August 2023

Realistic Speed Control of Agents in Traffic Simulation

Lakshman Karthik Ramkumar
University of Wisconsin-Milwaukee

REALISTIC SPEED CONTROL OF AGENTS IN TRAFFIC SIMULATION

by

Lakshman Karthik Ramkumar

A Thesis Submitted in
Partial Fulfillment of the
Requirements for the Degree of

Master of Science
in Computer Science

at
The University of Wisconsin-Milwaukee
August 2023

ABSTRACT

REALISTIC SPEED CONTROL OF AGENTS IN TRAFFIC SIMULATION

by

Lakshman Karthik Ramkumar

The University of Wisconsin-Milwaukee, 2023
Under the Supervision of Professor Tian Zhao

Agents in multi-agent traffic simulation tend to be more dependent on the rules and existing instructions to move mechanically and unnaturally imitating human behaviors. The agents will not accelerate or decelerate as humans do. Humans have an irregular pattern of acceleration and deceleration when it comes to real-time driving. This includes hitting breaks when not necessary and sometimes even driving above the speed limit to catch up. In prior works, other factors such as drag and simulation-specific parameters were not considered in the models. Additionally, the models were not tested on the traffic simulation frameworks like SUMO. Instead, they utilized simple numerical models to simulate the environment and evaluate the performance of the models. Therefore, there is a need to further investigate and incorporate these additional factors, as well as validate the models on the SUMO platform, to enhance the realism and applicability of the research. It is also difficult to calibrate SUMO to a given traffic scenario as traffic engineers might need to specify manually the vehicle specifications while designing the experiments. It would be easier for engineers to populate the road network with pre-trained agents that require minimal tuning which includes specifying maximum acceleration, deceleration, and minimum and maximum speed of the vehicles to be simulated. We propose a unified system for agents to decide when to accelerate and decelerate with the help of deep reinforcement learning aided by a combination of factors such as instantaneous speed, time, and other important metrics. The proposed system will aid the agents to behave more like humans by acting based on the surrounding agents in complex situations. This in turn can help create a diverse traffic flow that can mimic real-life traffic scenarios.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

Transportation engineers and urban planners can benefit from using traffic simulation to model and analyze traffic flow in various scenarios. It is an effective tool for predicting traffic behavior, evaluating alternative solutions, reducing cost and time, improving safety, and enhancing public engagement. A multi-agent simulation is a useful approach for microscopic traffic simulation. Still, it can sometimes produce unnatural or ineffective results due to the use of simplistic behavior models based on existing research. For example, some models imitate stereotyped driving behaviors using predefined rules, which can lead to unnatural traffic flows and inaccurate real-life traffic investigations. To address this, agents should be constructed with the flexibility to adapt their driving behavior to different surrounding situations.

In a typical driving scenario, the agent might interact with vehicles that may show up from anywhere on the road. In reality, human drivers control their vehicles by responding to complex conditions which is impossible to model with rules for each possible condition, given the action (acceleration/deceleration) is continuous.

When it comes to velocity control driver models are a critical part that defines the car-following behavior. Driver models are distinguished as rule-based and supervised learning approaches. The rule-based approach represents the traditional car-follow models such as the Intelligent Driver Model (IDM). The supervised learning approach utilizes the data

from human demonstrations to approximate the actions taken and the car-following states [1]. Though these two approaches emulate human drivers' car-following behavior, it might not be the best solution to reproduce realistic traffic as the human driving data is from a very small sample when compared to the global drivers' population, which is more diverse. These autonomous agents can be trained to drive considering safety, efficiency, and comfort besides imitating human behavior as human drivers may not drive optimally.

In this paper, we present a Deep Reinforcement Learning (DRL) based approach which uses Soft Actor-Critic (SAC) [2] algorithm to train the agent to drive safely given complex conditions. The model optimizes safety, comfort, and efficiency by learning from interactions in the simulation environment (SUMO). SAC is known to work well on continuous action spaces and is better in early exploration which can be tuned through a temperature parameter.

The temperature parameter (α) controls the balance between exploration and exploitation during training [3]. It affects the exploration behavior of the policy by adjusting the strength of entropy regularization. A higher temperature promotes more exploration by increasing the impact of entropy regularization, while a lower temperature encourages more deterministic actions. By tuning the temperature parameter, SAC can strike a balance between exploring new actions and exploiting known ones, leading to effective learning and improved performance in reinforcement learning tasks. A reward function has been designed to evaluate driving features like speed limit, comfort, safety, and efficiency.

The major contributions of this paper are:

1. Soft Actor-Critic with Multiple Objectives: The proposed agent is based on Soft Actor-Critic, a reinforcement learning algorithm, capable of learning from multiple objectives simultaneously. Unlike previous approaches that primarily focused on learning velocity or acceleration directly, this work expands the scope by considering multiple objectives such as speed, safety, comfort, and efficiency. By incorporating multiple objectives, the agent can learn to make decisions that optimize across different aspects of driving performance.

2. Abstraction of Acceleration: In contrast to directly learning acceleration, this work abstracts acceleration control through action scaling. By decoupling the specific acceleration values from the action space, the agent can effectively learn how to adjust its behavior without explicitly learning the acceleration dynamics. This abstraction can simplify the learning process and improve the agent's ability to generalize to different driving scenarios.

3. New Reward Function: The work introduces a novel reward function that integrates various factors to guide the agent's behavior. The reward function incorporates desired speed, Time to Collision (TTC), headway (the distance between the agent vehicle and the vehicle ahead), and jerk (rate of change of acceleration). By including these elements, the reward function encourages the agent to drive within the desired speed range, maintain safe distances, and minimize sudden changes in acceleration, resulting in safer and smoother driving behavior.

4. Collision Warning System: To enhance safety, this work incorporates a collision warning system into the velocity control framework. The warning will be triggered when the distance between the lead vehicle and ego vehicle is less than d_{safe}, computed as suggested by [4]. This has proven to avoid collisions during both training and testing.

5. Detailed comparison of performances of the proposed model with the default Krauss model in SUMO, the Intelligent Driver Model, and prior work that uses the DDPG algorithm.

Chapter 2

Background

2.1 Car Follow Model

Car-following models are essential in understanding how an ego vehicle responds to the movements of a leading vehicle. These models are widely used in microscopic simulations and provide theoretical foundations for autonomous car-following systems. Throughout the years, several car-following models have been proposed, dating back to the 1950s.

The Intelligent Driver Model (IDM) [5] is a popular car-following model that considers factors such as desired time headway, desired speed, and the velocity difference between the ego and leading vehicles. By incorporating these variables, IDM calculates the acceleration of the ego vehicle, aiming to maintain a safe distance while accounting for driver preferences.

Another significant model is the Krauss Model [6], which focuses on the interaction between the ego vehicle and the leading vehicle. It takes into account parameters like distance, relative velocity, and relative acceleration to determine the acceleration of the ego vehicle. The Krauss Model emphasizes cooperative driving, where the ego vehicle adjusts its behavior to ensure a stable and harmonious traffic flow.

The optimal velocity model is another widely used car-following model. It operates on the assumption that drivers aim to maintain a constant time headway with the leading vehicle. By considering the desired time headway, current speed, and velocity differences, the

optimal velocity model computes the desired acceleration of the ego vehicle. Its objective is to achieve smooth and efficient traffic flow by optimizing vehicle spacing.

2.2 Reinforcement Learning

Reinforcement Learning (RL) is a computational approach to solving sequential decision-making problems. In RL, an agent interacts with an environment and learns to make optimal decisions to maximize a long-term reward. The process unfolds over a sequence of time steps.

At each time step, the agent observes the current state of the environment, denoted as s_t. Based on this observation, the agent selects an action, denoted as a_t, from a set of possible actions defined by the action space A. The selection of the action is guided by a policy, denoted as $\pi(a_t|s_t)$, which maps states to actions.

Once the agent chooses an action, it receives a reward, denoted as r_t, from the environment. This reward reflects the immediate benefit or penalty associated with the chosen action. Additionally, the environment transitions to a new state, denoted as s_{t+1}, based on the agent's action.

The process of observing the state, selecting an action, receiving a reward, and transitioning to the next state continues in a sequential manner until a terminal state is reached. At that point, the agent restarts the process. The objective of the agent is to maximize the accumulated reward over time.

The accumulated reward, denoted as R_t, is a measure of the total expected reward that the agent aims to maximize. It is computed by summing the discounted rewards obtained at each time step, where the discount factor γ determines the importance of future rewards relative to immediate rewards. The discount factor γ typically lies between 0 and 1, with higher values indicating a greater emphasis on long-term rewards.

By iteratively interacting with the environment, observing states, selecting actions, and receiving rewards, the RL agent learns to optimize its decision-making process to maximize the long-term, cumulative reward [7]. This iterative learning process allows the

agent to adapt its behavior based on the observed rewards and the underlying dynamics of the environment.

$$R_t = \sum_{k=1}^{\infty} \gamma^k r_t, \gamma \in (0, 1] \tag{2.1}$$

2.2.1 Value-based Reinforcement Learning

A value function is a measure of the quality or expected return associated with being in a particular state or state-action pair. The action value function, denoted as $Q^{\pi}(s, a)$, represents the expected return for selecting action a in state s when following a specific policy π. It quantifies the desirability or goodness of taking action a in state s.

