Fakultät Verkehrswissenschaften „Friedrich List“

Institut für Wirtschaft und Verkehr • Professur für Ökonomie und Statistik, insb. Im Verkehrswesen

Master thesis

On the topic

Autonomous Driving with Deep Reinforcement Learning

Student Name:	Yuhua Zhu
Major Field:	Elektrische Verkehrssysteme
Supervisor:	Prof. Dr. Ostap Okhrin
	Prof. Dr. Georg Hirte
Tutor:	M.Sc. Dianzhao Li

Dresden,

Zusammenfassung

Das Ziel dieser Masterarbeit ist es, die konkrete Implementierung von Deep Reinforcement Learning-Methoden im Bereich des autonomen Fahrens zu erforschen. Der Forscher implementiert eine autonome Fahrsimulation, indem er ein End-to-End-Policy-Modell auf der Grundlage von Deep Reinforcement Learning-Algorithmen in einer virtuellen Umgebung in Gym-duckietown trainiert. Die Steuerungsstrategie des Modells besteht darin, den Agenten für die Aufgabe der Spurerkennung zu steuern. Die Forscher haben mehrere Reinforcement-Learning-Algorithmen implementiert und schließlich den SAC-Algorithmus ausgewählt, um ein Nicht-End-to-End-Modell auf der Grundlage der vom Umfeld bereitgestellten Informationen wie Position und Geschwindigkeit als Eingabewerte und ein End-to-End-Modell mit Bildern zu trainieren, die von der Frontkamera des Agenten bereitgestellt werden. In der Arbeit vergleicht der Forscher die Vor- und Nachteile der beiden Modelle anhand der kinetischen Parameter in der Umgebung und führt eine Reihe von Experimenten zur Steuerungsstrategie des End-to-End-Modells durch, um die Auswirkungen unterschiedlicher Umgebungsparameter oder unterschiedlicher Reward-Funktionen auf die Modelle zu untersuchen. Die Vergleichsexperimente wurden durchgeführt, um die Variation der Dynamikparameter der verschiedenen Modelle zu untersuchen, aus denen die Leistung des End-to-End-Modells bewertet und die Wirksamkeit der trainierten Steuerungsstrategie verifiziert wurde. Die endgültigen Ergebnisse zeigen, dass Faktoren wie die Größe und Komplexität der Umgebung und das Gewicht der Reward-Funktion Auswirkungen auf die Leistung des End-to-End-Modells haben können.

Abstract

The goal of this master's thesis is to investigate the implementation of deep reinforcement learning methods in the context of autonomous driving. The researcher developed an autonomous driving simulation by training an end-to-end policy model using deep reinforcement learning algorithms in the Gym-duckietown virtual environment. The control strategy of the model was designed for the lane-following task. Several reinforcement learning algorithms were implemented and the sac algorithm was chosen to train a non-end-to-end model with the information provided by the environment such as position and speed as input values, as well as an end-to-end model with images captured by the agent's front camera as input. In this paper, the researcher compared the advantages and disadvantages of the two models using kinetic parameters in the environment and conducted a series of experiments on the control strategy of the end-to-end model to explore the effects of different environmental parameters or reward functions on the models. The comparison experiments were conducted to investigate the variation of the dynamics parameters of the different models, which evaluated the performance of the end-to-end model and verified the effectiveness of the trained control strategy. The final results revealed that factors such as the size and complexity of the environment and the reward function's weight can affect the end-to-end model's performance.

Contents

List of Figures

List of tables

List of listings

Symbols

Symbol	Einheit	Beschreibung
b		Bias term
ω		Weight
v	m/s	Speed
d	m	The distance between the agent and right lane’s centerline
Ψ	°	The angle between the agent and right lane’s centerline
γ		Discount factor
s		State
a		Action
r		Reward
R		Total reward function
V(s)		Value function
Q(s,a)		Action value function
$\pi(s)$		Policy function
H(P)		Entropy term
P		Probability distribution

List of abbreviations

DRL	Deep Reinforcement Learning
DQN	Deep Q-Network
A3C	Asynchronous Advantage Actor-Critic
PPO	Proximal Policy Optimization
DDPG	Deep Deterministic Policy Gradient
SAC	Soft Actor-Critic
WRC	World Rally Championship
Tanh	Hyperbolic Tangent Activation Function
ReLU	Rectified Linear Unit
MDP	Markov Decision Process
SARSA	State-Action-Reward-State-Action
OBS	Observation
SB3	Stable-Baseline3
Mlp	Multi-Layer Perceptron
CNN	Convolutional Neural Network Policy
RMSE	Root Mean Squared Error

Chapter 1 Introduction

1.1 Autonomous Driving Overview

Autonomous driving, commonly referred to as self-driving cars, is a rapidly evolving technology with the potential to revolutionize the automotive industry and profoundly impact individuals' lifestyles and mobility. The development of autonomous driving technology has been underway for several decades, with significant advancements made in recent years due to breakthroughs in machine learning, artificial intelligence, and sensor technology.

Autonomous driving technology can be categorized into different levels depending on the degree of automation, from Level 0 (no automation) to Level 5 (full automation). Level 0 and Level 1 mean that the vehicle has partially assisted driving functions. Level 2 and Level 3 vehicles have partially automated capabilities but cannot drive fully automatically. Level 4 and Level 5 mean that the vehicle can drive fully automatically and does not require human control.

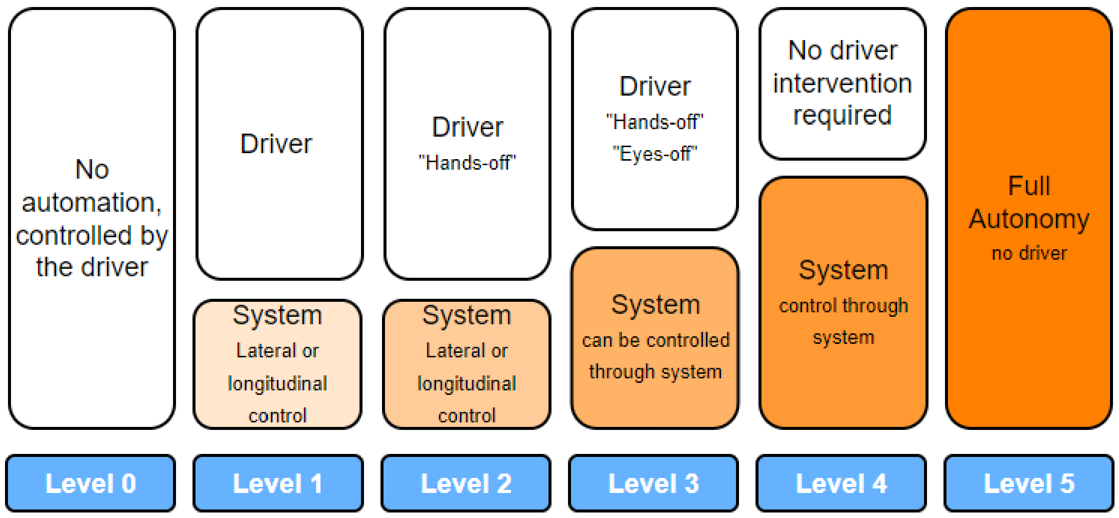

No automation, controlled by the driver
Driver
System
Lateral or longitudinal control
Driver
"Hands-off"
System
Lateral or longitudinal control
Driver
"Hands-off"
"Eyes-off"
System
can be controlled through system
No driver intervention required
System
control through system
Full Autonomy
no driver
Level 0
Level 1
Level 2
Level 3
Level 4
Level 5

Fig. 1. 1: Level of vehicle automation.

The development of autonomous driving technology is not solely the result of rapid advancements in artificial intelligence and other related technologies but is also motivated by the negative impact that the automotive industry has on the environment and road safety. Specific data indicate that increasing vehicles have adverse effects on the environment, traffic congestion, road safety, economic costs, and time efficiency.

