parameters, while the pedestrian performs actions to hit the car with some learnable policies.

In this framework, the AV model and the suicidal pedestrian play an indirectly constrained minimax game. This means the vehicle will try to avoid collisions, while the pedestrian seeks the opposite situation. However, finding solutions to this indirectly constrained minimax game could be either very trivial or totally impossible, depending on the situation. For example, suppose the suicidal pedestrian is initialized arbitrarily close to the front of the AV. In this case, it can easily lead to a collision whether the car drives forward or stands. Otherwise, if the suicidal pedestrian is initialized infinitely far from the AV, it is impossible to lead to crashes with finite timesteps. Therefore, in order to avoid such unpleasant cases, some constraints about

the initial distance and direction between the pedestrian and the vehicle are needed, and they will be discussed later in this section.

3.1.1 Walking as a Markov Decision Process

According to descriptions mentioned above, one of the core challenges is to model the suicidal behavior, and it is natural to view this challenge as an RL task with two agents, in which one agent is the AV under test, and the other agent is an adversarial suicidal pedestrian agent whose goal is trying to exploit the weakness of the test AV. However, since training the AV is out of the scope of this thesis work, this task can be further simplified to a single-agent RL problem with an NPC vehicle.

Therefore, a one-player MDP framework is suitable to formally describe this RL problem. Denote the corresponding MDP framework as $M_\alpha = (S_\alpha, P_\alpha, R_\alpha, A_\alpha, \gamma)$, where S_α is the state set, A_α is the action set, R_α is the reward function, $\gamma \in [0, 1]$ is the discount factor, and P_α is the transition dynamics. At timestep $t \in \{0, T-1\}$, the pedestrian walks by taking actions $a_\alpha^{(t)}$ from its policy $\pi_\alpha(\cdot|s_\alpha^{(t)})$, followed by receiving a reward. Hence, the goal of the suicidal pedestrian is to learn a policy that maximizes the sum of discounted rewards:

$$\pi_\alpha = \sum_{t=0}^{T} \gamma^t R_\alpha(s^{(t)}, a_\alpha^{(t)}, s^{(t+1)}), \tag{6}$$

where $s^{(t+1)} \sim P_\alpha(s^{(t)}, a_\alpha^{(t)})$ is the next state given the transition probability. Notably, the MDP dynamics model P_α is unknown because no assumption about the state transition has been made.

However, even though the MDP framework is established, the problem is still unsolvable since the reward function, the action set, and the state set are not yet defined. Thus, in order to fix the remaining issue, some dictated design is needed, which will be discussed in detail in Section 3.2.

3.1.2 Autonomous vehicle model

As illustrated in Figure 16, the AV model is the other important component of this framework. However, unlike the pedestrian model that must be designed, the AV model is only intentionally simple to demonstrate the structure empirically since this thesis aims to prove that the suicidal pedestrian behavior model can be successfully used to evaluate vehicle capacity in some pedestrian-related scenarios. Thus, existing AD algorithms can be directly utilized in this framework, and no further assumptions or constraints about the AV are required. Nevertheless, it is still beneficial to illustrate how the AV works in this framework from a general perspective without knowing its technical details.

Let x_ρ be the inner state vector of the AV, calculated based on data obtained from sensors such as GNSS, IMU, and odometry. This state vector contains its position, orientation, and velocity in a global reference coordinate. Let s_ρ be the state vector of pedestrians, other vehicles, and static obstacles in this simulated traffic scenario. Similar to the AV state vector, each element in s_ρ includes the position and

orientation of the corresponding object. Assume the policy generated by a particular driving algorithm is $\pi_\rho \sim (\cdot|o_\rho)$, which depends on the observation of the vehicle. The actions of the vehicle a_ρ are determined by this policy. The observation model of the AV is denoted as $o_\rho \sim h(o|x_\rho, s_\rho)$, which is a conditional distribution of the traffic scenario scene viewed from the AV perspective. Since this thesis uses CARLA [13] as the simulation platform, the observation can include camera images, LiDAR point clouds, radar data, and other relevant sensor readings.

Therefore, the AV model can be modeled and integrated into the proposed framework by using the following equations:

$$\begin{aligned} o_\rho^{(t)} &\sim h(o|x_\rho^{(t)}, s_\rho^{(t)}) \\ a_\rho^{(t)} &\sim \pi_\rho(a_\rho|o_\rho^{(t)}) \\ x_\rho^{(t+1)} &\sim p(x_\rho|x_\rho^{(t)}, a_\rho^{(t)}) \\ s_\rho^{(t+1)} &\sim q(s_\rho|s_\rho^{(t)}) \end{aligned} \tag{7}$$

where $p(x_\rho|x_\rho^{(t)}, a_\rho^{(t)})$ is the transition probability of the AV state, which is fully determined by the current AV state and the performed action. The $q(s_\rho|s_\rho^{(t)})$ is the transition probability of other objects in the scene, except the AV. This transition probability is controlled by the simulation and is unknown.

3.1.3 Pedestrian initialization

As discussed at the beginning of this section, the initial location of the suicidal pedestrian agent plays an important role in whether the pedestrian can successfully hit the vehicle in finite timesteps. On the one hand, if the pedestrian and the vehicle are too close to each other, the vehicle cannot detect the pedestrian in time and perform actions to avoid collisions, resulting in crashes always happening. Oppositely, if the pedestrian is far from the vehicle, it becomes highly possible that the pedestrian cannot catch the car, and therefore no collision happens.

Regarding these problems, the initial location of the pedestrian is restricted to an area close to the vehicle varying from 7 meters to 30 meters. The motivation behind this distance variety is that within a distance increasing, the pedestrian can show more complex behaviors when approaching the car, thus enhancing the diversity of the simulated traffic scene.

Furthermore, to avoid the vehicle driving away from the pedestrian, this thesis intentionally initializes the pedestrian in the front areas of the AV. Specifically, this area covers a sector ranging from -60 degrees to 60 degrees based on the forward direction of the vehicle.

Finally, small random offsets sampled from a Uniform distribution are added to the initial location of the pedestrian to ensure that the pedestrian is initialized at walkable areas rather than static obstacles such as flowerbeds or stone piers.

The detailed process of initializing the pedestrian is illustrated in Algorithm 1.

Algorithm 1 Suicidal Pedestrian Initialization

Input: Pose of the test vehicle $(x_{vel}, y_{vel}, \theta_{vel})$

Initialization: $Flag \leftarrow False$
while $Flag = False$ **do**
  Randomly select a spawn point p from the map
  Calculate the distance d and relative direction α to the vehicle pose
  if $7m \leq d \leq 30m$ **and** $-60° \leq \alpha \leq 60°$ **then**
    for $i = 1, .., 10$ **do**
      Generate an offset x_{off} from a Uniform distribution
      Update $p \leftarrow p + x_{off}$
      Spawn the pedestrian at p. If succeed, set $Flag \leftarrow True$ and end algorithm
    end for
  end if
end while

Output: Initialization localization p of the pedestrian

3.2 Suicidal pedestrian design

In this section, the detailed methods of designing the suicidal pedestrian agent will be described. Since this design problem has been formulated as an MDP in Section 3.1, the focus of this section is narrowed to the definition of reward functions, agent action space, agent state space, and policy search method.

3.2.1 Input and output representations

Input representations: the input state of the suicidal pedestrian agent represents how this agent observes the environment. Typical inputs include RGB images from different perspectives. However, such image input is highly dimensional and contains much useless information, which requires significant effort, such as a series of convolutions and Variational Autoencoders (VAEs) to extract critical components. Moreover, except for the suicidal pedestrian, the proposed traffic scenario contains only one dynamic vehicle and some other static environmental objects. This means the pedestrian can finish the collision task without being affected by other dynamic objects if the positions, orientations, and velocities of the target vehicle and the pedestrian are known. Therefore, considering these factors, this thesis will define the input state of the suicidal pedestrian with a finite-dimensional vector rather than the RGB image.