Value-based RL methods aim to estimate the action values by leveraging a history of experience. One commonly used value-based RL algorithm is Q-learning. In Q-learning, the agent begins with an initial Q-function, which assigns random values to the state-action pairs. As the agent interacts with the environment and receives rewards, it updates its Q-values based on the Bellman Equation [8].

$$Q(s, a) = E[r + \gamma \max_{a'} Q(s', a')] \tag{2.2}$$

The Bellman Equation captures the intuition that the maximum future reward for the current state s and action a is the immediate reward r obtained and the maximum expected future reward for the resulting state s'. By iteratively updating the Q-values using this equation, the agent gradually learns to estimate the optimal action values.

Once the Q-values are estimated, the agent can determine the optimal policy for decision-making. The optimal policy is to select the action with the highest Q-value in a given state, as it represents the action that is expected to lead to the maximum cumulative future rewards. By following the optimal policy, the agent maximizes its expected return over time.

Value-based RL methods, such as Q-learning, provide a way for the agent to learn and make decisions based on estimates of the action values. These methods enable the agent to navigate the environment, explore different actions, and converge to an optimal policy

that maximizes the expected future rewards.

2.2.2 Policy-based Reinforcement Learning

$$\bigtriangledown_\theta \log \pi(a_t|s_t;\theta) R_t \tag{2.3}$$

Policy-based reinforcement learning (RL) methods update the policy directly by adjusting its parameters θ using gradient ascent on the expected return $E[R_t]$. In contrast to value-based methods, which estimate action values and make decisions based on those estimates, policy-based methods focus on improving the policy itself.

One popular policy-based RL algorithm is REINFORCE. In REINFORCE, the agent interacts with the environment, collecting trajectories of states, actions, and rewards. Trajectories represent sequences of states and actions taken by the agent, along with the corresponding rewards received.

The policy parameters θ are updated using equation 2.3[7]. This step aims to increase the probability of actions that lead to higher rewards and decrease the probability of actions that lead to lower rewards, thereby improving the policy. By iteratively collecting trajectories, computing expected returns, and updating the policy parameters, REINFORCE seeks to find the optimal policy that maximizes the expected return over time.

Policy-based RL methods have advantages such as handling continuous action spaces naturally and learning stochastic policies. Continuous action spaces can be addressed without discretization or function approximation techniques, while stochastic policies can handle uncertainty and exploration.

However, policy-based methods can suffer from high variance in gradients, leading to instability and slow learning. Techniques like baseline subtraction and variance reduction methods, such as advantage estimation, can mitigate this issue. Overall, policy-based methods, including REINFORCE, offer a direct approach to updating policies, enabling them to handle continuous actions and learn stochastic policies, although they may require additional techniques to address high variance.

2.3 Actor Critic

In a RL problem, the agents are supposed to maximize their reward r. The policy network is called an Actor because it chooses the action and the estimated value function is known as critic as it evaluates the action taken and corrects it. The advantage of actor-critic method is that they require minimal computation in order to select actions. For example, in case of a continuous-valued action space (acceleration of the agent in our case), the method that learns the action values must search through the entire space of infinite set of possible actions to pick an action. This extensive computation can be avoided if the policy is explicitly stored or learned.

To address the high variance in gradient estimates and improve learning efficiency, actor-critic methods are often employed. In an actor-critic framework, there are two learning agents: the actor (policy) and the critic (value function). The actor is responsible for determining the actions to take based on the current policy, while the critic evaluates the quality of the actions and provides feedback to the actor.

The critic, typically implemented as a value function, estimates the expected return or the value associated with being in a particular state or state-action pair. It provides guidance to the actor on how to adjust the policy to improve performance. The actor takes this feedback from the critic into account when updating the policy parameters, allowing for a more informed and effective exploration of the action space.

The actor-critic approach combines the advantages of both policy-based and value-based methods. It leverages the policy-based updates to directly optimize the policy, while also utilizing the value-based updates to reduce variance and provide more stable and efficient learning. This two-agent setup enhances the learning process and enables the agent to improve its policy based on the feedback received from the critic.

2.4 Model-based Reinforcement Learning

Reinforcement learning (RL) methods can be broadly classified into two categories: model-based and model-free. These categories differ in how the agent interacts with the environment and learns to make decisions.

Model-based RL refers to algorithms where the agent tries to understand and create a model of the environment based on its interactions. The agent learns the dynamics of the environment, including transition probabilities and reward functions, and uses this model to plan its actions. The model can be learned from data collected during the agent's interactions with the environment or can be provided explicitly.

The key characteristic of model-based algorithms is that the agent uses the learned or provided model to make decisions. Given the model, the agent can perform planning to determine the best actions to take in order to maximize its reward. The agent considers the consequences of its actions and selects the action that is expected to yield the highest reward based on its model. This greedy approach focuses on maximizing immediate rewards without explicitly considering the uncertainty or exploration.

2.5 Model-free Reinforcement Learning

Model-free RL algorithms do not require an explicit model of the environment. Instead, they learn the consequences of their actions through experience. Model-free algorithms directly estimate the value function or the policy without explicitly modeling the environment dynamics. Examples of model-free algorithms include Q-learning and policy gradient methods.

In model-free RL, the agent interacts with the environment, observes the states and rewards, and updates its value function or policy based on the observed outcomes. The agent explores different actions and their consequences by interacting with the environment multiple times. It updates its policy to maximize the expected cumulative reward over time. Model-free algorithms use methods like Q-learning, where the agent estimates the

values of state-action pairs, or policy gradient methods, where the agent directly learns a parameterized policy.

The key difference between model-based and model-free RL lies in how they handle decision-making. In model-based RL, the agent makes decisions based on a learned or provided model of the environment. It is greedy in selecting actions that yield the maximum expected reward, taking into account the consequences of its actions. In contrast, model-free algorithms do not have an explicit model and instead learn the optimal policy or value function through repeated interactions. They adjust their policy based on the observed outcomes, aiming to optimize long-term rewards.

Soft Actor-Critic is a model-free deep actor-critic algorithm where the agent aims to maximize the expected reward while also maximizing the entropy of the actions taken, which helps in faster exploration and convergence. Entropy, in the context of SAC, refers to the uncertainty or randomness in the agent's policy. It measures the amount of information or unpredictability in the agent's action distribution.

The objective of SAC is to maximize the expected cumulative reward while also maximizing the entropy of the policy. By maximizing entropy, SAC encourages exploration and prevents the policy from becoming too deterministic or overly focused on exploiting a single optimal action.

Chapter 3

Related Work

In traditional traffic simulation methods like SUMO and AIMSUN, the Car-Following Model (CFM) is utilized to simulate vehicle interactions. CFM takes into account several key features, including safety distance and the relative velocity between adjacent vehicles in the same lane. The safety distance is crucial for maintaining a safe following distance and avoiding collisions, while the relative velocity influences how a vehicle adjusts its speed or acceleration to ensure a consistent traffic flow. CFM models may also consider lane-changing behavior and incorporate driver-specific parameters like desired speed and reaction time to capture variations in driving behavior. When it comes to calibrating a specific CFM (Car Follow Model), heuristic search algorithms like random search, Tabu search, and genetic algorithm [9] are commonly employed to identify the optimal parameters for the CFM system.

One of the pioneering works in the field of neural control is ALVINN (Autonomous Land Vehicle In a Neural Network). ALVINN [10], developed in the late 1980s, employed a neural network-based approach for autonomous driving tasks. Building upon this foundation, NVIDIA emerged as one of the early adopters of ALVINN by incorporating it into their deep neural network called PilotNet. NVIDIA's PilotNet [11] was specifically designed for lane-keeping applications and utilized supervised learning techniques. The training data for PilotNet consisted of an extensive 72-hour collection of human driving recordings. By

leveraging this training data, NVIDIA successfully trained PilotNet to effectively perform lane-keeping tasks using deep neural networks, further advancing the field of neural control in autonomous driving.

The early works have deeply influenced the development of various deep-learning techniques for autonomous vehicle control. Among these techniques, imitation learning has emerged as a widely favored approach. Notably, researchers such as Zhang et al [12]. and Pan et al. [13] have made notable advancements in this area by extending the Dataset Aggregation (DAgger) imitation algorithm to the specific domain of autonomous driving. Their studies effectively demonstrated that autonomous vehicle control can be successfully learned from visual inputs, underscoring the immense potential of vision-based approaches in this field.

Since then there have been works focusing on the implementation of a reinforcement learning (RL) based approach to optimize driving comfort through longitudinal velocity control [14]. This work involved utilizing RL algorithms to learn an optimal control policy that considers factors such as acceleration, braking, and vehicle dynamics to provide a smooth and comfortable driving experience. The work used a Soft Actor-Critic network and an online mode of training. The agent trained exhibited capabilities to accelerate with a minimal jerk and maintain the speed limit. However, it did not consider the car-follow behavior which demands a complex observation space and reward function.