Road traffic accidents are a global problem faced by countries around the world. In the United States, approximately 30,000 individuals die each year in car accidents [1], while in China, the number is approximately 260,000 [2]. According to the World Health Organization, about 1.24 million individuals die annually in road traffic accidents worldwide [3]. Many of these accidents are caused by fatigued or intoxicated drivers, and they can result in significant economic losses and seriously jeopardize the safety of those traveling on the road. To address this issue, not only must strict traffic rules be established and awareness of safe driving be raised, but technological advancements in vehicle technology must also be taken into account. The implementation of autonomous driving technology has the potential to greatly reduce traffic accidents by utilizing features such as collision warning, lane departure warning, and collision braking, which can effectively prevent traffic accidents. Traffic congestion and parking issues are among the challenges faced by large cities worldwide. In Germany, these problems contribute to harmful emissions and result in an annual economic loss of 25 billion euros [4]. To address these challenges, in-vehicle sensors can be integrated with intelligent traffic systems to optimize traffic flow at critical intersections. Furthermore, the parking function that comes with autonomous driving systems can assist drivers in parking more efficiently, thus saving time and space.

Air pollution is a critical environmental issue, and the automotive industry is one of the major contributors to air pollution. Research findings indicate that autonomous driving technology can lead to smoother acceleration and deceleration of vehicles, which improves fuel combustion efficiency and reduces harmful emissions such as nitrogen and carbon dioxide [1]. It is imperative to note that air pollution can have detrimental effects on human health and the environment, including respiratory diseases and climate change. Hence, adopting technologies such as autonomous driving can significantly reduce the automotive industry's negative impact on the environment.

Despite its many potential benefits, the deployment of autonomous driving technology is constrained by several factors. The biggest challenge is the ability to develop safe and reliable autonomous driving systems that can operate in a variety of complex conditions and environments. This requires not only advanced hardware but also sophisticated software, making it a major challenge. Therefore, more research must be devoted to autonomous driving to promote its potential benefits for the environment, economy, and life safety, among other factors.

1.2 Research Questions and Methods

1.2.1 Research Questions

The purpose of this research is to investigate the potential of Deep Reinforcement Learning (DRL) algorithms for autonomous driving. Given the rapid progress in machine learning and artificial intelligence, DRL is an emerging technique that holds great promise for achieving autonomous driving capabilities, as it can adapt to various driving scenarios. To accomplish this objective, the research will employ deep reinforcement learning algorithms in a simulated environment to control a cart and complete specific tasks. The study will compare and analyze several algorithms, selecting the one with the best training outcomes to present the final results. In summary, the study aims to develop a model based on reinforcement learning algorithms in a simulated environment that can act as an agent to control the cart and complete specific tasks. It will also provide insights into the research process and suggest directions for future improvements.

1.2.2 Research Methods

Autonomous driving can be achieved through the use of a conventional autonomous driving system, which consists of various subsystems such as perception, prediction, localization, and planning control. Each subsystem must meet high standards, and the system as a whole is very complex. Autonomous driving can only be achieved when these individual systems are perfectly integrated and working together.

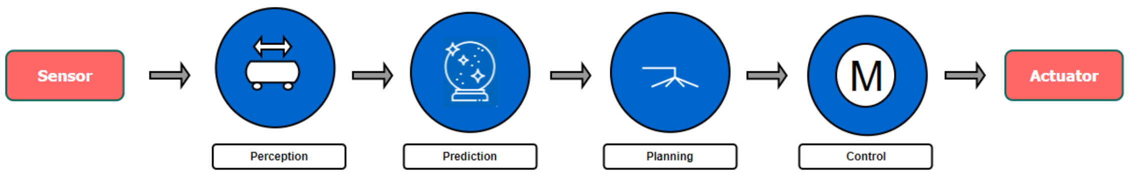


Fig. 1. 2: Conventional autonomous driving system.

With the advancement of Deep Reinforcement Learning, a novel approach to achieving end-to-end autonomous driving systems is gaining increasing attention. This approach relies on deep reinforcement learning algorithms that use images or unprocessed data from various sensors as input and agents that perform a specific action via a policy function. In the continuous driving model, the driver is replaced with a neural network that takes an image as input and outputs a control signal to adjust the vehicle's turning angle and speed. The input images are captured by a camera. The end-to-end system is simpler than the conventional self-driving system and does not require a series of operations such as perception, localization, prediction, etc. Through a trial-and-error mechanism, it can better explore the environment and achieve good performance. This mechanism continuously adjusts the driving strategy, which is beneficial for autonomous vehicles in complex road conditions. However, the input is an image, the learning efficiency is reduced, and the computer's computation time increases. The study is conducted in the context of the *Gym-Duckietown* simulation platform, which provides a virtual environment for agents to achieve autonomous driving [6].

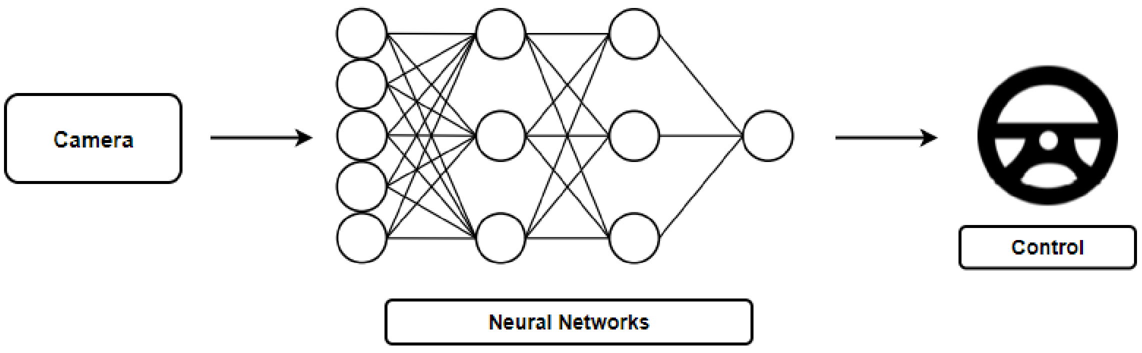


Fig. 1. 3: Structure of End-to-End System.

1.3 Paper Structure

In Chapter 2, the current research and development context of autonomous driving, end-to-end learning, and DRL will be presented. Additionally, the chapter will provide an introduction to the theoretical foundations of these topics, including machine learning, deep learning, and reinforcement learning.

Chapter 3 is the main focus of this thesis. It provides a detailed description of the research process and the steps involved in implementing autonomous driving in an end-to-end method. This includes the selection of appropriate algorithms, neural network structure, observation space, action space, and reward function. Moreover, this chapter examines the feasibility of these research steps to achieve autonomous driving simulation.

In Chapter 4, the training process based on the method described in Chapter 3 will be presented. The chapter will be divided into two parts based on the observations used: one-dimensional data such as position and velocity, and images. To train a non-end-to-end model based on reinforcement learning methods, one-dimensional data obtained after environmental processing will be used as input. On the other hand, images will be used as input to train an end-to-end model. The differences in the choice of various parameters between the two types of observations will be discussed and analyzed. The reasons for these differences will also be compared and analyzed.

In Chapter 5, the curve of the reward function change during training will be shown and the convergence rate of the two models will be shown. The performance of the end-to-end model will be also compared with that of the non-end-to-end model based on the training results and discuss the influence of the reward function on the policy.

In Chapter 6, a comprehensive evaluation of the trained models will be conducted through a series of tests. The tests will be carried out in different scenarios, and the number of iterations for each test will vary depending on the test scenario. All models will be tested five times, and the agent's velocity, distance from the midline, and angle concerning the midline will be recorded during testing. The collected data will be compared among different models to analyze the differences between the end-to-end model and two baselines and to investigate the effect of changes in reward function weights on the performance of the end-to-end model.

In Chapter 7, the conclusion will be drawn regarding whether an autonomous driving model based on an end-to-end approach can accomplish the lane-following task. Additionally, the stability of the model during testing will be analyzed, and the feasibility of implementing this approach from a simulated environment to the real world will be discussed, along with the future directions for its improvement.