Inspired by the work from Priisalu et al. [70] and the property of the simulation, i.e., it is easy to access all information about each object in the scene, one simple but effective way to define this state vector is to contain relative distance, direction, and velocity information between the vehicle and the pedestrian and represent them in the pedestrian coordinate. However, since all information is represented based on the world coordinate, some transformation tricks are required to calculate these desired elements. Take the relative distance and relative direction as an example. Given the vehicle position vector ${}^{W}P_{car}$ and the vehicle orientation vector ${}^{W}\theta_{car}$ in the world

coordinate W, as well as the pedestrian position and orientation vectors ${}^{W}P_{walker}$ and ${}^{W}\theta_{walker}$, the transformation matrix ${}^{W}_{P}T$ and the rotation matrix ${}^{W}_{P}R$ of the pedestrian coordinate P based on the world coordinate W can be firstly calculated. Then, treat the pedestrian coordinate P as the base and both the position and orientation of the vehicle can be represented in the pedestrian coordinate P using the following transformation equations:

$$\begin{cases} {}^{P}P_{car} = ({}^{W}_{P}T)^{-1} \cdot {}^{W}P_{car} \\ {}^{P}\theta_{car} = ({}^{W}_{P}R)^{-1} \cdot {}^{W}\theta_{car} \end{cases} \tag{8}$$

where ${}^{P}P_{car}$ and ${}^{P}\theta_{car}$ are the position and orientation of the target vehicle represented in the pedestrian coordinate P. Furthermore, based on the transformed vehicle position ${}^{P}P_{car}$ and orientation ${}^{P}\theta_{car}$, the required relative distance and relative direction are obtained:

$$\begin{cases} \alpha = \arctan 2\Big({}^{P}P^{(y)}_{car}, {}^{P}P^{(x)}_{car}\Big) \\ d = ||\,{}^{P}P_{car}||_2 \end{cases} \tag{9}$$

where ${}^{P}P^{(x)}_{car}$ is the element in x-axis, and ${}^{P}P^{(y)}_{car}$ is the element in y-axis. A similar operation can be applied to the velocity. Suppose the scalar velocities of the vehicle and the pedestrian in the world coordinate is ${}^{W}S_{car}$ and ${}^{W}S_{walker}$ respectively. The transformed velocity vectors represented in the pedestrian coordinate are calculated by:

$$\begin{cases} {}^{P}S_{car} = {}^{W}S_{car} \cdot \Big(({}^{W}_{P}R)^{-1} \cdot {}^{W}\theta_{car}\Big) \\ {}^{P}S_{walker} = {}^{W}S_{walker} \cdot \Big(({}^{W}_{P}R)^{-1} \cdot {}^{W}\theta_{walker}\Big) \end{cases} \tag{10}$$

Then the relative velocity and relative velocity direction can be obtained:

$$\begin{cases} \beta = \arctan 2\Big({}^{P}S^{(y)}_{car} - {}^{P}S^{(y)}_{walker}, {}^{P}S^{(x)}_{car} - {}^{P}S^{(x)}_{walker}\Big) \\ v = ||\,{}^{P}S_{car} - {}^{P}S_{walker}||_2 \end{cases} \tag{11}$$

Therefore, the final input state vector of the suicidal pedestrian agent can be defined as:

$$s = [\alpha, d, \beta, v], \tag{12}$$

where α, d, β, v are calculated from Equations 9 and 11.

Output representations: the output of the pedestrian agent is actions to be performed. Considering the allowed control commands for pedestrians provided by CARLA, this thesis designs an output vector with two elements:

$$a = [\theta_{target}, v_{target}], \tag{13}$$

where θ_{target} is the forward direction angle that specifies which direction the pedestrian will walk forward to, and v_{target} is the scalar velocity that defines the speed of the pedestrian.

Notably, since the input state is represented in the pedestrian coordinate P, the output action is also represented in this coordinate. However, the suicidal pedestrian agent walks in the environment defined in the world coordinate W. Hence, the output action must be transformed from the pedestrian coordinate to the world coordinate to avoid potential errors. Similar to the input state operations, the transformed direction vector represented in the world coordinate can be calculated by:

$$W_{direction} = {}^{W}\theta_{walker} \cdot \cos\theta_{target} + {}^{W}\theta_{walker} \cdot \sin\theta_{target}. \tag{14}$$

3.2.2 Reward functions

The reward function plays an essential role in training the pedestrian policy. In order to illustrate that a pedestrian with a velocity-proportional and collision-part-different reward function can lead to creating a more complex and unpredictable adversarial behavior than the collision-focus pedestrian agent, this thesis defines two different types of reward functions: the constant reward function and the combinational reward function.

The constant reward function, denoted as $R_{constant}$, is inspired by the fact that the goal of the pedestrian agent is to create collision cases, rendering AV failures. Therefore, one simple but effective way is to design a reward function that can maximize the collision rate without considering anything else, i.e., the pedestrian is rewarded only when it hits the vehicle. To this end, $R_{constant}$ can be formulated as:

$$R_{constant} = \begin{cases} 1, & \text{if the pedestrian hits the vehicle} \\ 0, & \text{otherwise} \end{cases} \tag{15}$$

On the other hand, the combinational reward function R_{comb} considers more conditions when maximizing the collision rate. First, the pedestrian should behave more positively to hitting the vehicle. This means the pedestrian needs to collide with a vehicle driving at high speed as much as possible. Second, the pedestrian should perform more complex and unpredictable behaviors to cheat the vehicle. Rather than hitting the car from the rear, it is preferred to create collisions from some occluded space, such as the intersection of the front and the side parts of the vehicle. Hence, R_{comb} is formulated as:

$$R_{comb} = \begin{cases} \max(3, 1.5 \cdot v), & \text{if pedestrian hits the front part of vehicle} \\ \max(1, 0.5 \cdot v), & \text{if pedestrian hits other parts of vehicle} \\ 0, & \text{otherwise} \end{cases} \tag{16}$$

where v is the velocity of the vehicle when the collision happens.

As described in Equation 16, different rewards are designed for different collision parts, and the vehicle velocity is considered in the reward function. For simplification, this thesis only distinguishes the front collision part of the vehicle and the other collision part. However, in order to ensure collision diversity and complexity, the front collision part is intentionally designed to contain some areas from the sides, which is illustrated in Figure 17.

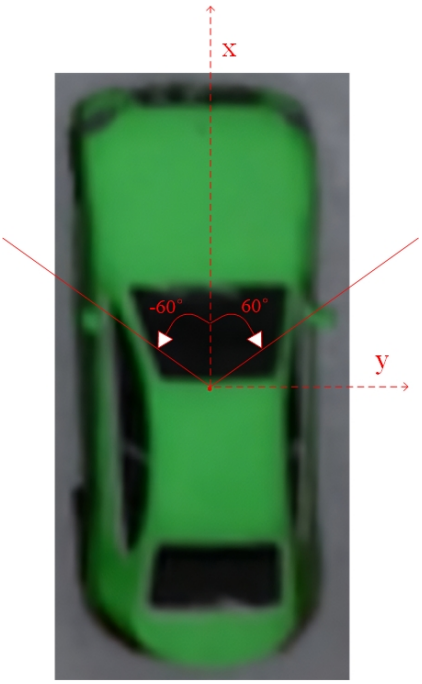


Figure 17: Visualization of collision area of the vehicle. The front collision part includes the central front and some side areas close to the front. Such design is based on the fact that human drivers can see both the front and parts of their sides when driving.

3.2.3 Policy optimization

This thesis uses the PPO [33] algorithm to learn a walking policy for the designed suicidal pedestrian. As described by Schulman et al. [33], the PPO performs online learning in each training episode and outputs a probability distribution function to represent the strategy. Furthermore, the PPO is a policy-based RL algorithm. It uses the policy gradient principle to update its current strategy and utilizes the received reward to weaken or enhance the probability of chosen behaviors so that behaviors that bring higher returns are more likely to be selected. Moreover, the PPO loss consists of three parts: the clipped policy loss, the value loss, and the regularization entropy loss.

The clipped policy loss is calculated based on the policy gradient principle and is used to update the policy parameters. This loss is defined as:

$$L(\theta) = -E\Big[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\Big] \tag{17}$$

where $\hat{A}_t$ is the estimated advantage function, ϵ is a clip factor, and $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio under the new and old policies.

The value loss is a squared error that is responsible for updating the value state network parameterized by ω, which is defined as:

$$L(\omega) = E\Big[(r_t + \gamma V(s_{t+1}) - V(s_t))^2\Big] \tag{18}$$

where r_t is the received reward, γ is the discount factor, and $V(s)$ is the state value function.

Lastly, the entropy loss $ENT(\pi_\theta)$ serves as a regularization term, ensuring that the algorithm performs sufficient exploration. Therefore, the total loss function of PPO can be written as:

$$LOSS = c_1 L(\theta) + c_1 L(\omega) - c_3 ENT(\pi_\theta) \tag{19}$$

where c_1, c_2, c_3 are loss coefficients.

However, since implementing the PPO algorithm is not in the scope of this thesis, it is straightforward to use some open-source implementations to train the suicidal pedestrian. Among those already-made versions, the PPO provided by the stable-baselines3 [74] is selected due to its high-quality implementation, clean interface, excellent performance, and reader-friendly documents.

Finally, the detailed hyperparameters used to train the suicidal pedestrian agent are illustrated in Table 1.