While there are online Actor-Critic approaches, some works have focused on improving safety and comfort in driving by employing offline reinforcement learning algorithms, such as Deep Deterministic Policy Gradient (DDPG) networks, trained with NGSIM data [15]. These approaches aimed to optimize driving behavior by considering variables like Time-to-Collision (TTC), headway, and jerk, with the goal of ensuring a safe and comfortable driving experience. One notable consideration in these works is the use of DDPG as an offline RL algorithm. DDPG networks learn an optimal control policy by leveraging historical data, allowing for the optimization of driving behavior even in scenarios where real-time interaction with the environment is not feasible. This offline training approach

provides an effective means to train the network using available data without requiring online interactions. However, a common limitation observed in these studies is the evaluation methodology, which often relies on mathematical models. While mathematical models offer valuable insights, they may not fully capture the intricacies of simulation environments, where real-world dynamics and complexities come into play.

Another notable work in the field involves leveraging Inverse Reinforcement Learning (IRL) and expert trajectory data to imitate human behavior [1]. The objective is to learn driving policies that mimic human decision-making processes. By utilizing IRL, this approach aims to understand the underlying preferences and intentions of expert drivers by observing their trajectories. This work takes into consideration two crucial factors: the lead vehicle gap and traffic signals. The lead vehicle gap plays a significant role in maintaining safe distances, while the consideration of traffic signals allows the model to adapt its behavior based on the current traffic conditions. The state space employed in this work includes various parameters to capture the driving context, such as the length of the current lane, speed limit, phase of the traffic light, velocity, position in the lane, and distance to the traffic light. These state variables collectively provide the necessary information for the learning algorithm to make informed decisions. Interestingly, unlike previous works that focused on learning acceleration, this model specifically learns the speed as the desired control output. This shift in focus highlights a different approach, emphasizing the importance of speed control to imitate human driving behavior accurately. By combining IRL, expert trajectory data, and a comprehensive state space representation, this work strives to capture the intricacies of human-like driving behavior. However, it might not be sufficient to generate a diverse traffic flow as the sample space is very small and it is hard to determine whether the RL agent has generalized the reward function despite the work showcasing better performances by imitating the expert trajectories.

Imitation learning-based approaches have made significant advancements in the realm of traffic simulation [1]. However, their effectiveness diminishes when deployed in environments that differ from their training distribution [16]. While these driving models,

built using supervised techniques, are typically evaluated based on performance metrics on pre-collected validation datasets, it is crucial to recognize that low prediction error on offline testing does not necessarily guarantee high-quality driving performance [17]. Despite showcasing promising results during closed-loop testing in naturalistic driving scenarios, imitation learning models often experience performance degradation due to distributional shift [18], the presence of unpredictable road users [19], or causal confusion [20] when confronted with diverse driving scenarios. These limitations highlight the challenges of deploying imitation learning approaches in real-world autonomous driving applications, necessitating the exploration of more robust and adaptable methodologies.

However, Reinforcement learning (RL) approaches offer notable advantages for longitudinal velocity control. RL algorithms have the capability to learn general driving rules that can adapt to new and varying environments. This has been demonstrated through numerous successful applications of RL in the autonomous vehicle longitudinal control [14]. The suitability of RL for longitudinal control is attributed to the ability to learn from low-dimensional observations such as relative distances and velocities. This mitigates the sample-efficiency problem commonly associated with RL. Additionally, defining the reward function for RL in the longitudinal control scenario is comparatively easier, often relying on safety distances to vehicles ahead. These factors contribute to the effectiveness and appeal of RL in autonomous vehicle longitudinal control, showcasing its potential for advancing autonomous driving capabilities. For these reasons, we focus on longitudinal velocity control by combining safety, comfort, efficiency, and speed constraints.

Chapter 4

Approach

In order to develop a velocity control strategy for a system with a continuous variable like acceleration, the Soft-Actor Critic (SAC) algorithm was chosen for the task. The SAC algorithm is a reinforcement learning technique that has been shown to achieve state-of-the-art performance on continuous control tasks.

In this section, we will describe the approach taken to learn the velocity control strategy using the SAC algorithm. We will start by outlining the main steps involved in the process, which include defining the problem, selecting appropriate reward functions, designing the neural network architecture, and training the model. The implementation is made available through GitHub [1].

Firstly, we defined the problem by specifying the task and the environment in which it will be performed. We then selected appropriate reward functions that would incentivize the agent to learn the desired behavior. Next, we designed a neural network architecture that could learn from the input data and output the appropriate control action. Finally, we trained the model using the collected data and iteratively improved its performance.

Overall, the SAC algorithm proved to be a powerful tool for learning a velocity control strategy in a system with a continuous variable like acceleration.

[1] `https://github.com/latchukarthick98/realisitc_speed_control`

4.1 State and Action

For our specific velocity control problem, the state is defined by the following variables:

- **Current velocity (v):** The current speed of ego vehicle.
- **Current acceleration (a):** The current acceleration of the ego vehicle.
- **Previous acceleration (a_{prev}):** The acceleration of the ego vehicle at the previous time step.
- **Vehicle gap (gap):** The distance between the leading and ego vehicles.
- **Relative speed (v_{rel}):** The difference between the velocity of the ego vehicle and the velocity of the leading vehicle.

In the velocity control problem described, the action determines the desired acceleration for the next time step.

This means that the agent receives feedback on its performance based on how well it can control the acceleration to achieve the desired velocity while taking into account the previous acceleration of the ego vehicle, the acceleration of the leading vehicle, the gap between both vehicles and the relative speed. By optimizing the reward function to incentivize the agent to reach the desired velocity quickly and smoothly while maintaining a safe distance from the leading vehicle, we can train the agent to reliably control the system's acceleration and achieve the desired velocity.

4.2 Simulation Setup

The simulation environment used in this project is the Simulation of Urban Mobility (SUMO), which is a widely used framework for traffic simulations. In this particular simulation, there are around 200 vehicles in addition to the ego vehicle. These vehicles are programmed to follow specific traffic rules, such as stopping at red lights and following speed limits, and behave based on the default car-follow model (Krauss). The learning

agent vehicle's specification has been shown in 4.1. The agent in the simulation has to handle two different situations: one with a leading vehicle and another without one. These situations require the agent to use different strategies to navigate the traffic. To avoid any potential impact of sequence, events in the simulation were randomly shuffled. Shuffling the sequence of events in the simulation is a technique used to mitigate any potential impact or bias caused by the order in which the events occur. Reinforcement learning algorithms often learn from sequential data, where the order of events can influence the learning process. By randomly shuffling the events, we ensure that the learning algorithm is exposed to a diverse and varied sequence of events, reducing the risk of it overfitting specific patterns or dependencies in the original sequence. This allowed for a more accurate representation of real-life traffic scenarios, where events can occur in any order. Overall, the SUMO simulation environment and setup allow for a comprehensive evaluation of the agent's performance in various traffic scenarios.