Chapter 2 Research Background

2.1 Research Status

In recent years, DRL has made significant progress and has been applied to various fields, including autonomous driving research. Pomerleau proposed the first end-to-end driving model based on deep learning, which uses images as input to control the vehicle through a three-layer fully connected network [7]. Krizhevsky et al combined reinforcement learning with deep learning and developed the "Deep Q-Network" (DQN) algorithm, which can solve complex problems with unprocessed, high-dimensional inputs [8, 9]. The DQN algorithm can solve the problem of high-dimensional observation space, but it can only deal with discrete and low-dimensional action spaces. Xia, Li et al used the DQN algorithm to teach vehicles brake control on urban roads, reducing the probability of dangerous accidents under complex road conditions [10]. Jaritz et al implemented an end-to-end lane following policy using the "Asynchronous Advantage Actor-Critic" (A3C) algorithm to train an agent in WRC (World Rally Championship) platform [11, 12]. Vitelli et al created a DQN agent on the TORCS platform and successfully implemented a vehicle-to-vehicle policy in a simulated environment by comparing its performance with several standard agents [13]. Kalapos et al chose vision-based end-to-end reinforcement learning to solve the vehicle control problem and used a Proximal Policy Optimization (PPO) strategy to train the neural network in a "Duckietown" environment, achieving lane tracking and obstacle avoidance and implement the process from simulation to reality [14]. Huang et al investigated end-to-end decision-making for continuous action output based on the “Deep Deterministic Policy Gradient” (DDPG) algorithm in a TORCS environment and trained a superior decision control model by comparing it with the DQN model [15, 16]. In 2018, Haarnoja et al proposed the "Soft Actor-Critic" (SAC) algorithm, which softens the Q-value update and introduces an entropy term, allowing the algorithm to handle the balance between exploration and exploitation more flexibly [17].

Overall, DRL has made great breakthroughs and has the potential for further development. Deep reinforcement learning algorithms will be increasingly used in autonomous driving research.

2.2 Theoretical Basis

2.2.1 Machine Learning

Both deep learning and reinforcement learning are subfields of *machine learning* [18]. Machine learning involves the improvement and optimization of information processing systems through the use of appropriate data. These systems can learn from the data to perform not explicitly programmed actions. However, while the trained system can perform well on the training data, it may not be able to generalize to new, unseen data, resulting in poor generalization performance, which is known as overfitting.

Machine learning encompasses various methods, including supervised, unsupervised, and reinforcement learning. Supervised learning involves training a model on labeled data where each data point is associated with a corresponding label or target value. The objective is to predict the labeling of new input data by learning from the patterns observed in the training data. On the other hand, unsupervised learning entails training a model on unlabeled data without predefined labels or target values. The aim is to uncover underlying patterns or structures in the data using clustering and dimensionality reduction techniques. Reinforcement learning, in contrast, involves training an agent to make decisions based on feedback obtained from its environment. The agent learns by taking actions in the environment and receiving rewards or penalties in response. The goal is to optimize the cumulative reward over time by learning an optimal strategy.

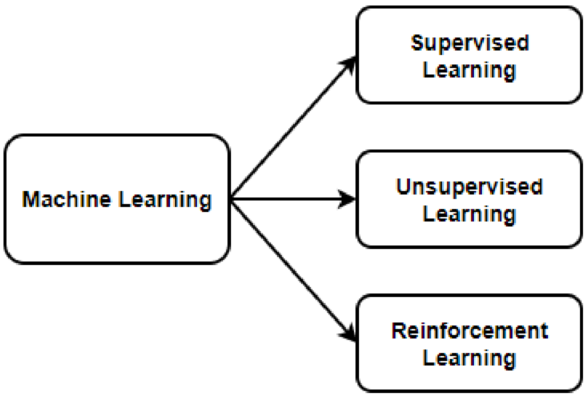


Fig. 2. 1: Components of Machine Learning.

2.2.2 Deep Learning

The artificial neural network is the earliest network model for deep learning. As early as 1943, American mathematician W. Pitts and psychologist W. McCulloch proposed artificial neural networks and established a theoretical model of each neuron in the artificial neural network through mathematical modeling [19]. Subsequently, psychologist D. Olding Hebb proposed a mathematical model of neurons and learning rules for neural networks [20]. The perceptron was then introduced as the earliest and structurally simplest model of artificial neural networks. In the 1980s, G. Hinton et al. replaced the single-layer structure of the perceptron with a deep structure with multiple hidden layers, which became the earliest deep-learning network model [21].

Deep learning is a subfield of machine learning that enables computers to process data in a similar way to the human brain. Essentially, deep learning is a multi-layer neural network that can recognize images, text, sounds, and other types of data to make accurate predictions and classifications. As mentioned previously, single-layer neural networks can also make approximate predictions, but additional hidden layers can optimize the network and improve recognition accuracy. In essence, a neural network is used to establish a mapping relationship between input and output. Neural networks can reduce the dimensionality of data and extract features more effectively. Typically, neural networks are composed of many neurons, and these neurons form each layer of the neural network. When a neural network is represented as a diagram, as shown in Fig 2.2 (a), the leftmost column usually represents the input layer, the middle column is referred to as the hidden layer, and the rightmost column is the output layer.

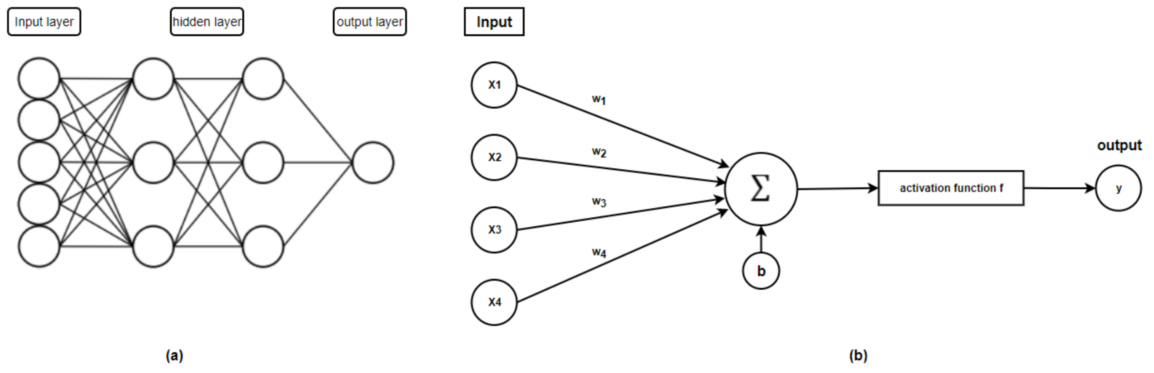


Fig. 2. 2: (a): Artificial neural network architecture, (b): The working principle of a neural network.

Fig 2.2 (b) depicts a single artificial neuron, which effectively illustrates the working principle of a neural network. Assuming that the inputs to the neural network are represented by x, each input is assigned a corresponding weight value ω. The bias term b is also included in the computation, and the resulting values are summed together to obtain the total output value through an activation function.

Equation (1) is the equation corresponding to the Fig. 2.2 (b) [22].

$$y = f(\sum_i \omega_i x_i + b) \tag{1}$$

Deep Learning utilizes neural networks as function approximators to model complex functional relationships. However, to create complex functions, a nonlinear activation function is necessary. The activation function layer, also known as the nonlinear mapping layer, increases the nonlinearity of the function, allowing the network to learn complex data and providing Deep Learning with powerful representation capabilities to obtain complex functions. Without an activation function, the input and output of a neural network, regardless of how complex its architecture, would be linear, making it impossible to solve complex problems.

2.2.3 Reinforcement Learning

Reinforcement learning (RL) involves several important concepts, including agent, environment, state, and reward. The agent is an intelligent individual capable of learning and reasoning. In the context of this paper, the agent refers to a self-driving car. The environment provides the agent with a state, and each time the agent performs an action, the environment returns a new state. Reinforcement learning involves the process of an agent learning how to select the best action in a particular state by gaining experience and maximizing the reward value. The higher the reward value, the better the action.

Reinforcement learning can be thought of as a trial-and-error learning process, where the agent makes mistakes and then acquires the correct strategy from experience [25]. The learning is delayed because the agent does not receive external feedback or correction after each action like in supervised learning. Instead, the agent must achieve its objective according to the reward value. The environment presents the agent with various possible states and determines the corresponding reward or punishment for taking an action in the current state.

However, reinforcement learning has some limitations. For example, it can require a large amount of computation and can be computationally intensive, making it difficult to implement on mobile devices with limited hardware resources. Additionally, the design of the reinforcement learning algorithms and the neural networks can be complex, especially for complex tasks, which increases the time and labor cost of developing the models.

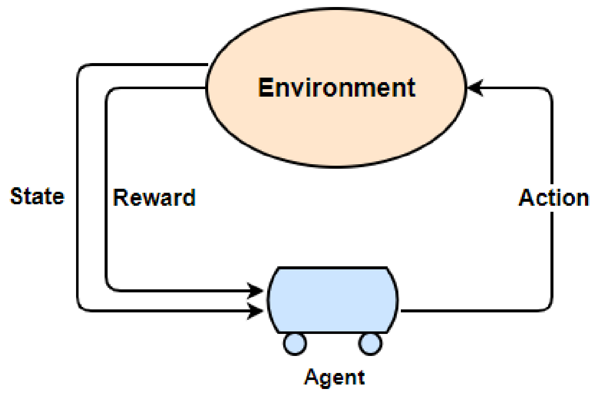


Fig. 2. 3: Reinforcement Learning.