Hyperparameter		**Value**
	Number of total training steps	70000
	Number of epochs when optimizing the surrogate loss	10
	Number of steps to run for each environment per update	150
	Batch size	64
λ	Learning rate for actor and critic networks	$3 \cdot 10^{-4}$
γ	Discount factor for calculating the return	0.98
λ_{gae}	Bias-variance trade-off factor for Generalized Advantage Estimator	0.95
λ_{clip}	Objective clipping value	0.2
	Value loss coefficient	0.5
	Entropy regularization coefficient	0.01

Table 1: The PPO algorithm parameters used for training the suicidal pedestrian.

3.3 Evaluation metrics

Considering the goal of the pedestrian is to hit the AV to cause car decision failures, it is natural to use the collision rate to evaluate the performance of the designed suicidal pedestrian. However, since the behavior complexity of the pedestrian is also of interest, other metrics focusing on different aspects are needed. Therefore, this thesis designs the following metrics to evaluate the performance of the suicidal pedestrian:

- C_V: collision rate with the target vehicle
- C_R: collision rate with the running target vehicle
- C_F: front part collision rate with the target vehicle
- C_S: side part collision rate with the target vehicle

Notably, different from C_V and C_R which are absolute collision rates calculated based on all test episodes, C_F and C_S are relative collision rates calculated over episodes that the collision happens. Furthermore, the front collision area covers some side parts of the vehicle, which is shown in Figure 17.

4 Experiments

In this chapter, the experiments for simulation environment setup, suicidal pedestrian training and testing, as well as state-of-the-art AD algorithm evaluation using the trained pedestrian agent are described, followed by discussions about their results. First, the environment simulation setup is discussed. Second, the procedure for training the suicidal pedestrian agent is illustrated. Third, an effectiveness evaluation for the pedestrian agent is performed, and the corresponding results are discussed. Finally, two state-of-the-art AD algorithms are tested with the adversarial suicidal pedestrian, exposing their weaknesses and pointing out possible improvements.

4.1 Simulation setup

As discussed in previous chapters, this thesis uses the CARLA [13] urban driving simulation platform to train and validate the designed suicidal pedestrian, as well as evaluate some state-of-the-art AD algorithms using the created traffic scenario with the participation of the pedestrian agent. Furthermore, in order to utilize the PPO [33] algorithm provided by the stable-baselines3 [74] toolkit, this thesis also uses the OpenAI Gym [75] framework, a standard toolkit for developing and comparing RL algorithms using a series of virtual environments, to wrap up the designed suicidal pedestrian.

For building the RL simulation framework, many hyperparameters need to be specified, including the episode length, simulation speed, and control command frequency. In this thesis, the CARLA simulator is designed to run at a speed of 20 timesteps per second. Furthermore, each episode is set to terminate at most after 600 timesteps. This means each episode equals a 30-second simulation in the real world if no other termination condition, i.e., the suicidal pedestrian collides with the AV, is triggered earlier. Moreover, considering the pedestrian and vehicle speed difference, this thesis utilizes different control frequencies to navigate them. For the pedestrian, each control command is repeated for 20 timesteps, which is equal to a 1-second simulation in the real world. The motivation behind this action repeat is that pedestrians occasionally change their directions and speed during walking, and such repeat will not cause serious sequences. On the other hand, for the vehicle, the control command is updated every timestep to avoid accidents caused by delayed controls. The detailed hyperparameters are illustrated in Table 2.

Additionally, all experiments in this thesis are developed in Python and performed with a computer with Intel i7-11800H central processing unit @ 2.3 Gz $\times$ 8, 16 GB RAM, and NVIDIA GeForce RTX 3060 GPU.

4.2 Pedestrian agent training

This thesis uses Town 2 provided by CARLA to build up the traffic scenario and to train the suicidal pedestrian agent. This town contains some T-intersections and 2-lane roads. The motivation behind this town choice comes from two aspects. The first is that T-intersections can provide more complex traffic scenarios. The other

Hyperparameter		Value
f	Maximal simulation time per frame for Carla	0.05 s
L	Maximal episode length	600
d_{min}	Minimal initial distance between the vehicle and the pedestrian	7 m
d_{max}	Maximal initial distance between the vehicle and the pedestrian	30 m
α_{min}	Clockwise minimal initial relative direction from the vehicle to the pedestrian	-60°
α_{max}	Clockwise maximal initial relative direction from the vehicle to the pedestrian	60°
r_{walker}	Action repeat for the pedestrian	20
r_{car}	Action repeat for the vehicle	1

Table 2: Parameters used for building the suicidal pedestrian RL environment.

motivation is that 2-lane roads are the main road structure in residential areas where pedestrians are more likely to appear.

As for the AV agent, this thesis uses a rule-based expert during the training procedure. The reasons for this choice are the following. Firstly, this expert, called the CARLA behavior agent, is fast in action inferring and contains the sufficient capacity to deal with scenarios in the training process. Secondly, this expert agent can access privileged information in the CARLA simulator, thus simplifying and accelerating the training task. Thirdly, two state-of-the-art AD algorithms to be tested with the trained pedestrian in later sections strongly connect with the CARLA behavior agent. Due to these connections, it is enough to use a pedestrian trained with the CARLA expert to test their capacities and find failures. However, to force the vehicle agent to behave more aggressively, i.e., the vehicle is more prone to collide with the pedestrian at high speed, its maximal driving speed is set faster, and the minimal hard brake distance is decreased. Moreover, other parameters are also modified, which are shown in Table 3.

Hyperparameter		Value
v_{max}	Maximal velocity of the vehicle	30 km/h
$d_{decrease}$	Velocity decrease for vehicle following	8 km/h
d_{limit}	Minimal velocity decrease under safety time distance	2 km/h
t_{safe}	Safety time for vehicle following	3 s
d_{brake}	Minimal distance to perform hard brakes	4 m
$d_{threshold}$	Minimal distance to be alerted of possible collisions	8 m

Table 3: The parameters used for CARLA behavior agent.

Finally, instead of randomly initializing the vehicle over the whole town, this

thesis selects four positions and orientations to spawn the vehicle for the purpose of reproducing the training procedure. These starting locations and orientations are:

- location 1: [104.3, 241.3, 0.5], orientation 1: [0, 0, 0]
- location 2: [88.8, 302.6, 0.5], orientation 2: [0, $-\pi$, 0]
- location 3: [190.0, 293.5, 0.5], orientation 3: [0, $\frac{\pi}{2}$, 0]
- location 4: [193.8, 218.8, 0.5], orientation 4: [0,$-\frac{\pi}{2}$ 0]

where components of the location are $[x, y, z]$, and components of the orientation are $[pitch, yaw, roll]$. All of these locations and orientations are represented in the world coordinate. On the contrary, the pedestrian agent is initialized nearby the vehicle, which is described in detail in Algorithm 1.

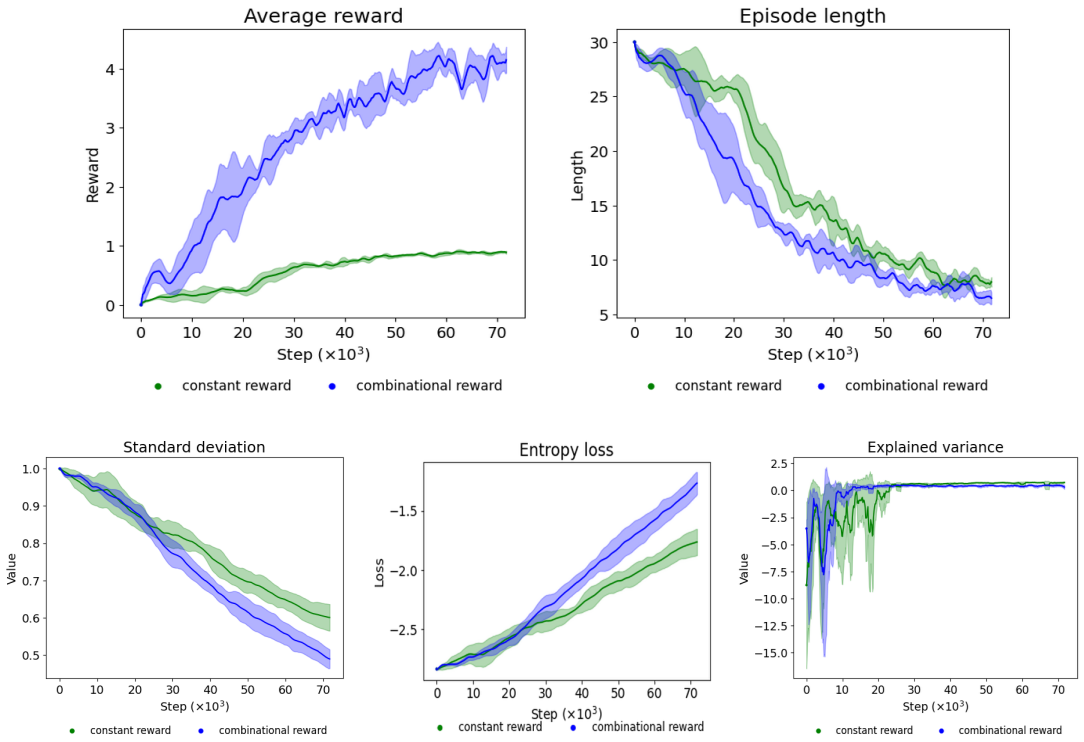


Figure 18: Training performance of the suicidal pedestrian against the CARLA behavior agent using two different reward functions. Each policy is trained three times with the same conditions. The solid line represents the mean average over three trains, and the light-colored area represents the standard deviation. Furthermore, all plots are smoothed by the moving average over 9 data points.