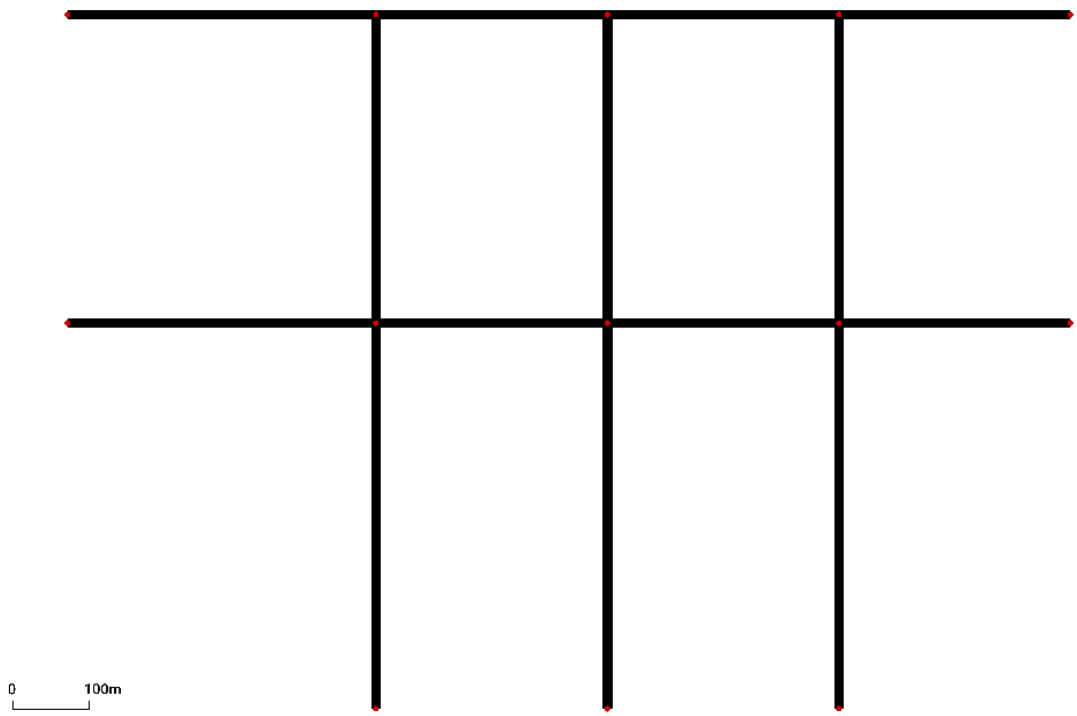

Figure 4.1: Road Network used for Training

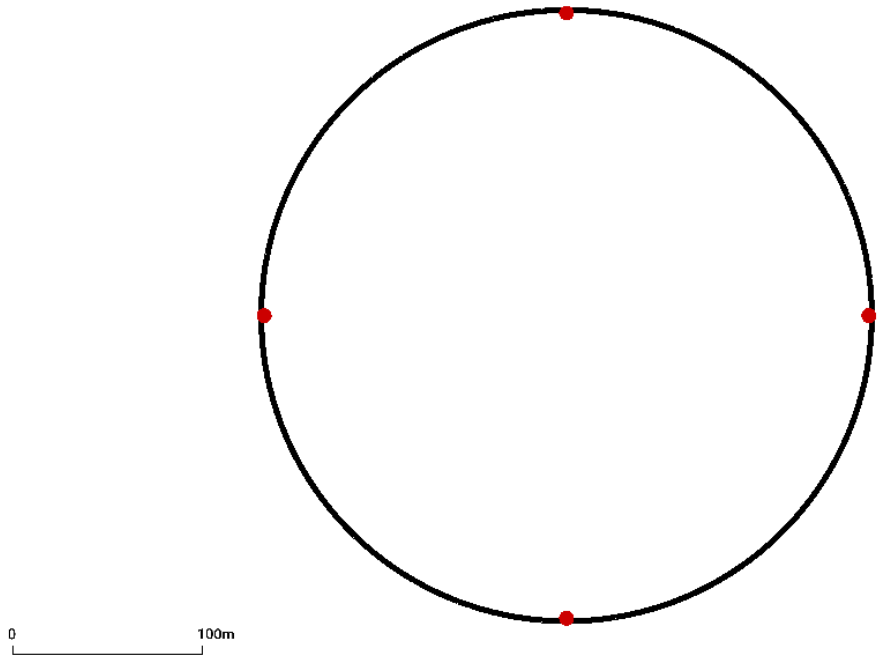

Figure 4.2: Road Network used for Testing the car follow behavior

4.3 Reward Function

The reward function used in the velocity control task consists of four components: the desired speed reward, the time-to-collision (TTC) reward, the headway reward, and the comfort reward. Each of these components is designed to incentivize the agent to learn a specific aspect of the desired behavior.

4.3.1 Desired Speed Reward

The desired speed reward is a simple reward that encourages the agent to reach the desired speed as quickly and smoothly as possible. The agent is encouraged to drive towards the desired speed. Without rewarding the speed, the agent might fall prey to the zero rewards imposed by comfort and TTC rewards, as standing still is much safer than driving. It is calculated as the negative absolute difference between the current velocity and the desired velocity.

The desired speed reward $r_{forward}$ is defined as the ratio of the absolute difference

Figure 4.3: Road Network of Shorewood neighborhood used for Testing

between the desired speed and current speed to the desired speed which is shown in equation 4.1.

$$\begin{aligned} r_{forward} &= \frac{|v_{desired} - v_{current}|}{v_{desired}} \\ &= lmap(r_{forward}, [1, 0], [0, 1]) \end{aligned} \tag{4.1}$$

The reward $r_{forward}$ will be approaching zero when the current speed is approaching the desired speed and approaching one when the current speed is zero. But, with this kind of reward, the agent might not even move as the goal is to maximize the overall reward. So, a linear map function (equation 4.9) that maps the reward in the interval $[0, 1]$ to $[1, 0]$ is applied to the equation 4.1. It is also ensured that $r_{forward}$ is clipped in the range $[0, 1]$. This also helps in normalizing the overall reward.

4.3.2 Time-to-Collision Reward

Time-to-Collision is calculated by dividing the gap between ego and leading vehicles with the relative velocity (v_{rel}) of the ego and lead vehicles (equation 4.2). The TTC reward incentivizes the agent to maintain a safe distance from the leading vehicle. It is calculated with the equation 4.3. A high TTC reward indicates that the ego vehicle is far from the leading vehicle, while a low TTC reward indicates that the ego vehicle is close to the leading vehicle and needs to slow down to avoid a collision.

$$TTC = -\frac{gap}{v_{rel}} \tag{4.2}$$

$$r_{ttc} = \begin{cases} \log(\frac{TTC}{4}) & 0 \leq TTC \leq 4 \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

The TTC reward will approach negative infinity as the TTC values approach zero, hence penalizing heavily for near-miss collision scenarios. The upper bound of 4 for TTC is chosen as it resulted in better overall performance [15].

4.3.3 Headway Reward

The headway reward also encourages the agent to maintain a safe distance from the leading vehicle. It is calculated as log-normal distribution (equation 4.5) of the headway distance, which is the distance between the leading vehicle and the ego vehicle divided by the current velocity of the ego vehicle (equation 4.4). A high headway reward indicates that the ego vehicle is far from the leading vehicle, while a low headway reward indicates that the ego vehicle is too close and needs to slow down.

$$headway = \frac{gap}{v_{current}} \tag{4.4}$$

$$r_{headway} = f_{lognorm}(headway|\mu = 0.4426, \sigma = 0.4365) \tag{4.5}$$

The log-normal distribution along with the parameters specified in equation 4.5 is used to encourage the agent to maintain a constant headway. The distribution corresponds to the largest value (about 0.65) when the headway is about 1.2 seconds and the lower value for too long or too short headway values [15].

4.3.4 Comfort Reward

Finally, the comforting reward incentivizes the agent to control the acceleration smoothly. It is calculated as the negative squared difference between the current acceleration and the previous acceleration of the ego vehicle (equations 4.6 and 4.7). A low comfort reward indicates that the acceleration is changing quickly, while a high comfort reward indicates that the acceleration is changing smoothly.

$$comfort = abs(a_{prev} - a_{current}) \tag{4.6}$$

$$r_{comfort} = \frac{-comfort^2}{3600} \tag{4.7}$$

4.3.5 Combined Rewards

By combining these four rewards, the agent is incentivized to reach the desired speed quickly and smoothly while maintaining a safe distance from the leading vehicle and controlling the acceleration smoothly. This reward function has been shown to effectively train the agent to learn the desired velocity control strategy in the simulation environment.

$$R = \omega_1 * r_{forward} + \omega_2 * r_{comfort} + \omega_3 * r_{headway} + \omega_4 * r_{ttc} - \omega_5 * r_{collision} \tag{4.8}$$

In the above equation 4.8, $r_{collision}$ refers to the heavy penalty that is imposed when the vehicle collides. The weights are $\omega_1, \omega_2, \omega_3, \omega_4, \omega_5 = 1$ for this task, which means each feature is equally regarded.

4.3.6 Reward Normalization

It is known that the optimal policy is invariant by scaling and shifting rewards. Since the rewards must be bounded for stable training, we chose to normalize them in the range $[0, 1]$ by convention. Normalizing the reward has been proven to be practically beneficial in Deep Reinforcement Learning as the same model can be used for various tasks. We forbid negative rewards, since they may encourage the agent to prefer terminating an episode early by colliding rather than suffering a negative return if no satisfying trajectory can be found. Normalization is done through a linear map function (equation 4.9).

$$lmap(val, [x_{min}, x_{max}], [y_{min}, y_{max}]) = y_{min} + \frac{(val - x_{min}) * (y_{max} - y_{min})}{(x_{max} - x_{min})} \tag{4.9}$$

4.4 Action Scaling

In the context of vehicle control, actions refer to the actuation of vehicle pedals, specifically the accelerator and brake pedals. The accelerator pedal is associated with positive actions, indicating an intention to accelerate the vehicle. On the other hand, the brake pedal is associated with negative actions, signifying the intention to apply braking force and slow down or stop the vehicle.

$$a = \frac{(F_{drive} - F_{drag})}{m} \tag{4.10}$$

$$F_{drive} = \frac{P_{engine}}{v} + m * g + F_{friction} \tag{4.11}$$

$$F_{drag} = 0.5 * C_d * \rho * A * v^2 \tag{4.12}$$

where:

F_{drive} Driving Force

F_drag Air Drag Force

$m =$ Mass of the vehicle

$P_{engine} =$ Engine power

$v =$ Velocity

$g =$ Acceleration due to gravity

$A =$ Car frontal area

$C_d =$ Drag coefficient

$\rho =$ Air Density

The acceleration of the vehicle is influenced by its current speed. The relationship between speed and acceleration can be determined based on the vehicle's technical data or specifications which is applied to equation 4.10. These specifications provide information about the maximum and minimum acceleration values at different speeds. By considering the current speed, the controller can determine the appropriate level of acceleration within the speed-dependent limits.

When the action is set to 0, it implies that neither pedal is actuated. In this scenario, the vehicle decelerates due to the simulation of driving resistance. This deceleration accounts for factors such as aerodynamic drag, rolling resistance, and other forces that act to slow down the vehicle in the absence of pedal actuation. By incorporating this deceleration component, the controller ensures that the vehicle gradually slows down when no pedal is being pressed. Fig.4.4 shows a diagrammatic representation of action scaling which is inspired by [14].