Reinforcement learning is commonly modeled as a Markov decision process (MDP) [25]. A process that adheres to an MDP has the property that, in a given state, the conditional probability distribution of the next state is independent of the previously experienced state and depends only on the current state. This implies that, regardless of the number of state sequences experienced in reaching the current state, the conditional probability distribution of the next state remains the same and is solely determined by the current state.

In the MDP framework, the reinforcement learning agent aims to maximize the total discounted reward by finding an optimal policy. The current state, action a, reward r, and the next state s_{t+1} constitute a tuple and are represented by the set (s_t, a, r, s_{t+1}). The policy function π maps states to actions and the expectation of reward can be maximized by optimizing the policy function. Thus, the agent continuously updates the policy function through training to maximize the expected reward.

The value function $V(s)$ is used to describe the expectation of the sum of the discounted rewards obtained by executing a certain strategy in state s:

$$V^{\pi}(s) = E[\sum_{i=0}^{n} \gamma^{i} r_{i+1} | s = s_0] \tag{2}$$

In the reinforcement learning process, the discount rate γ is a key parameter that determines the weight of future rewards. The value of γ ranges from 0 to 1, and its value depends on the significance of future rewards. If future rewards are not significant, the value of gamma will be small. Generally, it is necessary to gradually decrease the value of future rewards to reduce their influence on the agent's current action choice.

The action value function Q is defined as the expected cumulative reward obtained by taking an action in a given state:

$$Q^{\pi}(s,a) = E[\textstyle\sum_{i=0}^{n} \gamma^{i} r_{i+1} | s = s_0, a = a_0] \tag{3}$$

When the strategy is deterministic, the Q function can be expressed by the Bellman equation:

$$Q^{\pi}(s_t, a_t) = E[r + \gamma Q^{\pi}(s_{t+1}, a_{t+1}) | a_{t+1} = \pi(s_{t+1})] \tag{4}$$

where the optimal policy is the one with the highest Q value for a given state:

$$\pi^{*}(s_t) = argmax_{a} Q^{\pi}(s_t, a_t) \tag{5}$$

At the outset, it may not always be possible for the agent to obtain the action corresponding to the highest Q value. Therefore, the agent needs to continuously attempt to search for the maximum Q value and the most effective strategy.

There are two primary types of reinforcement learning methods: value-based RL and policy-based RL. Value-based reinforcement learning algorithms involve estimating the expected reward that can be accumulated by following a certain policy in each state. The most common algorithms in this category include Q-learning and SARSA [26]. For example, in Q-learning, the agent estimates the value of each action A in a state using the Q-table. The agent selects the action with the highest expected value and updates the Q-table according to the reward received. This iterative process eventually converges the Q-table to the optimal policy. Value-based reinforcement learning algorithms are computationally efficient and can handle large state spaces. However, in cases where the action space is high-dimensional, it may not be possible to discretize the space, making it challenging to optimize the Q-function. In such cases, a policy-based reinforcement learning algorithm may be necessary. Policy-based reinforcement learning algorithms focus on finding the optimal policy that maximizes the expected reward. These algorithms typically use neural networks to approximate the policy and update it based on the reward received. When the action space is high-dimensional, policy-based algorithms tend to be more effective than value-based algorithms. However, when learning complex policies, policy-based algorithms can be computationally expensive, time-consuming, and unstable.

2.2.4 Deep Reinforcement Learning

DRL is an integration of deep learning and reinforcement learning. In DRL, deep neural networks are capable of extracting features from high-dimensional, unprocessed state spaces, generating policy models through reinforcement learning algorithms, and producing actions, thereby enabling the processing of high-dimensional input spaces. However, at the outset, the policy is random and the agent must engage in continuous trial and error to determine the optimal policy.

DRL has been extensively employed in various fields with favorable outcomes. AlphaGo is a notable example of DRL's success, which combined tree search and deep neural networks to defeat the Go world champion [27]. DRL algorithms, like RL algorithms, are classified into value-based and policy-based approaches. Value-based approaches, such as DQN, use deep neural networks to estimate the value that arises from executing an action in a given state. This forms the basis for algorithms like "double DQN" (DDQN), which employs two networks to evaluate the value and select actions [28]. Policy-based approaches aim to optimize and determine the optimal policy, such as REINFORCE [29], which uses policy gradients to update the policy based on rewards.

However, deep reinforcement learning has several disadvantages. Since the input consists of high-dimensional state spaces or unprocessed images, it takes longer for the computer to process the input, which prolongs the learning process and requires additional time to train the neural network. Furthermore, the training results are more difficult to converge. Despite the widespread use of deep reinforcement learning algorithms, there is a need to improve their learning efficiency and stability.

Chapter 3 Method

In this chapter, the research methodology is presented, which comprises various steps based on the MDP. The programming language utilized for this study is *Python*. Each step in the methodology is implemented through *Python* code for the specific implementation. The specific implementation framework is depicted in the following figure:

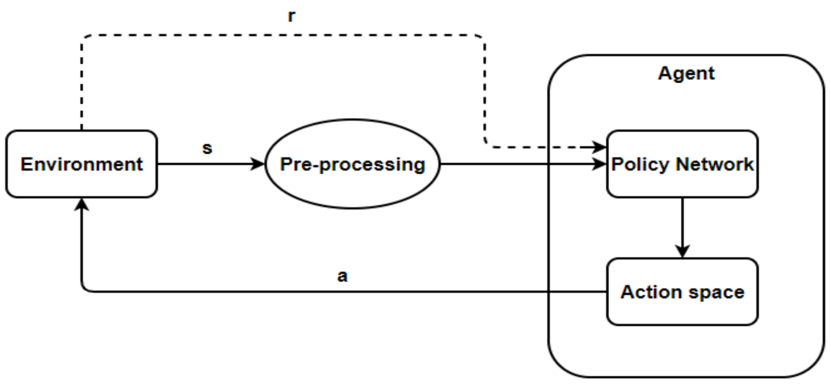


Fig. 3. 1: Approximate framework.

The first step involves creating a virtual environment to receive input observations from the actual environment. The input is preprocessed and then fed into the neural network, with the car acting as the reinforcement learning agent. The neural network extracts feature from the input and map them to action distributions using a linear layer. An action is then selected and executed, and the environment returns a reward value to the agent.

To determine the research platform, the observation space, and input values must first be considered based on the structure of the end-to-end system. The onboard camera in the virtual environment provides RGB images to the car in a size of 640 x 480 [6]. Additionally, the environment provides information such as position and speed, which can be interpreted as obtained from various onboard sensors in the real world. After determining the input, preprocessing of the input is required, which is then imported into the neural network for feature extraction and output of action values. The selection of the neural network depends on the input space selection. The action space is divided into discrete action space and continuous action space, and the selection of action space depends on the reinforced learning algorithm used. The reward function is determined according to the task.

3.1 Simulation Platform

The Gym-duckietown is a free and open-source autopilot simulator that is available on the GitHub website. It is based on *Python* programming language and utilizes the OpenAI Gym toolkit to construct a virtual environment that closely resembles the real World. It offers a virtual environment that emulates a real-world scenario, enabling researchers to train reinforcement learning agents. The platform includes various components, including roads, intersections, obstacles, and pedestrians, among others, to replicate a real-world driving experience, as shown in Fig. 3.2.

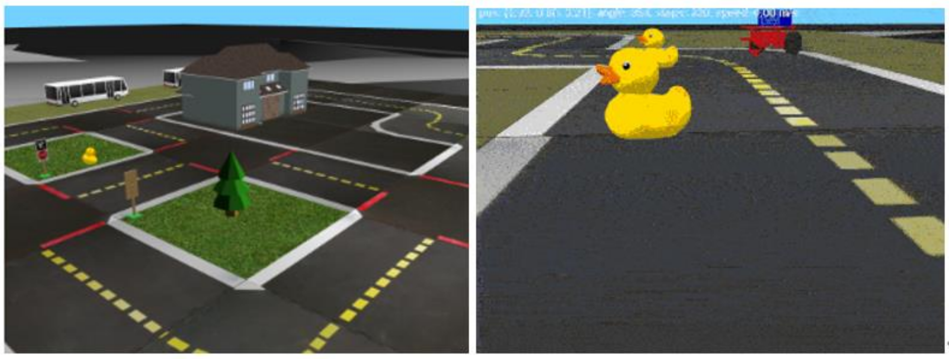


Fig. 3. 2: Gym-duckietown, includes different scenarios with necessary components such as Duckiebot, different roads, intersections, obstacles, pedestrians, etc [30].