Following the settings mentioned above, the pedestrian is trained separately with two different reward functions defined in Equations 15 and 16. In addition, each reward function is trained three times to avoid occasional cases. The training performance of the suicidal pedestrian using two designed reward functions are

illustrated in Figure 18. Overall, by training the pedestrian policies for 70000 steps (each episode equals 30 steps at most due to the pedestrian action repeat), both the policy $\pi_{constant}$ incented by the constant reward and the policy π_{comb} incented by the combinational reward converge to some specific behaviors. Moreover, some general conclusions can be drawn from this figure.

Considering the convergence rate, the policy $\pi_{constant}$ converges faster than the policy π_{comb}. Policy $\pi_{constant}$ gets to a stable episodic mean reward state after 50000 steps of training. In contrast, policy π_{comb} only converges almost at the end of the training phase. This is unsurprising since the combinational reward function is much more complex than the constant reward.

Considering the resulting behavior, policy π_{comb} performs better than policy $\pi_{constant}$ in terms of average steps needed to hit the vehicle. This indicates that behaviors from policy π_{comb} are more aggressive in searching for and hitting the vehicle, while behaviors from policy $\pi_{constant}$ are not. Furthermore, surprisingly, the standard deviation and entropy loss from Figure 18 suggest that the policy π_{comb} is more stable than the policy $\pi_{constant}$. This is probably because the policy $\pi_{constant}$ only provides simple behaviors such that it cannot handle some complex traffic scenarios.

4.3 Effectiveness evaluation of the suicidal pedestrian

In order to analyze the effectiveness of the suicidal pedestrian in detail such that the pedestrian is ensured to have the ability to find failures for different state-of-the-art AD algorithms, this thesis conducts additional evaluation experiments in two towns: Town 1 and Town 2. Notably, the start positions and orientations of the vehicle in Town 2 for evaluation are different from those used during the training phase: the vehicle is randomly initialized over the whole town during evaluation, while during training, it is spawned at four selected locations. In addition, the CARLA behavior agent is still the target AV.

For each evaluation experiment, testing is carried out over 100 episodes. Each episode follows the settings described in Table 2 and is considered successful only if the pedestrian hits the target AV. Moreover, each experiment is repeated three times in the corresponding town.

The effectiveness of the suicidal pedestrian is measured by metrics defined in Section 3.3: collision rate C_V, collision running rate C_R, front collision rate C_F, and side collision rate C_S. For the first metric C_V, having values closer to 1 indicates that the pedestrian is performing adversarial suicidal behaviors that can cause the vehicle decision error, whereas closer to 0 means a failed suicidal policy. For the second metric C_R, a higher value indicates a more complex and unpredictable pedestrian behavior in terms of the fact that the vehicle does not perform hard brakes in time before the collision happens. Finally, the last two metrics C_F and C_S are a detailed analysis of the collision rate C_V to illustrate the exact collision areas.

4.3.1 Result analysis

The evaluation results are shown in Table 4 and Table 5. Based on these two tables, several significant findings can be concluded. First, both two reward functions can render excellent policies that have enough capacity to hit the vehicle. Second, generalizing the pedestrian agent to a new town only leads to a slight performance decline. Third, the combinational reward function outperforms the constant reward function on almost all metrics. The following will discuss these key conclusions in detail.

Reward type	**Collision rate**	**Front collision**	**Side collision**	**Collision running**
	% ↑	% ↑	% ↓	% ↑
constant reward	0.84 ± 0.05	0.77 ± 0.04	0.16 ± 0.04	0.42 ± 0.09
combinational reward	$\mathbf{0.90} \pm 0.03$	$\mathbf{0.82} \pm 0.05$	$\mathbf{0.15} \pm 0.03$	$\mathbf{0.55} \pm 0.01$

Table 4: Performance evaluation of the pedestrian in the same town (Town 2) in terms of metrics C_V, C_R, C_F, and C_S, averaged across 3 runs, with each run containing 100 episodes. Values shown in the table are the mean and the corresponding standard deviation. ↑ and ↓ denote that a higher/lower value represents better performance. The results suggest that the pedestrian trained using both reward functions can successfully collide with the target vehicle.

Reward type	**Collision rate**	**Front collision**	**Side collision**	**Collision running**
	% ↑	% ↑	% ↓	% ↑
constant reward	0.79 ± 0.00	0.73 ± 0.05	0.22 ± 0.05	0.43 ± 0.04
combinational reward	$\mathbf{0.86} \pm 0.02$	$\mathbf{0.80} \pm 0.04$	$\mathbf{0.17} \pm 0.01$	$\mathbf{0.54} \pm 0.05$

Table 5: Performance evaluation of the pedestrian in a new town (Town 1). The results illustrate that the pedestrian trained with both reward functions can generalize to new environments.

Performance. Both two reward functions perform well in guiding the pedestrian to hit the target AV. Regarding the collision rate, it fluctuates between 75% and 90% in the two towns with small deviations. Furthermore, more than half of all collisions happen when the AV does not stop in time, meaning that the pedestrian has successfully cheated the vehicle. As for the collision areas, the front part of the vehicle receives almost 80% of collisions.

However, none of the methods can perfectly hit the vehicle. Most possible reasons for this situation come from two aspects. On the one hand, the pedestrian directly observes the relative location and velocity of the target AV while ignoring all environmental obstacles, such as dustbins, billboards, and traffic lattices. Due to this limitation, the pedestrian may be blocked by these barriers when it tries

to walk close to the vehicle. Another important reason is that the pedestrian and the vehicle are independent non-communicating players, rendering a very limited ability for the pedestrian to predict the future trajectory of the AV. Therefore, the suicidal pedestrian may sometimes fail to hit the car because of insufficient or wrong predictions.

Generalization. The generalization experiment focuses on deploying the suicidal pedestrian agent to a previously unknown environment. Surprisingly, the performance of both reward functions declines slightly, where the collision rate of the constant reward decreases from 84% to 79%, and the collision rate of the combinational reward decreases from 90% to 86%. Similarly, other metric values also reduce with small amounts.

Obviously, the suicidal pedestrian shows an excellent generalization capacity, which indicates a potential that the pedestrian can be further extended to various traffic scenarios. One explanation for this phenomenon is the environment-independent property of the pedestrian input and output representations. For the pedestrian agent generalized to a new environment, the most challenging thing is how to represent the necessary information as it does in familiar environments. Luckily, the designed input state vector represents valuable information in the pedestrian coordinate, thus eliminating related problems caused by environmental changes.

Policy comparison. Interestingly, the performance of the two reward functions is similar under two testing towns in terms of collision rate, front collision rate, and side collision rate. These metric values of the two methods differ by less than 10%. However, the collision running rate is an exception. In Town 2, the collision running rate of the constant reward function is 13% less than that of the combinational reward function. In Town 1, this value differs by 11%. This difference in collision running rate implies that behaviors produced by the combinational reward function are more complex and unpredictable such that they can cause the AV decision error more easily. In this sense, the combinational reward function is preferred.

On the other hand, although the performance of both reward functions does not dramatically differ, policies produced by the combinational reward function always perform better regarding all four evaluation metrics. This is because the combinational reward function is more instructive in the training phase due to its reward ranking and increment mechanism.

4.3.2 Behavior visualization

In order to intuitively understand the typical behaviors of the suicidal pedestrian, a visualization of some successful episodes gathered from two towns is provided in Figure 19. This figure shows the process of the pedestrian approaching the target AV from a bird-eye view. Based on the visualization, behaviors of the suicidal pedestrian are categorized into three modes: simple mode, cheating mode, and chasing mode. The simple mode means that the pedestrian directly hits the AV from the central front without any other cheating actions. This mode is straightforward, and therefore it is easy for the AV to avoid decision errors. The cheating mode refers to the pedestrian hitting the vehicle with cheating or unpredictable actions, such as walking

close to the vehicle from occluded areas or pretending to walk away. Compared to the simple mode, this cheating mode can result in more AV decision failures. Lastly, the chasing mode indicates that the pedestrian fails to collide with the target AV. The reason for this manner is that the pedestrian has wrongly predicted the future trajectories of the target AV.