In summary, in vehicle control, actions correspond to the actuation of the accelerator and brake pedals. Positive actions represent acceleration, negative actions represent

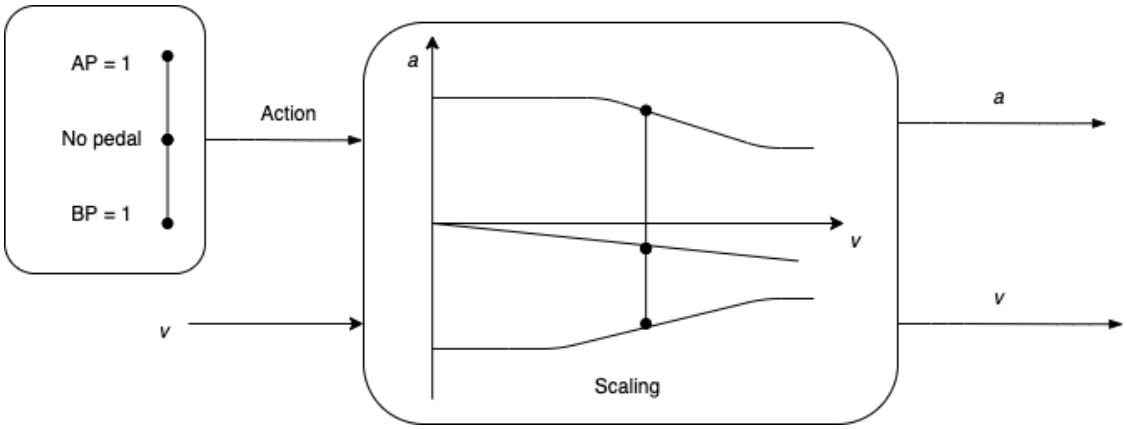

Figure 4.4: Action Scaling based on the current speed and action input. AP = Acceleration Pedal, BP = Break Pedal, a = acceleration, v = velocity.

braking, and 0 action indicates no actuation of either pedal. The relationship between speed and acceleration is determined by the vehicle's technical data, and the controller applies speed-dependent limits to the acceleration. When no pedal is actuated, the vehicle decelerates based on the simulation of driving resistance.

Table 4.1: Vehicle Specification

Attribute	**Value**
Acceleration Limits	$[-3.0, 3.0]\ m/s^2$
Velocity Limits	$[0, 50]\ m/s$
Mass	$1443\ kg$
Frontal Area	$2.38\ m^2$
Emergency Breaking	$-4.5\ m/s^2$

Algorithm 1 Velocity Control based on Soft Actor-Critic (SAC)

Require:

Initial state s
Learning rate α
Discount factor γ
Target update rate τ
Entropy coefficient β
Replay buffer size N
Batch size B

Ensure:

Trained actor network π
Initialization:
Initialize actor network π with random weights θ_π
Initialize critic network Q with random weights θ_Q
Initialize target networks π' and Q' with weights $\theta_{\pi'} = \theta_\pi$ and $\theta_{Q'} = \theta_Q$
Replay Buffer:
Create an empty replay buffer R of size N
Training Loop:
repeat
    Sample an action $a_t \sim \pi(s_t, \theta_\pi)$ from the actor network π based on the current state s_t
    Execute the action a_t in the environment and observe the next state s_{t+1}, reward r_t, and termination signal *done*
    Store the experience $(s_t, a_t, r_t, s_{t+1}, done)$ in the replay buffer R
    if replay buffer size $> B$ **then**
        Sample a batch of experiences $(s_i, a_i, r_i, s_{i+1}, done_i)$ from the replay buffer R
        Compute the target value y_i for each sample:
        $y_i = r_i + \gamma(1 - done_i)Q'(s_{i+1}, \pi'(s_{i+1}, \theta_{\pi'}), \theta_{Q'})$
        Update the critic network by minimizing the mean squared Bellman error:
        $\theta_Q = \theta_Q - \alpha \nabla_{\theta_Q} \left(\frac{1}{B} \sum_i (y_i - Q(s_i, a_i, \theta_Q))^2 \right)$
        Update the actor network by maximizing the expected Q-values and the entropy regularization term:
        $\theta_\pi = \theta_\pi + \alpha \nabla_{\theta_\pi} \left(\frac{1}{B} \sum_i Q(s_i, \pi(s_i, \theta_\pi), \theta_Q) - \beta \nabla_{\theta_\pi} H(\pi(s_i, \theta_\pi)) \right)$
        Update the target networks by slowly blending the main network weights:
        $\theta_{\pi'} = \tau\theta_\pi + (1 - \tau)\theta_{\pi'}$
        $\theta_{Q'} = \tau\theta_Q + (1 - \tau)\theta_{Q'}$
    end if
until convergence or maximum number of episodes reached

4.5 Collision Warning and Avoidance

The safety reward system, despite penalizing situations with small time-to-collision (TTC) values, may not completely prevent the agent from taking unsafe actions that could result in collisions, even after the training has converged. In such cases, collisions lead to the simulation being reset. A collision avoidance strategy is employed both during the training and testing phases to enhance the agent's ability to avoid collisions. This approach also aids the agent in recognizing and responding appropriately when the leading vehicle abruptly comes to a stop.

$$d_{\text{safe}} = \frac{v^2}{2 * a} - \frac{v_{lead}^2}{2 * a_{lead}} + v * R + C \tag{4.13}$$

where:

$v =$ Velocity of the ego vehicle

$a =$ Maximum deceleration of the ego vehicle

$v_{lead} =$ Velocity of the lead vehicle

$a_{lead} =$ Maximum deceleration of the lead vehicle

$R =$ Reaction time

$C =$ Confidence distance, $C = 2$ in this implementation

The safe distance(d_{safe}) formulation presented in equation 4.13 is based on a kinematic stop distance proposed by [4]. The intuition is that when the car in front hits a hard break and suddenly stops, the ego vehicle has no chance of collision if it maintains a distance larger than d_{safe}. The agent takes a hard brake (emergency breaking mentioned in Table 4.1) when its distance from the lead vehicle is less than the d_{safe}. It also acts as a teacher for the agent during training to correct itself in the event of taking an unsafe action.

4.6 Network Architecture

For the velocity control task, the network architecture of a Soft Actor-Critic (SAC) network is tailored to handle continuous action spaces and predict the optimal control inputs to regulate the vehicle's speed. Here's an adapted description of the network architecture:

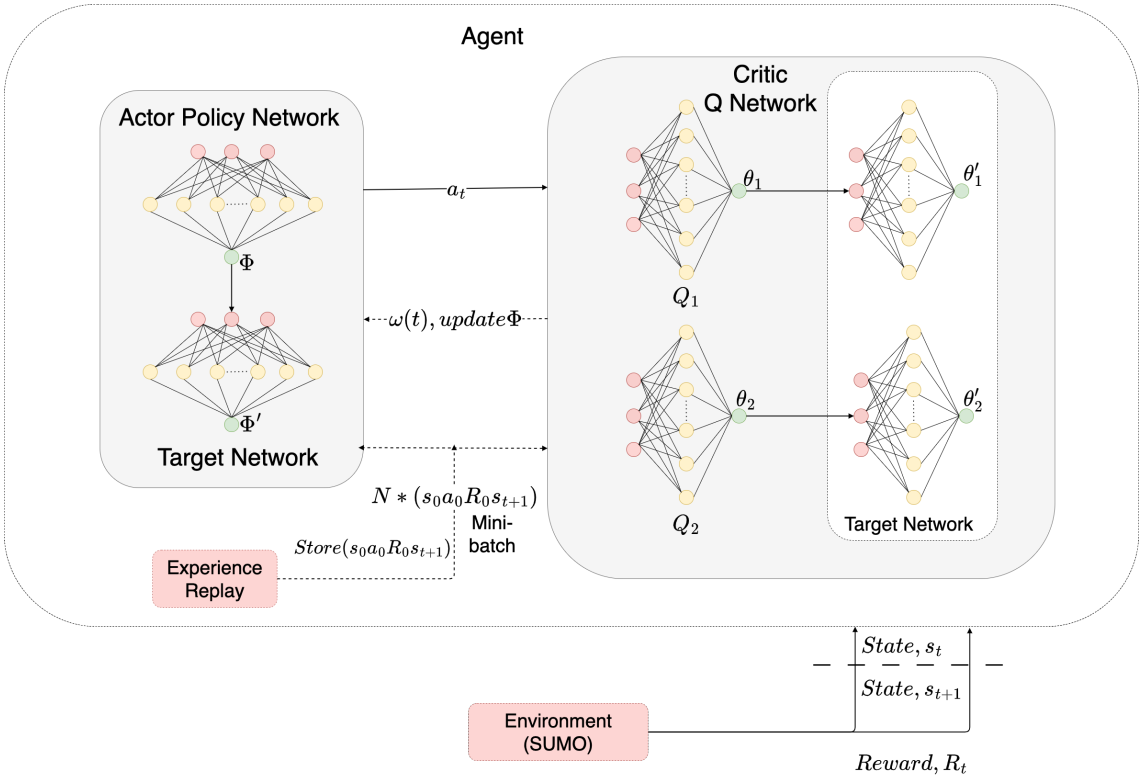

Figure 4.5: Soft Actor-Critic Network

Table 4.2: Hyperparameters for Soft Actor-Critic (SAC) Algorithm

Hyperparameter	**Value**
Optimizer	Adam [21]
Learning rate	0.0003
Discount factor (γ)	0.99
Replay buffer size	1,000,000
Number of hidden layers (all networks)	2
Number of hidden units per layer	256
Optimization batch size	256
Target entropy	$-\dim(A)$
Activation function	ReLU
Soft update factor	0.01
Target update interval	1
Gradient steps	1

4.6.1 Actor Network

The actor network takes the current state of the vehicle as input and outputs the control signal or action that determines the desired velocity. The architecture of the actor network typically consists of one or more hidden layers followed by an output layer.