The agent in the environment is equipped with a front camera that captures image information of the environment. These raw images can be used to train an end-to-end model. Additionally, the environment offers a comprehensive set of position, velocity, and other relevant data that is crucial for evaluating the current dynamics of the cart. This information is valuable not only in the virtual environment but also in the real world, where it can be collected and analyzed by various sensors.

3.2 Control Task

The environment in Gym-duckietown presents a closed-loop scenario with two lanes, wherein the reinforcement learning agent must be trained to follow the right lane. Specifically, the agent travels at variable speeds on the right lane and quickly adjusts its position to return to the lane if it deviates from it. The optimal behavior for the agent is to drive along the center line of the right lane at the maximum speed, as it yields the highest reward value.

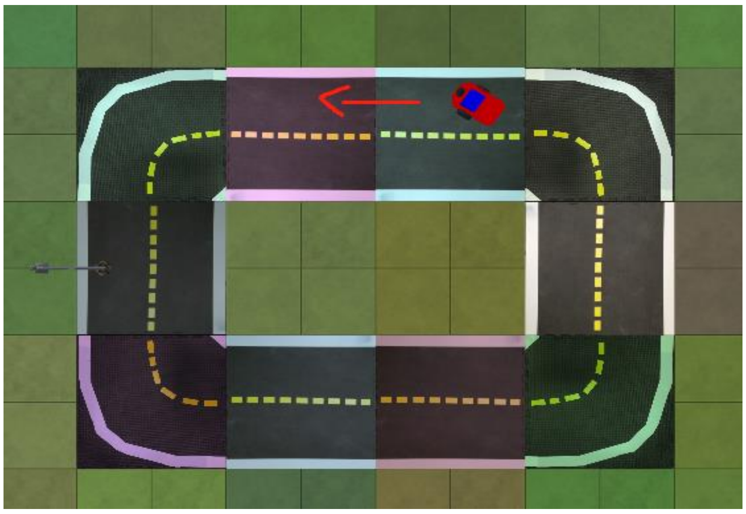


Fig. 3. 3: Control Task: Right lane-following.

3.3 Observation Space

In OpenAI Gym, the observation space defines the set of states that the environment may be in, and it specifies the type and shape of observations that the agent receives from the environment at each time step. This information is used by the agent to make decisions about what action to take in the environment. In the Gym-duckietown environment, the default observation space is defined as a 640 x 480 observation image that is provided by the onboard camera, as described earlier.

The observations in OpenAI Gym can take various forms depending on the type of environment being used. For example, in the Gym-duckietown environment, low-dimensional position, velocity, and other relevant data provided by the environment can also be selected as observations, in addition to the high-dimensional images that can be defined as observations.

```
# We observe an RGB image with pixels in [0, 255]
        # Note: the pixels are in uint8 format because this is more compact
        # than float32 if sent over the network or stored in a dataset
        self.observation_space = spaces.Box(
            low=0, high=255, shape=(self.camera_height, self.camera_width, 3), dtype=np.uint8
        )
```

Listing. 1: Image as observation.

The above code is the default observation space in this virtual environment. The observation is a box space because RGB images are used as observations at this time.

As explained in the preceding section, the observation space can be either continuous, using the box space to define a high-dimensional image as input, or a low-dimensional cart dynamic variables as input, including information such as vehicle speed and steering angle. Therefore, this study will define two different types of observations, images, or information as input:

Tab. 1: Observation spaces.

Observations	Type	Dimension	Type of model
Image	Box	High	End-to-end
Information	Box	Low	Non-End-to-end

The two observations serve as distinct inputs for training the model. The main objective is to develop the end-to-end model that can achieve the control task, which involves using the image as input, and this is the primary focus of this paper. The non-end-to-end model (which uses information as input) is only employed in subsequent tests to evaluate the reward function and for simple comparisons with the end-to-end model.

3.3.1 Information as Observation (Non-end-to-end)

As previously mentioned, the agent's main task is to follow the right lane, and therefore the observations need to be selected based on this task for the data provided by the environment. Through several experiments, it was determined that the agent's velocity, distance, and angle relative to the right lane centerline was the most useful data for this task.

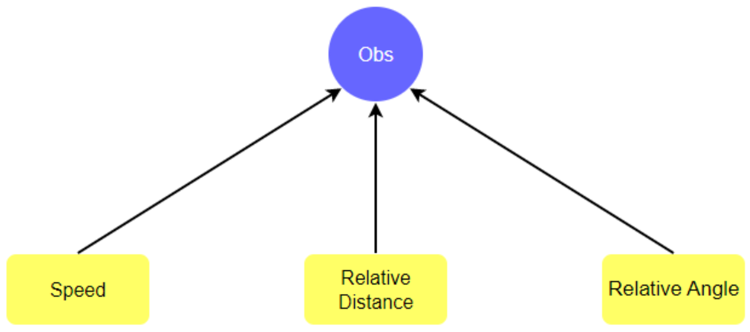
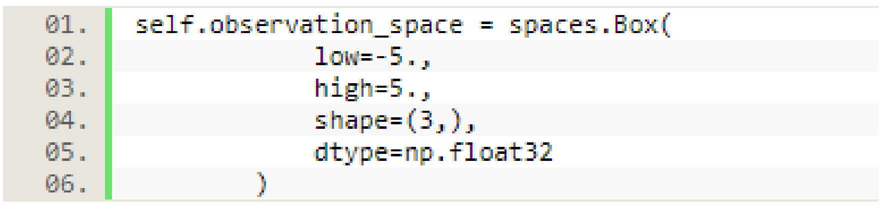


Fig. 3. 4: 1D-observation space.



```
self.observation_space = spaces.Box(
                low=-5.,
                high=5.,
                shape=(3,),
                dtype=np.float32
           )
```

Listing. 2: Information as observation.

This code defines the observations used in the environment by creating an observation space of shape 3, which represents a one-dimensional array with three elements. These three elements are presented in Fig.3.4 as floating-point numbers ranging from -5 to 5. The choice of this value range (-5, 5) is dependent on the requirements of the task and the expected behavior of the agent within the specified range. This range of values is chosen to normalize the input values and enhance the stability and convergence rate during the training process.

3.3.2 Images as Observation (End-to-end)

The duckiebot as the agent in Ducktown is equipped with a front camera that captures RGB images of size 640 x 480. This study uses the unprocessed image as the input for achieving end-to-end learning.

```
self.observation_space = spaces.Box(
                low=0.,
                high=1.,
                shape=(3, self.camera_height, self.camera_width),
                dtype=np.float32
        )
```

Listing. 3: Images as observation after normalization.

Listing.1 defines the default observation values for the environment, which are the three RGB channels of an image in the range [0, 255]. To normalize this range, as shown in Listing.3, [0, 255]/255 is converted to [0, 1]. This normalization process ensures the stability of the neural network and accelerates the convergence rate during training.

However, to improve the learning efficiency and reduce the learning difficulty, a scaling process is applied to the image before feeding it into the neural network. The image is reduced from 640 x 480 to 32 x 32 through preprocessing. After multiple training sessions, it has been demonstrated that the preprocessed image can be learned at a faster rate. The images provided by the camera are shown in Fig 3.5.