Figure 19: Typical behaviors of the trained pedestrian agent using the combinational reward function. The top row illustrates that the pedestrian directly hits the vehicle from the central front. The middle row demonstrates that the pedestrian crashes the car from the side. Finally, the bottom row illustrates that the pedestrian first misses the vehicle due to a wrong prediction and then chases the missed car. Both behaviors displayed in the top and middle rows are desired because these behaviors would challenge the ability of AVs facing such sudden emergence. In contrast, the behavior illustrated in the bottom row is ineffective since the vehicle does not need to avoid collisions with pedestrians behind it while driving forward.

4.4 Finding failures in autonomous driving algorithms

Considering that one important goal of this thesis is to test various AD algorithms using the designed suicidal pedestrian and find their potential improvements, this section selects some state-of-the-art end-to-end AD models to illustrate how this goal can be done. Because it is easier to integrate CARLA-implementation-based AD algorithms into the designed testing environment, all AD algorithms are selected from the CARLA leaderboard [76], an open platform for the AD community to evaluate and compare various autonomous agents. Besides, two additional criteria are applied during the selection: the method rank and the source code availability. The method

rank, illustrated in Appendix A, is an intuitive indicator of the performance of each AD algorithm. Usually, a method with a higher rank can have better performance, whereas methods with lower ranks perform worse. Therefore, it is more valuable to test some higher-ranked AD algorithms. On the other hand, the decision of source code availability criteria is from practical considerations rather than technical concerns: since developing AD algorithms is not in the scope of this thesis, using available driving algorithms is sufficient for performance testing purposes.

Based on these selection criteria, two algorithms are chosen. The first selected driving algorithm is LAV [64], an imitation learning-based method in which the AV agent is trained to imitate the CARLA behavior agent with a dataset collected from all vehicles it observes. The second algorithm is InterFuser [65], which aims to develop a comprehensive scene understanding ability with the help of sensor-fusion to interpret the decision process of the AV and ensure driving safety. Furthermore, both methods use their pre-trained models and parameters submitted in the CARLA leaderboard, which can be seen in Appendix B. Lastly, the CARLA behavior agent is used as a baseline to verify whether these two algorithms can better tackle the designed suicidal pedestrian-related traffic scenarios.

In this section, three metrics are used to evaluate the performance of the AV agent: the pedestrian average episodic reward, the collision rate C_V, and the collision running rate C_R. Notably, since the evaluation entity is now the AV agent, both collision rate and collision running rate are explained from the perspective of the vehicle. For the collision rate, a value closer to 0 means the vehicle has enough capacity to avoid collisions with the suicidal pedestrian, whereas a value closer to 1 means the opposite. For the running collision rate, lower values suggest that the vehicle tries to reduce collision hazards by stopping early, thus proving the capacity of the AV to deal with the sudden emergence of pedestrians by performing hard brakes. Furthermore, the pedestrian average episodic reward is a general metric to describe the collision, with lower values implying that it is more difficult to result in crashes or cause severe consequences. This metric is affected by both the collision rate and the collision running rate.

Finally, as has been done in Section 4.3, all AD algorithms are tested three times in each town, with each test containing 100 episodes. The results and corresponding analysis are described in the following.

4.4.1 Performance verification

Table 6 illustrates the performance of LAV and InterFuser against the suicidal pedestrian. Notably, these quantitative results of the three driving models shown in this table are not comparable, i.e., it cannot be concluded that LAV and InterFuser perform better than CARLA behavior agent even though they achieve the lowest pedestrian reward and collision running rate. This is because the suicidal pedestrian is trained with the CARLA behavior agent, rendering that the pedestrian has learned to exploit the CARLA agent specifically. Once the pedestrian is trained with LAV or InterFuser, its behavior may change significantly.

However, some helpful information can still be obtained. Overall, the trained

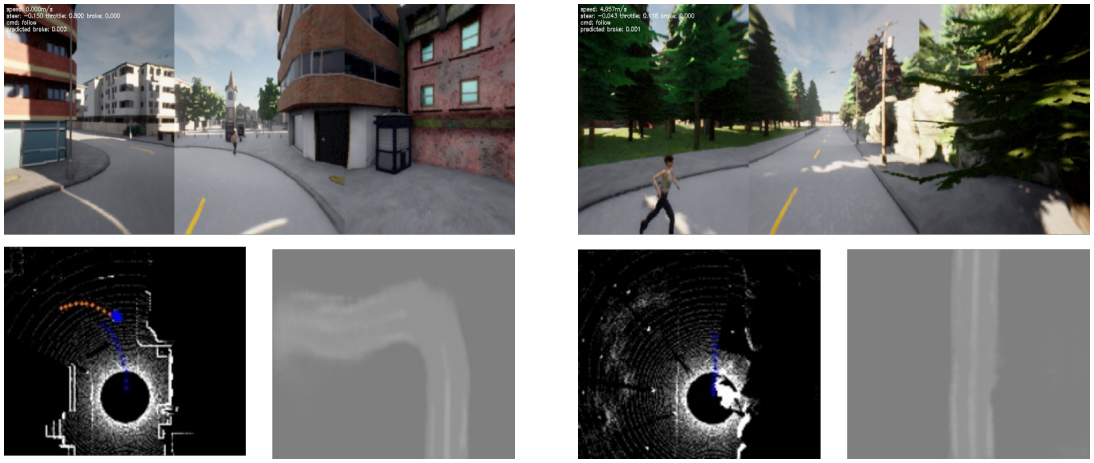
pedestrian can generate collisions with LAV and InterFuser, showing potential weaknesses of these two driving algorithms. For LAV, the collision rate reaches 93% in Town 2 and 90% in Town 1, with 47% and 61% collision running rates, respectively. For InterFuser, the collision rates in the two towns are a bit higher, but the collision running rates are much lower. This suggests that most crashes of InterFuser are not severe, while LAV is more likely to cause hazardous consequences when a collision happens.

	Same town (Town 2)			New town (Town 1)		
Method	**Pedestrian reward**	**Collision rate**	**Collision running**	**Pedestrian reward**	**Collision rate**	**Collision running**
	% ↓	% ↓	% ↓	% ↓	% ↓	% ↓
Baseline	4.57 ± 0.15	$\mathbf{0.90} \pm 0.03$	0.55 ± 0.02	4.29 ± 0.29	$\mathbf{0.86} \pm 0.02$	0.54 ± 0.05
LAV [64]	$\mathbf{3.56} \pm 0.26$	0.93 ± 0.02	0.47 ± 0.04	3.94 ± 0.26	0.90 ± 0.03	0.61 ± 0.01
InterFuser [65]	3.77 ± 0.38	0.94 ± 0.02	$\mathbf{0.32} \pm 0.06$	$\mathbf{3.82} \pm 0.16$	0.92 ± 0.02	$\mathbf{0.37} \pm 0.02$

Table 6: Evaluation results of two state-of-the-art AD algorithms using the trained suicidal pedestrian in terms of metrics pedestrian average episodic reward, collision rate, and collision running rate.

4.4.2 Failure case visualization

In order to better understand the decision failures of LAV and InterFuser such that advice can be given to improve them, it is necessary and instructive to visualize their working process.



(a) (b)

Figure 20: Visualization of two detection errors of LAV with three concatenated RGB images, detection and motion predictions, and predicted road geometries. (a) Visualization of incorrect detection error that predicts the pedestrian as a vehicle. (b) Visualization of unsuccessful detection error that fails to find the pedestrian due to insufficient fusion of images.

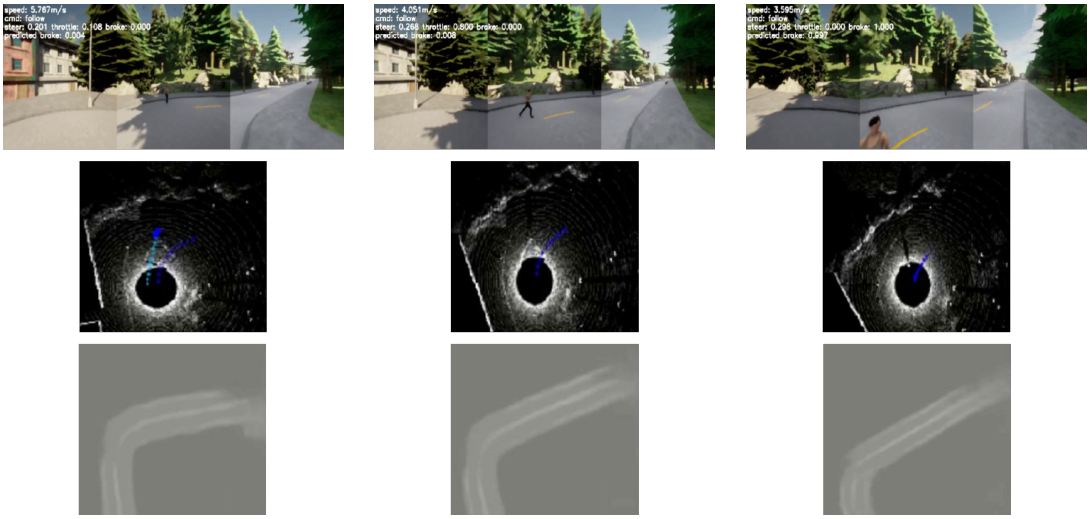


Figure 21: Visualization of one failure episode of LAV in which the vehicle agent does not perform hard brakes in time to avoid the pedestrian when it is turning. The left column shows the pedestrian is incorrectly detected as a vehicle. The middle column illustrates the vehicle fails to detect the pedestrian. Lastly, the right column shows that LAV successfully finds the pedestrian but is already late to avoid a hazardous collision.