The hidden layers of the actor network can utilize fully connected (dense) layers as the input includes sensor readings such as velocity, gap, and acceleration. There are 256 units in the hidden layer of the actor network. The actor network and its interaction with the critic network is shown in Figure 4.5

The activation function used between the layers is ReLU which introduces non-linearity into the network. These activation functions help the network model complex relationships and extract relevant features from the input state.

The output layer of the actor network is typically a single neuron that outputs the desired vehicle action (accelerate/decelerate). As this is a continuous control task, the output neuron employs a linear activation function to directly provide the desired action.

4.6.2 Critic Network

The critic network estimates the value function, which evaluates the expected return or utility of a given state and the associated control signal. It takes the current state and the selected action as inputs and outputs a scalar value representing the estimated value of that state-action pair. This value is used to assess the quality of the chosen action by the actor network. The architecture of the critic network is similar to the actor network, with hidden layers and an output layer.

The hidden layers of the critic network use fully connected layers. These layers extract relevant features to estimate the state-action value. The activation function, ReLU is employed between the layers. The critic network's hidden layer has about 256 units.

The output layer of the critic network consists of a single neuron that provides the estimated state-action value. This value is used to compute the advantage function, which represents the advantage of taking a specific action in a given state compared to the average value of that state.

4.6.3 Value Network

Similar to the critic network, SAC includes a value network to estimate the state value function, which measures the expected return from a given state. However, for the vehicle velocity control task, the value network is not necessary, as the focus is primarily on the actor and critic networks.

In summary, for a vehicle velocity control task, the SAC network architecture comprises an actor network that outputs the desired action, a critic network that estimates the state-action value, and a value network for state-value estimation. The actor network determines the control inputs, the critic network evaluates their quality, and the value network estimates the expected return. This architecture enables the SAC algorithm to learn an optimal policy for regulating the vehicle's velocity based on the given state information.

4.7 Training the agent

The Actor-Critic network used for the training has been depicted in Fig.4.5. The hyper-parameters used for the training are listed in Table 4.2. A detailed algorithm on how the SAC-based agent was trained has been presented in Algorithm 1. In the training of a SAC agent for velocity control in the SUMO environment, the agent undergoes a learning process of over 1 million steps with approximately 1000 episodes where each episode has a maximum of 1000 steps as a time horizon. A time horizon is a hard limit on time steps to stop training the same episode and reset the environment for a new episode. By doing this we eliminate the chances for the model being stuck due to a huge negative cumulative reward and fail to improve. During each episode, the agent interacts with the SUMO environment by observing the current state, selecting an action (accelerator or brake pedal actuation), and receiving a reward based on the action and resulting state. The agent's training involves collecting data, storing transitions in a replay buffer, and periodically updating its policy and value function networks through optimization. The objective is to find a policy that maximizes the expected cumulative reward while also balancing exploration and exploitation through the entropy term. By training for a large number of episodes, the SAC agent learns to control the vehicle's velocity effectively in the SUMO environment. The mean rewards obtained during training over different episodes have been presented in Fig.4.6. It can be noted that the rewards started converging after $3\tilde{5}0$ episodes.

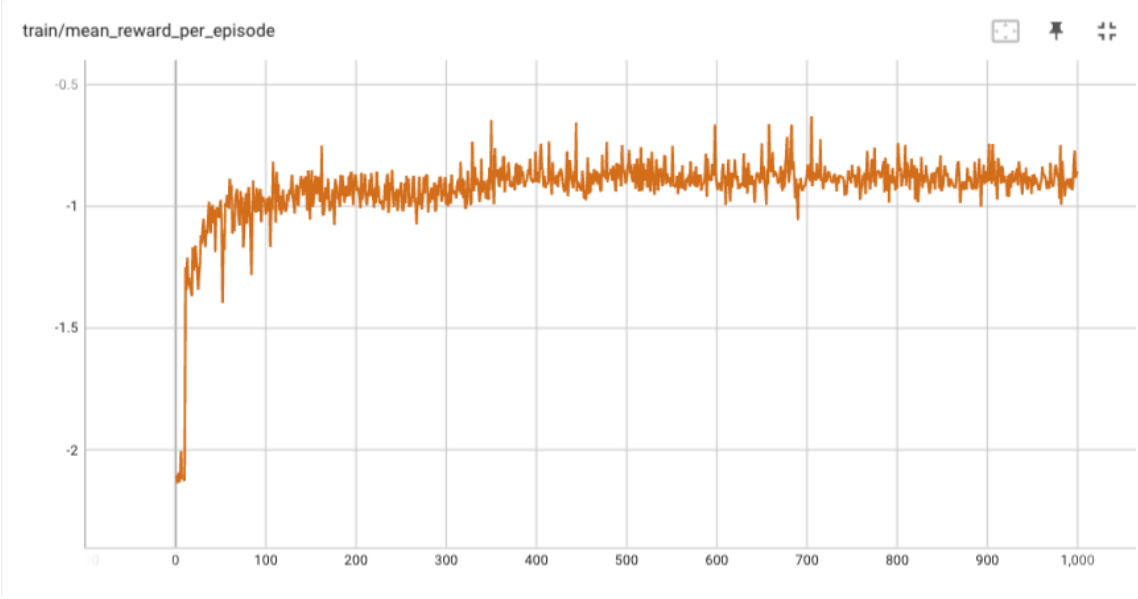

Figure 4.6: Mean reward of each episode during the training of SAC agent. $x - axis$ is episodes and $y - axis$ is mean reward per episode.

Chapter 5

Evaluation

This section aims to compare the car-following behavior simulated by the Krauss, IDM, and SAC-based models, showcasing their respective abilities to safely, efficiently, and comfortably follow the leading vehicle.

5.1 Driving Comfort

In evaluating driving comfort during car-following events, one common metric used is the jerk, which represents the rate of change of acceleration. A lower jerk value indicates smoother and more comfortable driving behavior.

To calculate the jerk, measurements are taken at each step of the car-following event, capturing the acceleration changes experienced by the following vehicle. By comparing the jerk values generated by the Krauss, IDM, and SAC-based models, and a baseline model (referred to as DDPG), we can gain insights into the comfort levels achieved by each model.

The cumulative distribution of jerk values during the car-following events simulated by different models is presented in Fig.5.1. The mean jerk values for the Krauss model, IDM model, SAC-based model (our implementation) and DDPG are $5.45m/s^3$, $0.67m/s^3$, $2.11m/s^3$, and $6.11m/s^3$ respectively.

Comparing these values, we can observe that the IDM model achieves the lowest mean

jerk, indicating a smoother and more comfortable car-following behavior. The SAC-based model also performs well, with a higher mean jerk than IDM but still providing relatively comfortable driving. On the other hand, the Krauss model and DDPG model have higher mean jerk values, suggesting less comfortable driving experiences.

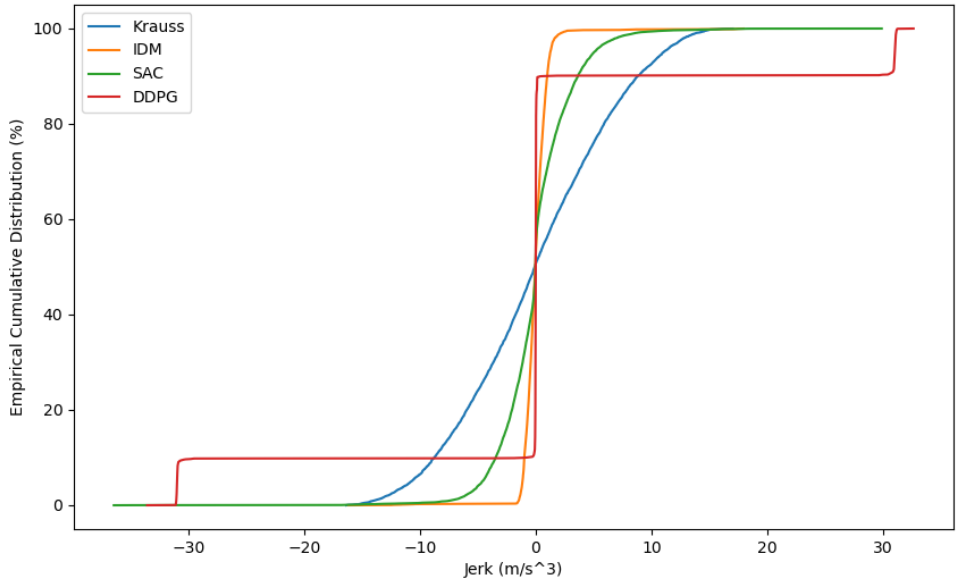

Figure 5.1: Comparison of Jerk from different models. SAC is our implementation.