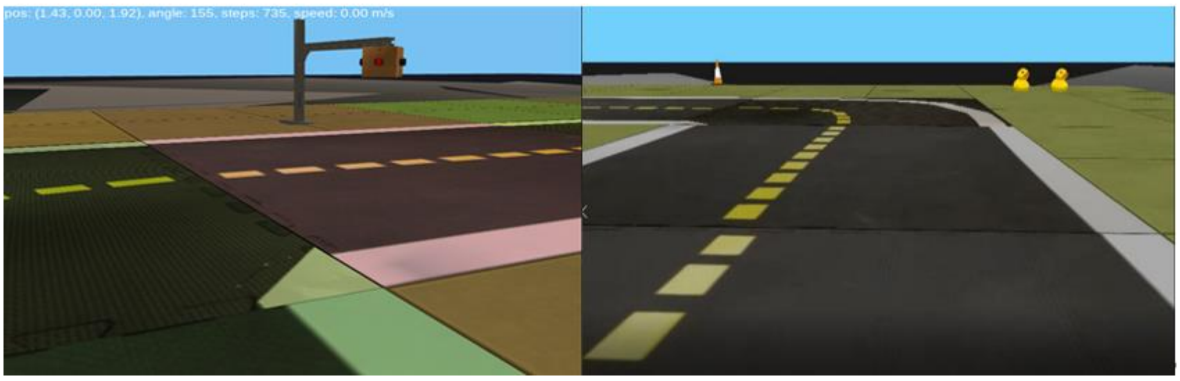

pos: (1.43, 0.00, 1.92), angle: 155, steps: 735, speed: 0.00 m/s

Fig. 3. 5: Image observation space.

Scaling down the image size can improve training efficiency, particularly when using larger neural networks or when limited by hardware resources. However, it may also result in a loss of information in certain regions of the image, leading to inadequate training outcomes. Consequently, the selection of an appropriate input image size should be based on the specific task requirements and hardware constraints.

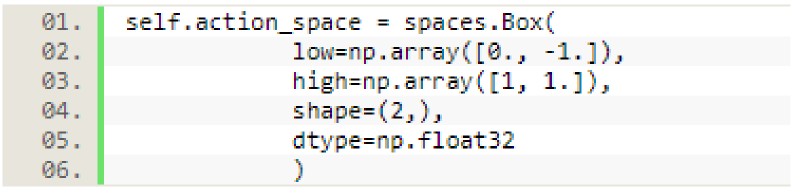
In summary, when using an image as the observation, the input image is an RGB image of size 32 × 32 × 3.

3.4 Action Space

In the context of reinforcement learning, the action space refers to the set of all possible actions that an agent can take in a given environment. The choice of action is typically determined by a policy that maps the current state of the environment to a distribution over the set of possible actions. The policy may be deterministic, meaning that it selects a single action with the highest probability, or stochastic, meaning that it selects actions randomly according to the probability distribution.

In the Gym-duckietown environment, the agent under control is a differential-wheeled robot, and the range of rotation for its two wheels can be interpreted as being within the interval [-1, 1], meaning it can move in either direction at maximum speed. To simplify the training process and minimize unnecessary actions, the rotation range for the two wheels can be restricted to [0, 1], limiting the agent's motion to forward movement or coming to a stop, as shown in the listing. 4.

Furthermore, the rotation of the steering wheel adds another dimension to the action space, allowing the agent to turn. As such, this environment employs a continuous action space consisting of two dimensions, with the first representing the agent's speed in the interval [0, 1], indicating that the agent's speed can only vary between 0 and maximum speed. The second dimension represents the steering angle, defined in the interval [-1, 1], indicating the maximum steering angle of the agent, either to the left or to the right, is 45°.



```
self.action_space = spaces.Box(
                low=np.array([0., -1.]),
                high=np.array([1, 1.]),
                shape=(2,),
                dtype=np.float32
                )
```

Listing. 4: Action space.

In summary: two dimensions of a continuous action space. The first dimension is velocity, which takes values in the range [0, 1], and the specific velocity values range from 0 to the maximum velocity. The movements that the agent performs backward are truncated to reduce the difficulty of the training. The second dimension is the lateral rotation angle of the Agent. The range of this value is [-1, 1], meaning that the agent can turn to the left or to the right, which corresponds to the actual angle range of [-45°, 45°] in the environment.

3.5 Algorithm

The deep reinforcement learning algorithm used in this application is the Soft Actor-Critic (SAC) algorithm, which is a model-free, off-policy reinforcement learning algorithm employed to solve problems with continuous output spaces [17]. The SAC algorithm maintains a critical network that estimates the state-action value function (also known as the Q-function) and an actor-network that maps states to actions. The actor network is updated using the gradients of the estimated Q-values concerning the actions, similar to the DDPG algorithm. However, SAC also introduces an entropy term into the actor loss function, which encourages the policy to be more random and exploratory.

SAC combines the concepts of maximum entropy reinforcement learning and value-based methods to learn a stochastic policy that maximizes the expected long-term reward while also maximizing entropy or the degree of randomness in the policy. This encourages exploration and helps to avoid getting stuck in local optima. The entropy in SAC can be interpreted as the degree of confusion or randomness, and higher entropy indicates that it contains a larger amount of information [31].

3.5.1 Mathematical Foundations

In general, the objective of deep reinforcement learning algorithms is to maximize the expected value of accumulated rewards, as illustrated in Equation (5). However, in the case of maximum entropy reinforcement learning algorithms, the goal is not only to maximize the expected value of rewards but also to maximize the entropy of each output action. This means that the algorithm aims to encourage exploration by ensuring that the policy produces a diverse set of actions with a high degree of randomness or uncertainty:

$$\pi^*_{MaxEnt} = argmax_\pi \sum_i E_{(s_t,a_t)\sim\rho_\pi}[r(s_t,a_t) + \alpha H(\pi(\cdot\,|s_t))] \tag{6}$$

By maximizing entropy, the algorithm ensures that the policy does not become too deterministic and explores a wide range of actions. The temperature parameter α controls the trade-off between entropy and reward. In information theory, entropy is a measure of the amount of uncertainty or randomness in a probability distribution. If we have a random variable x with distribution P, then the entropy H(P) of x is given by:

$$H(P) = E_{x\sim P}[-logP(x)] \tag{7}$$

Therefore, based on the objective function (6), entropy is introduced to obtain the soft value function:

$$Q^\pi_{soft}(s,a) = E[r(s,a) + \gamma\left(Q^\pi_{soft}(s',a') + \alpha H(\pi(\cdot\,|s'))\right)] \tag{8}$$

And soft V-function:

$$V^\pi_{soft}(s') = E[Q^\pi_{soft}(s',a') + \alpha H(\pi(\cdot\,|s'))] \tag{9}$$

Equation (8) and Equation (9) show that the relationship between the V-function and the Q-function can be expressed by the following equation:

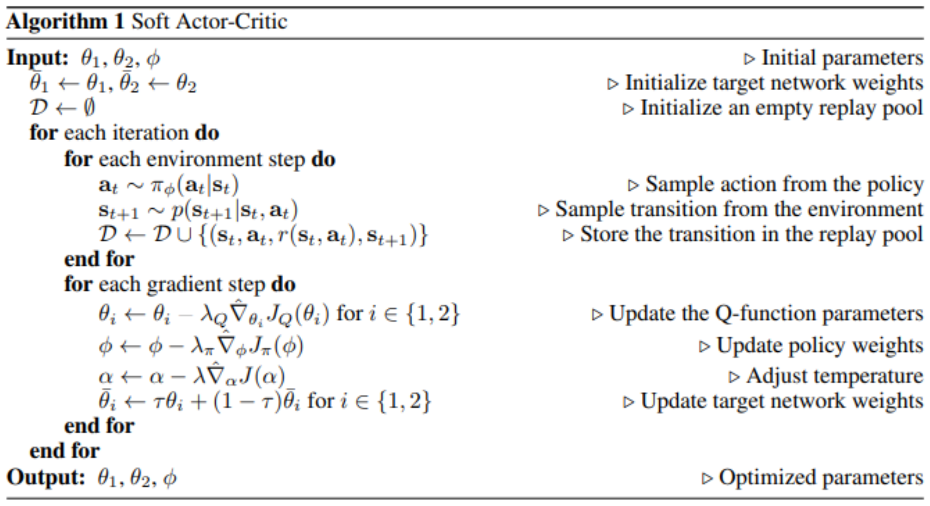
$$Q^\pi_{soft}(s,a) = E[r(s,a) + \gamma V^\pi_{soft}(s')] \tag{10}$$

The policy function, also known as the actor, is a crucial component of the SAC algorithm that selects actions to maximize the expected reward. Typically, the policy function is implemented using a neural network that maps states to actions. However, in SAC, the policy function is stochastic, meaning that it generates a probability distribution over the action space, instead of a deterministic action. This approach enables the algorithm to explore different actions and enhances its robustness to noise in the environment.