In Figure 20, two typical errors of LAV are illustrated: incorrect detection and unsuccessful detection. The incorrect detection error shown in Figure 20(a) means that LAV detects the pedestrian as a vehicle, thus applying unreasonable dynamic models to the pedestrian to predict the corresponding trajectories. In contrast, the unsuccessful detection error refers to a case where LAV fails to detect the pedestrian due to technical flaws, which is shown in Figure 20(b). Interestingly, both errors can happen at different stages of one episode. For example, Figure 21 provides a failure case of LAV to deal with the designed suicidal pedestrian. In this case, the AV agent first incorrectly detects the pedestrian as a vehicle and predicts inappropriate future trajectories of the pedestrian. Then, the unsuccessful detection error occurs after some timesteps, resulting in the tracking failure of that pedestrian. Finally, LAV recovers its accurate predictions and decides on proper control commands, though severe collision is already unavoidable.

Figure 22 and Figure 23 illustrate two failure episodes of InterFuser, where the former one shows that the vehicle agent does not perform any actions to avoid collisions with the suicidal pedestrian even if the pedestrian is always detected, whereas the latter one shows a late and unsuccessful survival action when the pedestrian is too close to the vehicle. In addition, these two failures expose a common property: both of them successfully and accurately detect the pedestrian but perform the corresponding prediction extremely poorly. For example, according to RGB images shown in Figure 22, the pedestrian is walking close to the vehicle; however, InterFuser predicts the opposite, i.e., the pedestrian will walk away from the car. Therefore, this property suggests that the flaws of InterFuser lie in its prediction and subsequent

decision-making modules rather than the detection part, and potential improvements should focus on its prediction process.

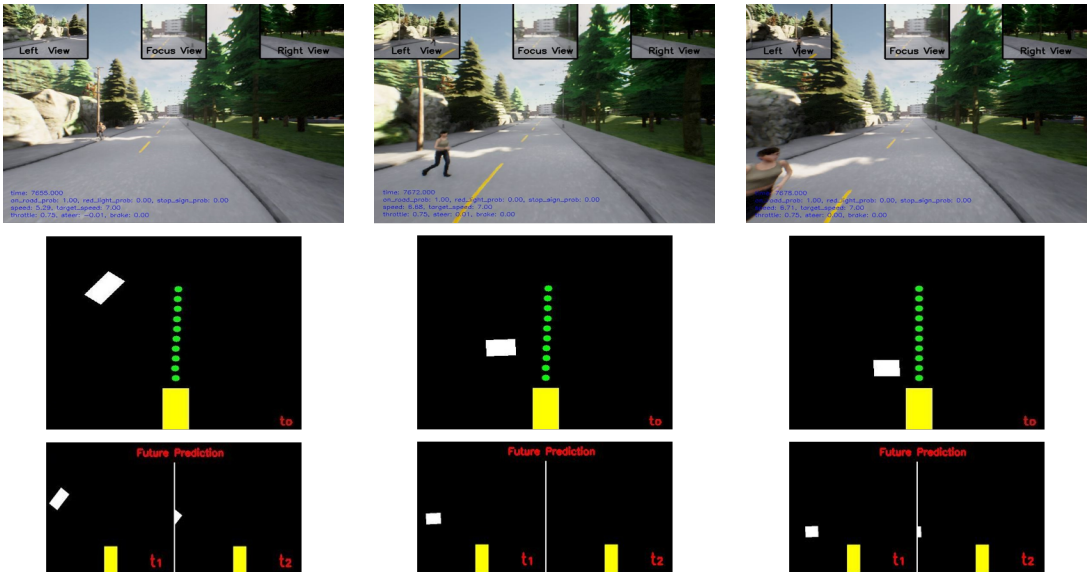


Figure 22: Visualization of one failure episode of InterFuser in which the AV agent does not perform any actions to avoid collisions. Each row visualizes three RGB images, detected traffic scenes at the current timestep, and predicted traffic scenes at the next two timesteps. The yellow rectangle in the last two rows represents the ego vehicle, while white rectangles represent other detected objects. Green dots are the future trajectory of the ego vehicle.

4.4.3 Discussion

As discussed in previous sections, AD algorithm evaluation aims to find weaknesses and propose potential improvements such that the developed algorithms can achieve better performance. According to Figure 20 and Figure 21, most failures of LAV to deal with the designed suicidal pedestrian-related traffic scenarios are caused by its detection errors. By analyzing the structure of the detection module in detail, it is found that the information loss during multi-view image fusion is the main reason for these errors. In LAV, three RGB images without perspective overlapping are directly concatenated along the channel dimension, which loses critical information at the edges of each concatenating image. Therefore, one possible improvement is to enhance the multi-view sensor fusion ability by either overlapping parts of the camera perspective or adding a more powerful information-grasping mechanism to this module.

As for InterFuser, its main weakness is that it needs more sufficient predictions about the future trajectories of detected vehicles and pedestrians. Currently, InterFuser assumes constant and deterministic models for observed vehicles and pedestrians, predicting their future motion purely based on their moving average

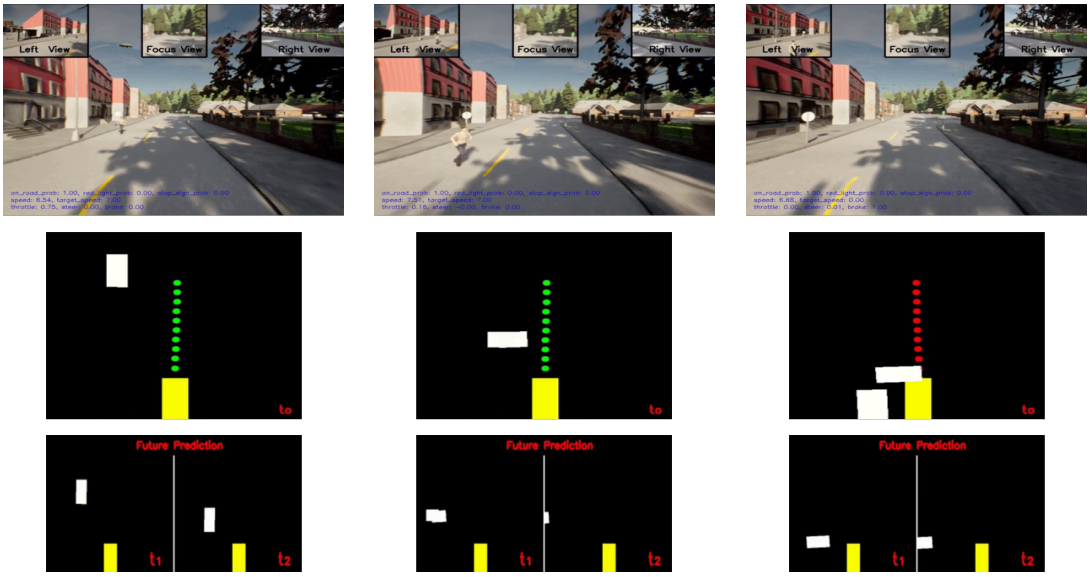


Figure 23: One failure episode of InterFuser in which the ego vehicle has continuously detected the suicidal pedestrian but performed wrong predictions, rendering a late brake that causes the hazardous collision. Red dots represent that the future trajectory is currently not reachable.

states over several historical timesteps. Therefore, in order to improve the performance of InterFuser, more advanced trajectory prediction methods can be considered. One easiest way is to follow the technique used in LAV that takes trajectories of other detected vehicles into the training process and produces a model to predict their future movements.

5 Conclusion and discussion

5.1 Conclusion

Autonomous driving is an important area of research. Developing capable autonomous driving methods can bring extensive and profound practical significance. Regarding the safety concerns of autonomous vehicles, this thesis has developed a suicidal pedestrian model to automatically generate some simulated safety-critical traffic scenarios to perform extensive adversarial testing of various AD algorithms. The main work is concluded in the following.