5.2 Safe Driving

Driving safety is a crucial aspect of evaluating car-following behavior, and one commonly used metric is Time to Collision (TTC). TTC measures the estimated time it would take for a following vehicle to collide with the leading vehicle if their current trajectories were maintained. Higher TTC values indicate safer distances between vehicles, reducing the risk of collisions.

A cumulative distribution of TTC for the car-following event by all four models in comparison is shown in Fig.5.2. The mean TTC values for the models are as follows: the Krauss model has a mean TTC of 33.5 seconds, the IDM model has a mean TTC of 26.8

seconds, the SAC model has a mean TTC of 36.7 seconds and the DDPG model has a mean TTC of 13.6 seconds.

This means that the SAC model exhibits the highest mean TTC value, indicating safer following distances and a reduced risk of collisions during car-following events. The IDM model follows with a slightly lower mean TTC, while the Krauss model has a moderate mean TTC. The DDPG model, however, demonstrates the lowest mean TTC, suggesting closer following distances and potentially higher collision risks.

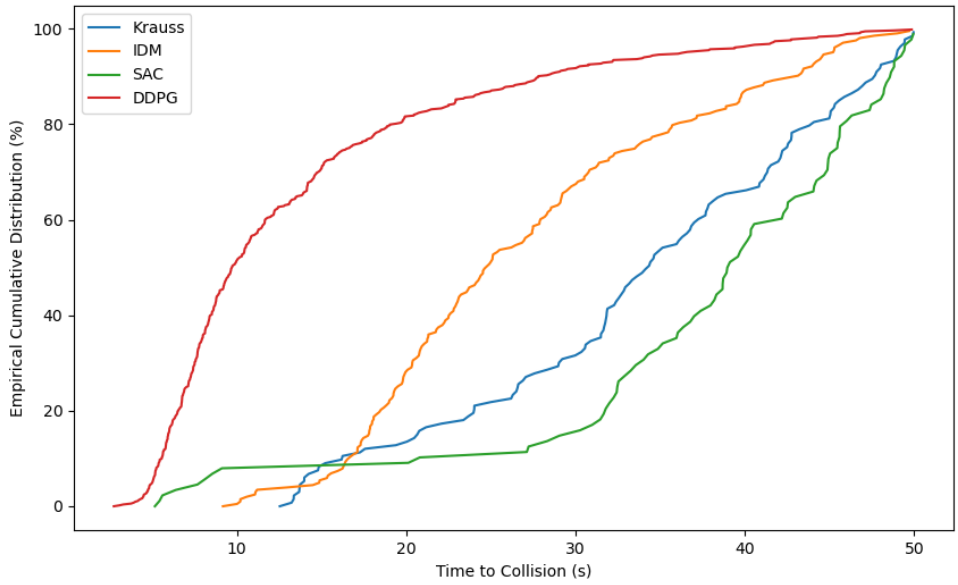

Figure 5.2: Comparison of Time to Collision values from different models. SAC is our implementation.

5.3 Efficient driving

Driving efficiency, a key factor in evaluating car-following behavior, can be assessed using the concept of time headway. Time headway represents the time interval between the leading and following vehicles, indicating how closely vehicles are spaced on the road. A higher time headway signifies a more efficient and smoother traffic flow.

To evaluate driving efficiency, time headway is measured at each step of the simulation,

providing insights into the efficiency of car-following events. By comparing the mean time headway values for the Krauss, IDM, SAC, and DDPG models, we can assess their respective efficiency levels.

A cumulative distribution of time headway times exhibited four different car-follow models were compared in Fig.5.3. The mean time headway values for the models are as follows: the Krauss model has a mean time headway of 1.37 seconds, the IDM model has a mean time headway of 1.37 seconds, the SAC model has a mean time headway of 1.48 seconds, and the DDPG model has a mean time headway of 0.36 seconds.

These results indicate that both the Krauss and IDM models exhibit similar mean time headway values, suggesting comparable levels of driving efficiency. The SAC model demonstrates a slightly higher mean time headway, indicating a relatively more efficient traffic flow. In contrast, the DDPG model shows the lowest mean time headway, indicating a less efficient traffic flow with vehicles being closely spaced.

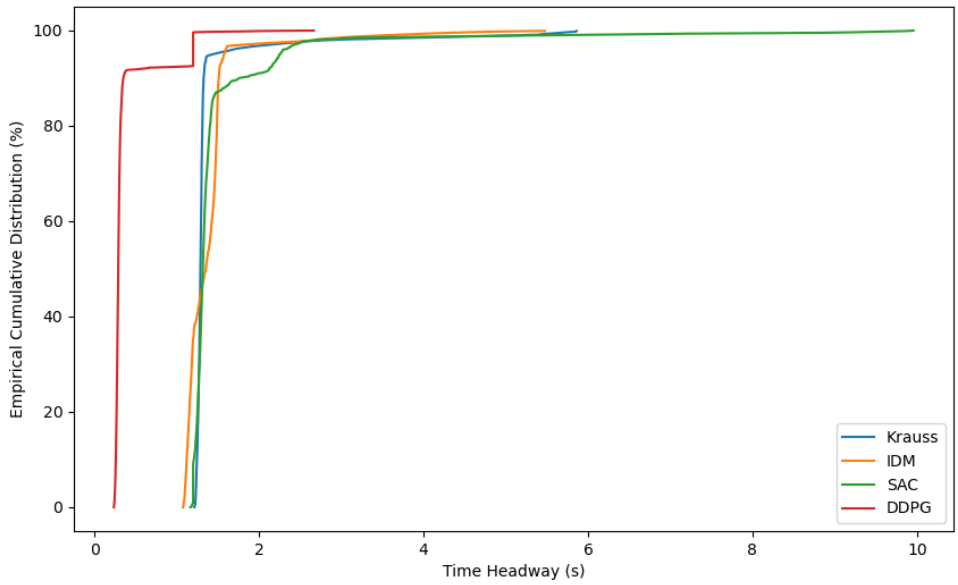

Figure 5.3: Comparison of Headway times from different models. SAC is our implementation.

5.4 Leader Speed vs Follower Speed

To give an illustration of how the following vehicle copes up with the leader in terms of speed, a sample car-follow event has been chosen to compare the speed of the leader versus the following vehicle with respect to the given time step as shown in Fig.5.4. It can be

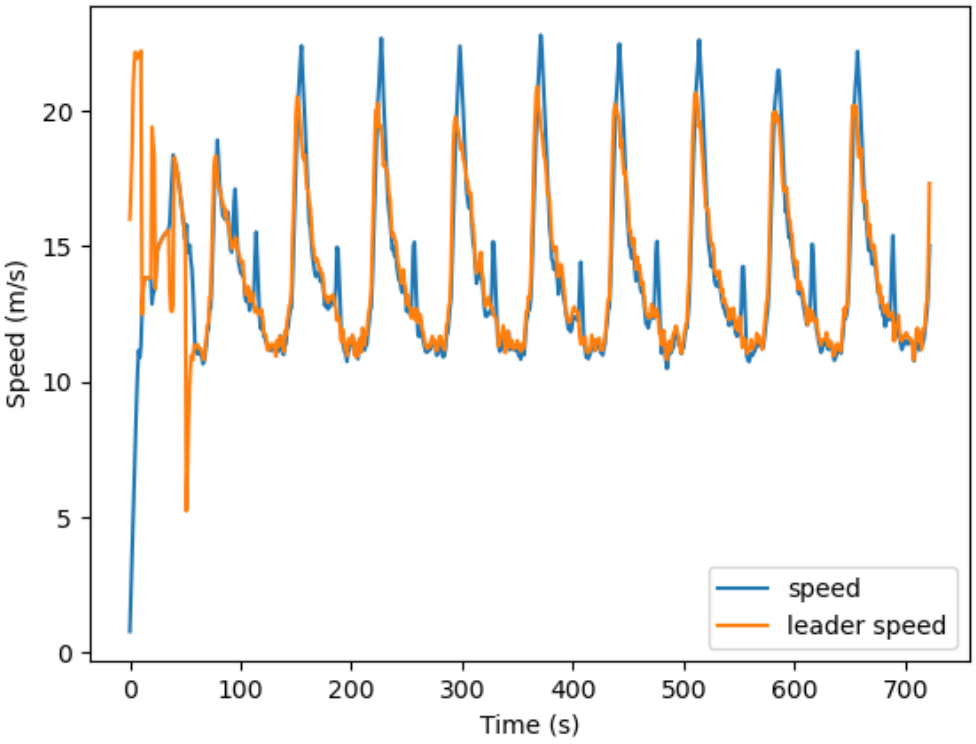

Figure 5.4: Leader vs Follower speed in a random car follow event

seen that the following vehicle tries to keep up the speed of the leader vehicle to maintain a safe gap. Though the follower goes a little higher speed than the leader sometimes, it maintains the speed of the leader most of the time. The spikes that appear to be higher than the leader's speed may be due to the follower trying to maintain an efficient headway time, rushing to close the distance.