3.5.2 Policy Iteration

During policy evaluation in the SAC algorithm, the current policy interacts with the environment to generate a batch of transitions (state, action, reward, next state). The Q-values are then estimated based on this batch of transitions, which represents the expected cumulative reward for following the current policy from a given state. This is achieved through two neural networks: a critic network and a target critic network. The critic network takes a state-action pair as input and outputs a scalar value, which is an estimate of the Q-value. The target critic network is a copy of the critic network and is updated slowly over time using a soft update rule. The soft update rule ensures that the target critic network is a moving average of the critic network and helps to stabilize the learning process.

Once the Q-values are estimated, the policy update step begins. The goal of policy improvement is to maximize the expected cumulative reward and improve the policy accordingly. This is done by taking the gradient of the expected cumulative reward concerning the policy parameters and updating the policy parameters in the direction of the gradient.



```
**Algorithm 1** Soft Actor-Critic
**Input:** $\theta_1, \theta_2, \phi$                                          ▷ Initial parameters
  $\bar{\theta}_1 \leftarrow \theta_1, \bar{\theta}_2 \leftarrow \theta_2$      ▷ Initialize target network weights
  $\mathcal{D} \leftarrow \emptyset$                                           ▷ Initialize an empty replay pool
  **for** each iteration **do**
      **for** each environment step **do**
          $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$                 ▷ Sample action from the policy
          $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$   ▷ Sample transition from the environment
          $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$   ▷ Store the transition in the replay pool
      **end for**
      **for** each gradient step **do**
          $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$   ▷ Update the Q-function parameters
          $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$             ▷ Update policy weights
          $\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha)$             ▷ Adjust temperature
          $\bar{\theta}_i \leftarrow \tau\theta_i + (1 - \tau)\bar{\theta}_i$ for $i \in \{1, 2\}$   ▷ Update target network weights
      **end for**
  **end for**
**Output:** $\theta_1, \theta_2, \phi$                                         ▷ Optimized parameters
```

Fig. 3. 6: SAC algorithm flowchart [31].

3.6 Policy Architecture

The choice of neural network architecture for a given reinforcement learning problem is influenced by the nature of the observations and the algorithm being used. In this study, the policy model provided in *Stable-Baselines3* (SB3) is employed as the controller [32]. SB3 policies incorporate multiple networks, including actor, critic, target actor, and target critic, to create an optimizer. Each network consists of a feature extractor and a fully connected network [32]. The feature extractor primarily comprises a deep neural network, followed by a fully connected layer that converts the input feature values into output values.

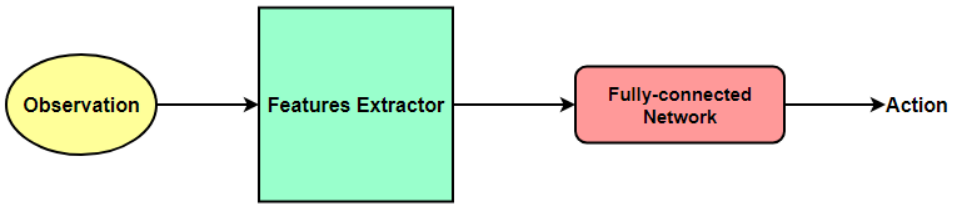


Fig. 3. 7: Default Network Architecture of SB3.

The policy network structure in SB3 can be seen in the above figure. The choice of observation space and algorithm determines the specific network structures used.

3.6.1 Network Architecture for Non-end-to-end Model

As mentioned previously, the environment provides information parameters such as the agent's specific speed, distance, and angle between the agent and the right lane's centerline, which are initially input as observations. These observations form a one-dimensional input to the neural network. However, due to the low dimensionality of the observation space, there is no need to use a deep neural network as a feature extractor. Therefore, a simple two-layer fully connected network with 32 units is sufficient [32].

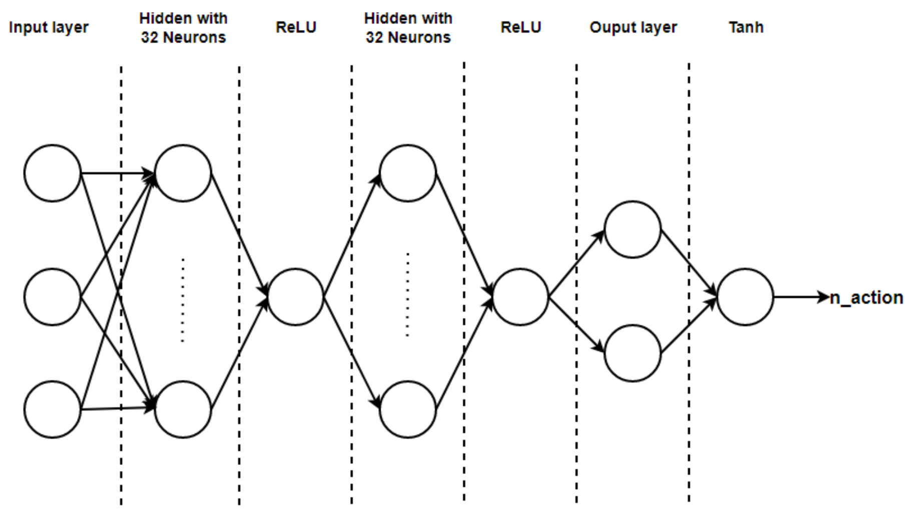


Fig. 3. 8: Actor-Network Architecture for 1D observation space.

In the SB3 library, the policy network is called 'MlpPolicy' and is a class that defines the actor-network structure, as shown in Fig. 3.8 [32]. The input is a one-dimensional set of 3 elements, which corresponds to the information parameters provided by the environment, including the specific speed of the agent, the distance, and the angle between the agent and the right lane's centerline. The output 'n_actions' represents the dimension of the action space, which is 2 in this case. The number of neurons in the first and second hidden layers is 32. After the input of observations, there is a flattened layer that transforms the state variables into a one-dimensional tensor, which is equivalent to the feature extraction layer. The activation function of each hidden layer is the Rectified Linear Unit (ReLU) function. The output layer is the last, and it uses the hyperbolic tangent (Tanh) function to scale the output range between -1 and 1, which is suitable for the continuous action space in this problem.

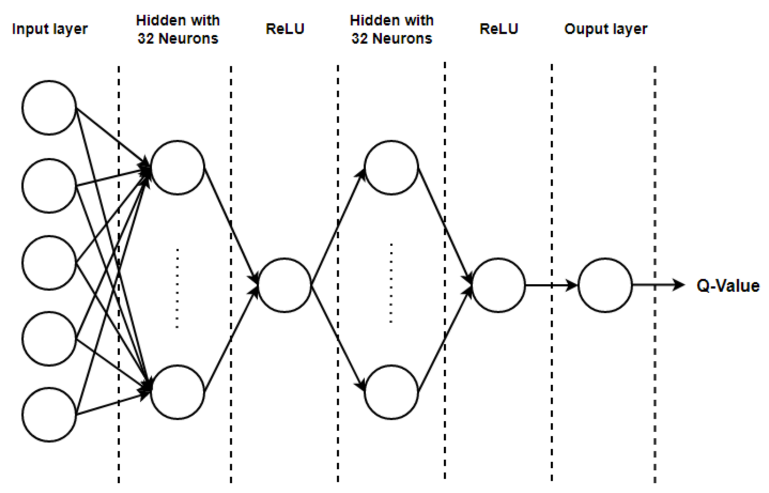


Fig. 3. 9: Critic-Network Architecture for 1D observation space.

The critic network in the 'MlpPolicy' of Stable Baseline3 consists of three fully connected layers with 32 neurons, 32 neurons, and 1 neuron respectively, as shown in Fig. 3.9. The input layer includes the ConcatLayer, which concatenates the observation and action variables to form a 5-dimensional vector (3 observations and 2-dimensional action space). The final output of the critic network is the estimated Q-value.

3.6.2 Network Architecture for End-to-end Model

When the input is an image, a deep neural network is required to serve as the feature extractor. A two-layer convolutional neural network (CNN) will be selected as the feature extractor, which can extract high-level features from images. The extracted feature values are then passed to a linear layer, which is connected to the corresponding actions or values.