First, this thesis builds an urban traffic environment with the participation of one vehicle and one pedestrian based on the CARLA simulator [13], where the vehicle could be controlled by any AD algorithms, while the pedestrian is designed to hit this vehicle spontaneously. In this environment, the pedestrian is limited to being initialized in the front of the vehicle with a finite distance such that both the pedestrian and the vehicle can observe each other. Moreover, the OpenAI Gym toolkit [75] is used to standardize the constructed environment as a general RL framework.

Second, this thesis models the suicidal pedestrian as an RL problem and trains it with a standard RL algorithm, as well as analyzes the performance of this trained pedestrian using proposed evaluation metrics. By properly defining the input state, output action, and reward functions of the pedestrian, the PPO [33] algorithm is successfully utilized to help the pedestrian find some suicidal behaviors that can cause the autonomous car to perform failure decisions. Besides, evaluation experiments for two proposed reward functions suggest that a function giving rewards proportional to the collision speed and the collision area can help the pedestrian find more complex and unpredicted behaviors.

Finally, this thesis also evaluates LAV [64] and InterFuser [65], two state-of-the-art AD algorithms from the CARLA leaderboard [76], using the designed suicidal pedestrian. Experimental results of these two driving algorithms demonstrate that the suicidal pedestrian can be effective in finding decision errors in different driving algorithms.

5.2 Discussion

Although the suicidal pedestrian is effective for AV testing, some limitations exist. Firstly, the input state of the suicidal pedestrian only contains information about the target vehicle while ignoring the environmental obstacles. Due to this simplification, the pedestrian may be blocked by those obstacles when it tries to walk close to the vehicle, resulting in the pedestrian failing to hit the car. One future extension is to utilize raw image inputs for replacing the hand-crafted state vector, thus allowing the pedestrian to plan movements according to the surroundings. However, since images are highly dimensional and contain much useless information, dimension-reduction and feature-extraction techniques, such as variational autoencoders or generative adversarial networks, should be considered in the future.

Furthermore, the simulated traffic scenarios are simplified to contain one pedestrian and one vehicle, which significantly constrains the application scope of these scenarios. Therefore, future work can focus on extending the simulated scenarios having more pedestrians and cars. In such complex scenarios, the suicidal pedestrian should distinguish between the target vehicle and other vehicles, as well as pedestrians.

References

[1] B. Helou, A. Dusi, A. Collin, N. Mehdipour, Z. Chen, C. Lizarazo, C. Belta, T. Wongpiromsarn, R. D. Tebbens, and O. Beijbom, "The reasonable crowd: Towards evidence-based and interpretable models of driving behavior," in *Proceedings of the 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 6708–6715.

[2] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20255

[3] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Practical search techniques in path planning for autonomous driving," *Ann Arbor*, vol. 1001, no. 48105, pp. 18–80, 2008.

[4] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," in *Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 4693–4700.

[5] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 2722–2730.

[6] D. Chen, V. Koltun, and P. Krähenbühl, "Learning to drive from a world on rails," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 15 590–15 599.

[7] R. Chekroun, M. Toromanoff, S. Hornauer, and F. Moutarde, "Gri: General reinforced imitation and its application to vision-based autonomous driving," *arXiv*, 2021. [Online]. Available: https://arxiv.org/abs/2111.08575

[8] Z. Peng, Q. Li, K. M. Hui, C. Liu, and B. Zhou, "Learning to simulate self-driven particles system with coordinated policy optimization," *Advances in Neural Information Processing Systems*, vol. 34, pp. 10 784–10 797, 2021.

[9] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic traffic simulation using sumo," in *Proceedings of the 2018 IEEE 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 2575–2582.

[10] A. Sharif and D. Marijan, "Adversarial deep reinforcement learning for improving the robustness of multi-agent autonomous driving policies," in *Proceedings of the 2022 29th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2022, pp. 61–70.

[11] Y. Abeysirigoonawardena, F. Shkurti, and G. Dudek, "Generating adversarial driving scenarios in high-fidelity simulators," in *Proceedings of the 2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8271–8277.

[12] D. Karunakaran, S. Worrall, and E. Nebot, "Efficient statistical validation with edge cases to evaluate highly automated vehicles," in *Proceedings of the 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, 2020, pp. 1–8.

[13] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning (CoRL)*. PMLR, 2017, pp. 1–16.

[14] B. Hurl, K. Czarnecki, and S. Waslander, "Precise synthetic image and lidar (presil) dataset for autonomous vehicle perception," in *Proceedings of the 2019 IEEE Intelligent Vehicles Symposium (IV)*, 2019, pp. 2522–2529.

[15] M. Martinez, C. Sitawarin, K. Finch, L. Meincke, A. Yablonski, and A. Kornhauser, "Beyond grand theft auto v for training, testing and enhancing deep learning in self driving cars," *arXiv*, 2017. [Online]. Available: https://arxiv.org/abs/1712.01397

[16] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," in *Proceedings of the 2016 European Conference on Computer Vision (ECCV)*. Springer, 2016, pp. 102–118.

[17] B. Wymann, E. Espié, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner, "Torcs, the open racing car simulator," *Citeseer*, 2015.

[18] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "Sumo–simulation of urban mobility: an overview," in *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind, 2011.

[19] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of sumo-simulation of urban mobility," *International Journal on Advances in Systems and Measurements*, vol. 5, no. 3&4, 2012.

[20] A. Best, S. Narang, L. Pasqualin, D. Barber, and D. Manocha, "Autonovisim: Autonomous vehicle simulation platform with weather, sensing, and traffic control," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2018, pp. 1048–1056.

[21] E. Games. Unreal engine 4. [Online]. Available: https://www.unrealengine.com

[22] A. Best, S. Narang, D. Barber, and D. Manocha, "Autonovi: Autonomous vehicle planning with dynamic maneuvers and traffic constraints," in *Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 2629–2636.

[23] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[24] M. Toromanoff, E. Wirbel, and F. Moutarde, "End-to-end model-free reinforcement learning for urban driving using implicit affordances," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[25] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of The 33rd International Conference on Machine Learning (ICML)*. PMLR, 2016, pp. 1928–1937.

[26] R. Bellman, "On the theory of dynamic programming," *Proceedings of the National Academy of Sciences*, vol. 38, no. 8, pp. 716–719, 1952.

[27] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017. [Online]. Available: https://doi.org/10.1109/MSP.2017.2743240

[28] T. L. Lai, H. Robbins *et al.*, "Asymptotically efficient adaptive allocation rules," *Advances in Applied Mathematics*, vol. 6, no. 1, pp. 4–22, 1985.

[29] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv*, 2015. [Online]. Available: https://arxiv.org/abs/1509.02971

[30] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," *arXiv*, 2018. [Online]. Available: https://arxiv.org/abs/1810.12894

[31] R. Raileanu and T. Rocktäschel, "Ride: Rewarding impact-driven exploration for procedurally-generated environments," *arXiv*, 2020. [Online]. Available: https://arxiv.org/abs/2002.12292

[32] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv*, 2013. [Online]. Available: https://arxiv.org/abs/1312.5602

[33] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv*, 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[34] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*. PMLR, 2018, pp. 1861–1870.

[35] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[36] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on Machine Learning (ICML)*. PMLR, 2015, pp. 1889–1897.

[37] S. Racanière, T. Weber, D. Reichert, L. Buesing, A. Guez, D. Jimenez Rezende, A. Puigdomènech Badia, O. Vinyals, N. Heess, Y. Li *et al.*, "Imagination-augmented agents for deep reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[38] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel, "Model-ensemble trust-region policy optimization," in *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018. [Online]. Available: https://openreview.net/forum?id=SJJinbWRZ

[39] I. Clavera, J. Rothfuss, J. Schulman, Y. Fujita, T. Asfour, and P. Abbeel, "Model-based reinforcement learning via meta-policy optimization," in *Proceedings of the 2nd Conference on Robot Learning (CoRL)*. PMLR, 2018, pp. 617–629.

[40] J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee, "Sample-efficient reinforcement learning with stochastic ensemble value expansion," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[41] J. Kim and M. Lee, "Robust lane detection based on convolutional neural network and random sample consensus," in *Proceedings of the 21st International Conference on Neural Information Processing (ICONIP)*. Springer, 2014, pp. 454–461.

[42] A. Gurghian, T. Koduri, S. V. Bailur, K. J. Carey, and V. N. Murali, "Deeplanes: End-to-end lane position estimation using deep neural networksa," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2016, pp. 38–45.