5.5 Models in Comparison

5.5.1 Krauss Model

The Krauss car-following model [6], developed by Hermann Krauss, is the default car-following model in the SUMO traffic simulation framework. This model captures the behavior of individual vehicles in a traffic flow by considering factors such as desired time headway, actual time headway, and speed differences. It incorporates a stochastic term to account for random variations in acceleration. Widely used in traffic simulations, the Krauss model provides a simplified representation of vehicle interactions and is employed as the default car-following model in SUMO, enabling the study and analysis of traffic flow dynamics in various scenarios.

5.5.2 Intelligent Driver Model (IDM)

The Intelligent Driver Model (IDM) [5] is a car-following model commonly used in traffic simulations and transportation research. Developed by Treiber, Hennecke, and Helbing, the IDM aims to replicate human driving behavior by considering factors such as desired speed, desired time headway, and sensitivity to the speed and distance of the vehicle ahead. The model assumes that drivers strive to maintain a desired time headway while reaching their desired speed. It incorporates acceleration and deceleration terms based on these factors, including a comfortable deceleration term in response to a slower vehicle and an acceleration term reflecting the difference between desired and actual speeds. The IDM also considers safety distances that increase with decreasing speeds and sizes of the vehicle ahead.

5.5.3 DDPG model and its Implementation

DDPG, a model used for comparisons, has been reconstructed with the network architecture and hyperparameters specified in the paper [15] to support SUMO. However, it is important to note that the original implementation of the paper used trajectories extracted from NGSIM data and employed a mathematical model for vehicle simulation.

This means that the reconstructed DDPG might not be an exact replica of the original work. Consequently, the comparisons made between DDPG and other models may be affected by these differences in implementation. The variations in data source and simulation approach could potentially introduce discrepancies and impact the validity of the presented comparisons.

In summary, the SAC-based model demonstrates superiority in terms of safe, efficient, and comfortable driving behavior compared to the other three models. It achieves this through the following key points:

Safety The SAC-based model exhibits larger Time to Collision (TTC) values compared to the other models, indicating safer distances and a reduced risk of collisions during car-following events. It is also notable that the SAC based approach has fewer TTC values between the range 0 and 20 seconds and more TTC values between the range 40 and 50 seconds making it safer compared to other methods.

Efficiency The SAC-based model is capable of maintaining safe and efficient time headways between 1 second and 2 seconds. This ensures smoother traffic flow and better utilization of road capacity.

Comfort While the Intelligent Driver Model (IDM) excels in providing comfortable driving experiences, the SAC-based model still offers satisfactory levels of comfort, due to its smoother acceleration patterns and ability to follow the lead vehicle effectively.

Overall, the SAC-based model outperforms the other models, including Krauss and the baseline models, in terms of both safety and efficiency. Although IDM may offer better driving comfort, the SAC-based model strikes a balance between comfort, safety, and efficiency, making it a preferred choice for car-following simulations and traffic management applications.

Chapter 6

Conclusion and Future Work

In conclusion, we have presented a Soft Actor-Critic (SAC) based agent that effectively learns to control velocity by applying appropriate acceleration while prioritizing safety, comfort, and efficiency. The agent was trained using an online environment with SUMO as a simulation environment, which generates random traffic scenarios.

The performance of our model was compared against the Krauss, IDM, and DDPG models to evaluate its effectiveness. The results demonstrate that our model exhibits safe, efficient, and comfortable driving capabilities. While it may not outperform the IDM model in terms of comfort, it excels in safety and efficiency, outperforming both the Krauss model and DDPG model across all aspects.

An important aspect of our model is its ability to maintain speed with the lead vehicle, which is crucial for maintaining a safe distance and avoiding collisions. This finding highlights the effectiveness of the proposed approach in achieving safe driving behaviors.

Overall, the results indicate that reinforcement learning methods, such as the SAC algorithm, hold significant promise in handling diverse traffic simulations and effectively generalizing driving behaviors. Further research and improvements in the model can potentially enhance its comfort aspect while maintaining its superior safety and efficiency characteristics. Our study contributes to the growing body of knowledge in developing intelligent systems for traffic simulation.

The future works can be in one of the following aspects:

1. Test the performance of the SAC-based model using real-world traffic data by translating NGSIM data into SUMO traffic scenarios. This will allow for the validation of the model's ability to generalize and adapt to real-world traffic conditions.

2. Explore the potential of action scaling in the SAC-based model. Investigate whether scaling the actions can provide a way to test the model with vehicles that have different technical specifications without requiring separate training for each vehicle type. Action scaling can abstract the acceleration component, enabling the model to handle vehicles with varying performance characteristics.

3. Conduct experiments with different weights for the reward terms in the SAC-based model. By adjusting the weights assigned to different components of the reward function, such as comfort, energy efficiency, or safety, the model's behavior can be fine-tuned to prioritize specific objectives. Exploring different weight configurations can help find an optimal balance and uncover the trade-offs between different performance metrics.

Bibliography

[1] G. Zheng, H. Liu, K. Xu, and Z. Li, "Objective-aware traffic simulation via inverse reinforcement learning," May 2021. [Online]. Available: http://arxiv.org/abs/2105.09560.

[2] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," Jan. 2018. [Online]. Available: http://arxiv.org/abs/1801.01290.

[3] T. Haarnoja, A. Zhou, K. Hartikainen, *et al.*, "Soft actor-critic algorithms and applications," Dec. 2018. [Online]. Available: http://arxiv.org/abs/1812.05905.

[4] T. B. Wilson, W. Butler, D. V. Mcgehee, and T. A. Dingus, "Forward-looking collision warning system performance guidelines," 1997, pp. 701–725. [Online]. Available: https://www.jstor.org/stable/44731227.

[5] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations typeset using revt e x 1," 2000. [Online]. Available: http://www.theo2.physik.uni-stuttgart.de/treiber/,helbing/.

[6] S. Krauss, "Microscopic modeling of traffic flow: Investigation of collision free vehicle dynamics," Apr. 1998.

[7] Y. Li, "Deep reinforcement learning: An overview," Jan. 2017. arXiv: 1701.07274 [cs.LG].

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

[9] C. Osorio and V. Punzo, "Efficient calibration of microscopic car-following models for large-scale stochastic network simulators," en, *Trans. Res. Part B: Methodol.*, vol. 119, pp. 156–173, Jan. 2019.

[10] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Proceedings of the 1st International Conference on Neural Information Processing Systems*, ser. NIPS'88, Cambridge, MA, USA: MIT Press, 1988, pp. 305–313.

[11] M. Bojarski, D. Del Testa, D. Dworakowski, *et al.*, "End to end learning for self-driving cars," Apr. 2016. arXiv: 1604.07316 [cs.CV].

[12] J. Zhang and K. Cho, "Query-efficient imitation learning for end-to-end autonomous driving," May 2016. arXiv: 1605.06450 [cs.LG].

[13] Y. Pan, C.-A. Cheng, K. Saigol, *et al.*, "Agile autonomous driving using end-to-end deep imitation learning," Sep. 2017. arXiv: 1709.07174 [cs.RO].

[14] R. Liessner, J. Dohmen, C. Friebel, and B. Bäker, "Longicontrol: A reinforcement learning environment for longitudinal vehicle control real-world driving cycles view project energy management view project longicontrol: A reinforcement learning environment for longitudinal vehicle control a preprint," 2020. DOI: 10.13140/RG.2.2.26777.65123. [Online]. Available: https://www.researchgate.net/publication/342747549.

[15] M. Zhu, Y. Wang, Z. Pu, J. Hu, X. Wang, and R. Ke, "Safe, efficient, and comfortable velocity control based on reinforcement learning for autonomous driving," *Transportation Research Part C: Emerging Technologies*, vol. 117, Aug. 2020, ISSN: 0968090X. DOI: 10.1016/j.trc.2020.102662.

[16] F. Codevilla, E. Santana, A. M. López, and A. Gaidon, "Exploring the limitations of behavior cloning for autonomous driving," Apr. 2019. arXiv: 1904.08980 [cs.CV].

[17] F. Codevilla, A. M. López, V. Koltun, and A. Dosovitskiy, "On offline evaluation of vision-based driving models," Sep. 2018. arXiv: 1809.04843 [cs.CV].

[18] S. Ross, G. J. Gordon, and J. A. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," Nov. 2010. arXiv: 1011.0686 [cs.LG].

[19] S. Kuutti, S. Fallah, and R. Bowden, "Training adversarial agents to exploit weaknesses in deep control policies," Feb. 2020. arXiv: 2002.12078 [cs.LG].

[20] P. de Haan, D. Jayaraman, and S. Levine, "Causal confusion in imitation learning," May 2019. arXiv: 1905.11979 [cs.LG].

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," Dec. 2014. [Online]. Available: http://arxiv.org/abs/1412.6980.