[43] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue, F. Mujica, A. Coates, and A. Y. Ng, "An empirical evaluation of deep learning on highway driving," *arXiv*, 2015. [Online]. Available: https://arxiv.org/abs/1504.01716

[44] P.-R. Chen, S.-Y. Lo, H.-M. Hang, S.-W. Chan, and J.-J. Lin, "Efficient road lane marking detection with deep learning," in *Proceedings of the 2018 IEEE 23rd International Conference on Digital Signal Processing (DSP)*, 2018, pp. 1–5.

[45] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv*, 2014. [Online]. Available: https://arxiv.org/abs/1409.1556

[46] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 12 697–12 705.

[47] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[48] S. Vora, A. H. Lang, B. Helou, and O. Beijbom, "Pointpainting: Sequential fusion for 3d object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 4604–4612.

[49] M. Liang, B. Yang, S. Wang, and R. Urtasun, "Deep continuous fusion for multi-sensor 3d object detection," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 641–656.

[50] F. Bastani, S. He, S. Abbar, M. Alizadeh, H. Balakrishnan, S. Chawla, S. Madden, and D. DeWitt, "Roadtracer: Automatic extraction of road networks from aerial images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 4720–4728.

[51] G. Mattyus, W. Luo, and R. Urtasun, "Deeproadmapper: Extracting road topology from aerial images," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 3438–3446.

[52] S. Casas, A. Sadat, and R. Urtasun, "Mp3: A unified model to map, perceive, predict and plan," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 14 403–14 412.

[53] N. Homayounfar, W.-C. Ma, S. K. Lakshmikanth, and R. Urtasun, "Hierarchical recurrent attention networks for structured online maps," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 3417–3426.

[54] M. Lin, J. Yoon, and B. Kim, "Self-driving car location estimation based on a particle-aided unscented kalman filter," *Sensors*, vol. 20, no. 9, p. 2544, 2020.

[55] F. Ma, J. Shi, Y. Yang, J. Li, and K. Dai, "Ack-msckf: Tightly-coupled ackermann multi-state constraint kalman filter for autonomous vehicle localization," *Sensors*, vol. 19, no. 21, p. 4816, 2019.

[56] G. Bresson, M.-C. Rahal, D. Gruyer, M. Revilloud, and Z. Alsayed, "A cooperative fusion architecture for robust localization: Application to autonomous

driving," in *Proceedings of the 2016 IEEE 19th International Conference on Intelligent Transportation systems (ITSC)*. IEEE, 2016, pp. 859–866.

[57] L. Li, M. Yang, C. Wang, and B. Wang, "Road dna based localization for autonomous vehicles," in *Proceedings of the 2016 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2016, pp. 883–888.

[58] B. Paden, M. Cap, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *arXiv*, 2016. [Online]. Available: https://arxiv.org/abs/1604.07446

[59] T. Fraichard, "Trajectory planning in a dynamic workspace: a'state-time space'approach," *Advanced Robotics*, vol. 13, no. 1, pp. 75–94, 1998.

[60] J. Jaafar, E. McKenzie, and A. Smaill, "A fuzzy action selection method for virtual agent navigation in unknown virtual environments," in *Proceedings of the 2007 IEEE International Fuzzy Systems Conference*, 2007, pp. 1–6.

[61] S. Dixit, S. Fallah, U. Montanaro, M. Dianati, A. Stevens, F. Mccullough, and A. Mouzakitis, "Trajectory planning and tracking for autonomous overtaking: State-of-the-art and future prospects," *Annual Reviews in Control*, vol. 45, pp. 76–86, 2018.

[62] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," *Advances in Neural Information Processing Systems*, vol. 1, 1988.

[63] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, "Learning by cheating," in *Proceedings of the Conference on Robot Learning (CoRL)*. PMLR, 2020, pp. 66–75.

[64] D. Chen and P. Krähenbühl, "Learning from all vehicles," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 17 222–17 231.

[65] H. Shao, L. Wang, R. Chen, H. Li, and Y. Liu, "Safety-enhanced autonomous driving using interpretable sensor fusion transformer," *arXiv*, 2022. [Online]. Available: https://arxiv.org/abs/2207.14024

[66] T. Yin, X. Zhou, and P. Krahenbuhl, "Center-based 3d object detection and tracking," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 11 784–11 793.

[67] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," in *Proceedings of the 2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8248–8254.

[68] J. Chen, S. E. Li, and M. Tomizuka, "Interpretable end-to-end urban autonomous driving with latent deep reinforcement learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 6, pp. 5068–5078, 2021.

[69] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58 443–58 469, 2020.

[70] M. Priisalu, A. Pirinen, C. Paduraru, and C. Sminchisescu, "Generating scenarios with diverse pedestrian behaviors for autonomous vehicle testing," in *Proceedings of the 5th Conference on Robot Learning (CoRL)*. PMLR, 2022, pp. 1247–1258.

[71] W. Ding, B. Chen, M. Xu, and D. Zhao, "Learning to collide: An adaptive safety-critical scenarios generating method," in *Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 2243–2250.

[72] A. Rasouli and J. K. Tsotsos, "Autonomous vehicles that interact with pedestrians: A survey of theory and practice," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, pp. 900–918, 2020.

[73] M. Priisalu, C. Paduraru, A. Pirinen, and C. Sminchisescu, "Semantic synthesis of pedestrian locomotion," in *Proceedings of the Asian Conference on Computer Vision (ACCV)*, 2020.

[74] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[75] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv*, 2016. [Online]. Available: https://arxiv.org/abs/1606.01540

[76] C. team. Carla autonomous driving leaderboard. 2022. [Online]. Available: https://leaderboard.carla.org/

[77] P. Wu, X. Jia, L. Chen, J. Yan, H. Li, and Y. Qiao, "Trajectory-guided control prediction for end-to-end autonomous driving: A simple yet strong baseline," *arXiv*, 2022. [Online]. Available: https://arxiv.org/abs/2206.08129

A CARLA leaderboard rank

This section provides the performance comparison on the public CARLA leaderboard [76] to illustrate why LAV [64] and InterFuser [65] are selected as the testing algorithms. In Table A1, although ReasonNet ranks first with the highest driving score of 79.95 and the highest infraction score of 0.89, its submission is anonymous. Hence, no research paper and corresponding source code are provided. Among the remaining three methods, InterFuser achieves the highest driving score, TCP [77] achieves the highest infraction score, and LAV reaches the highest route completion. However, since InterFuser and TCP share some similarities in their design and perform almost the same in all three metrics, maintaining one of them is enough for the testing purpose in this thesis. Therefore, considering these factors, InterFuser and LAV are finally selected.

Rank	Method	Source code	Driving score	Route completion	Infraction score
1	ReasonNet	No	**79.95**	89.89	**0.89**
2	InterFuser [65]	Yes	76.18	88.23	0.84
3	TCP [77]	Yes	75.14	85.63	0.87
4	LAV [64]	Yes	61.85	**94.46**	0.64

Table A1: CARLA leaderboard [76] evaluation of different driving algorithms (accessed Jan 2023). Route completion measures the distance percentage completed by the agent. Infraction score is a discount value that measures the safety of the vehicle agent. Driving score is a weighted product of route completion and infraction score. All metrics are higher the better.

B Hyperparameter values

This section provides some fundamental hyperparameter values used in LAV and InterFuser, respectively.

Notation	Description	Value
$turn_{KP}$	Proportional coefficient value for steering PID controller	1.0
$turn_{KI}$	integral coefficient value for steering PID controller	0.5
$turn_{KD}$	Derivative coefficient value for steering PID controller	0.2
$speed_{KP}$	Proportional coefficient value for throttle PID controller	5.0
$speed_{KI}$	integral coefficient value for throttle PID controller	0.5
$speed_{KD}$	Derivative coefficient value for throttle PID controller	1.0
v_{max}	Maximal allowed speed	35 km/h
n_{iter}	Number of iterations to produce the future trajectory	5
n_{plan}	Number of future waypoints in one predicted trajectory	10
$r_{vehicle}$	Maximal distance to detect the vehicle	15 m
$r_{pedestrian}$	Maximal distance to detect the pedestrian	10 m

Table B1: Key parameters used for LAV.

Notation	Description	Value
$turn_{KP}$	Proportional coefficient value for steering PID controller	2.35
$turn_{KI}$	integral coefficient value for steering PID controller	0.65
$turn_{KD}$	Derivative coefficient value for steering PID controller	0.45
$speed_{KP}$	Proportional coefficient value for throttle PID controller	5.0
$speed_{KI}$	integral coefficient value for throttle PID controller	0.5
$speed_{KD}$	Derivative coefficient value for throttle PID controller	1.0
v_{max}	Maximal allowed speed	25 km/h
a_{max}	Maximal allowed acceleration	1.0 m/s^2
	Size of detected area	20 m × 20 m
	Higher threshold for filtering object in the density map	0.9
	Lower threshold for filtering object in the density map	0.5

Table B2: Key parameters used for InterFuser.