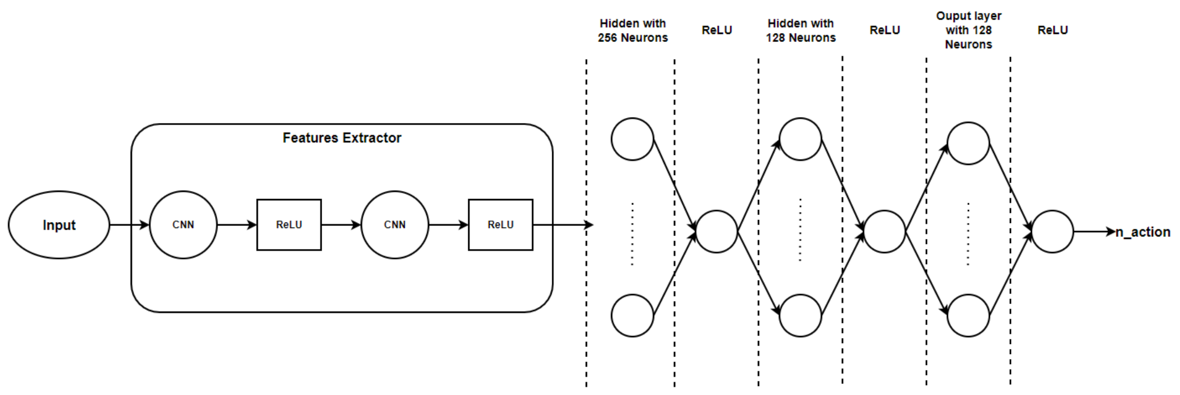


Fig. 3. 10: Network Architecture for image observation space, includes two CNN layers for feature extraction and three linear layers to map feature vectors to outputs.

In Stable-Baselines3, policy networks refer to all networks that are useful for training, not just those used to predict actions. The SAC algorithm is the off-policy algorithm, so separate feature extractors are needed for approximating the actor function and the critic function, respectively.

3.7 Reward Shaping

In reinforcement learning, the reward function is a critical component of the learning process that guides an agent to achieve a particular goal. The reward function maps a state-action pair to a scalar reward signal that provides feedback to the agent about the quality of its actions in a given state, as shown in Fig. 2.5. The agent's goal is to maximize the cumulative sum of these rewards over time, also known as the rate of return. Determining an appropriate reward function is critical to the success of the learning process. A well-designed reward function should encourage the agent to take actions that lead to the achievement of the desired goal while avoiding undesirable states or actions. The reward function should be designed considering the task at hand and the desired behavior of the agent.

A common approach to defining the reward function is to use a shaping reward that guides the agent toward a desired behavior. This can be done by combining multiple reward signals, each corresponding to a particular aspect of the behavior that the agent is expected to exhibit. Another approach is to use sparse rewards that provide feedback to the agent only when a specific goal has been achieved. This can be useful in cases where it is difficult to design a shaping reward.

This simulation aims to train an end-to-end model that can effectively control the agent to drive steadily along the right lane. The reward function used in the simulation is composed of two main parts. The first part is the speed reward, which incentivizes the agent to move with a constant velocity. This component of the reward function encourages the agent to maintain a steady speed, which results in a higher reward value.

The second part is the orientation reward, which is designed to encourage the agent to stay close to the centerline of the right lane. This component of the reward function is based on the agent's distance and angle to the right lane centerline. The reward value also includes a mileage component, which increases as the agent travels further along the right lane. If the agent moves out of the right lane, the reward value is 0. This part of the reward function encourages the agent to stay within the right lane and move steadily along the center line of the right lane to achieve a higher reward value.

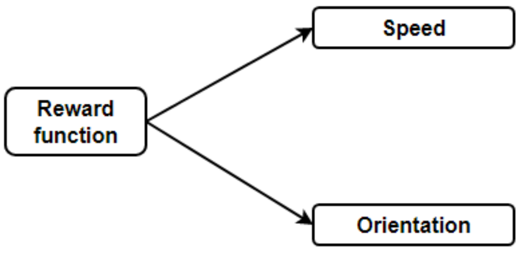


Fig. 3. 11: Components of the reward function.

3.7.1 Calculation of Speed-based Reward Function

In the autonomous driving simulation, the initial task involves assigning a speed to the car. To accomplish this, a maximum speed must be established within the environment, which also serves as the target speed for the agent. The reward function assigns a higher value to the agent when its current speed closely approximates the target speed. Thus, the greater the proximity of the agent's current speed to the target speed, the higher the reward value.

$$r_v = \frac{current\ speed}{target\ speed} \tag{11}$$

The aforementioned formula presents the calculation method for the bonus value corresponding to the speed component. As previously stated, the target speed is the upper limit speed specified by the environment, thus remaining constant throughout the task. Meanwhile, the current speed of the cart varies with each time step and is capped at the target speed. Consequently, the reward value of the speed part increases only as the current speed gets closer to the target speed.

3.7.2 Calculation of the reward function based on the position of the agent relative to the right lane

To successfully achieve the lane-following task, it is essential to have accurate information on the agent's distance and angle relative to the centerline of the right lane. The environment defines the position of the right lane, and this parameter is utilized in the formulation of this aspect of the reward function:

$$r_\psi = e^{-(5 \cdot right\_dist)} + Travel\ Distance \tag{12}$$

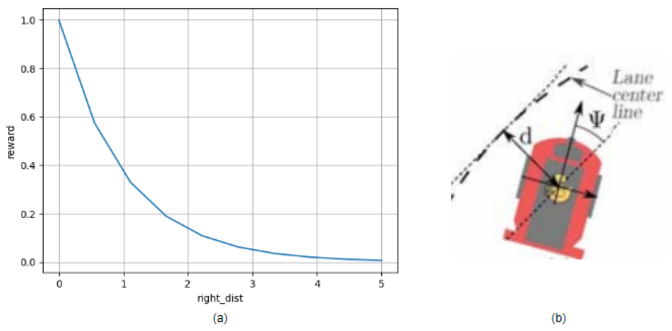


Fig. 3. 12: (a) shows that the reward value decreases when the relative distance and the relative angle become larger. (b) shows the distance and the angle between the agent and right lane's centerline [14].

The function 'right_dist' in the equation above is a variable that is set based on the right lane position in the environment, as illustrated in Fig. 3.12. It describes the distance and angle between the agent and the centerline of the right lane. When translated into code, the absolute value of the variable is utilized to ensure that the reward function is always positive and that it decreases as the absolute value of the variable increases. This function is directly used to evaluate the reward value of the localization component. As the agent approaches the centerline of the lane, the angle between the agent and the centerline decreases, increasing the reward value. A factor of 5 is added in front of this variable to amplify the effect of the relative distance and clamping angle on the reward value.

Another variable, 'Travel Distance', defines the distance traveled by the agent in the right lane. With each frame update, the agent changes its position in the environment, and the distance between these two positions is calculated as the bonus value for that turn. However, a condition needs to be set for this reward value to be valid, i.e., the agent must travel in the right lane; otherwise, the distance and the reward value are both zero.

The reward value for this component of the function is designed to encourage the agent to adopt a strategy of driving steadily along the centerline of the right lane for as long as possible.

The total reward function equation is:

$$R = \omega_v r_v + \omega_\psi r_\psi(d, \Psi) \tag{13}$$

The variable ω represents the weight assigned to each component of the reward function. Modifying the weight value can impact the training outcome of the agent. Increasing the weight value for a particular component amplifies its influence on the final training outcome.

Chapter 4 Training Process

During the training process, the agent continuously interacts with the environment, gaining experience and using it to update its policy and value function. As training progresses, the agent gradually learns to choose the optimal action based on the state to maximize the desired return.

To initialize the environment, the state space, action space, reward function, and other parameters are defined, and before any strategy network has been established, the agent will randomly sample from the action distribution for training purposes. The next step in the training process is the initialization of the agent's state. The agent will randomly appear at a location in the environment, and since no policy network has been formed yet, training will involve random sampling from the action distribution. As the training progresses, the agent gradually learns to select actions based on the current state and the existing policy and receives the next state and the corresponding reward value from the environment. This process continues until the maximum number of training steps is reached. The state, action, reward value, and next state are stored in an experienced pool, and a random batch of experiences is used to update the policy and value function. This process is repeated until the maximum number of training steps is reached.

In the previous chapter, it was introduced that the observations used in this study include position and velocity information provided by the environment and image data as input values. Therefore, in this chapter, these two cases will be presented separately.

4.1 Training Process of Non-end-to-end Model

The loop-empty map in the Gym-duckietown environment was selected as the optimal training scenario due to its circular shape and two additional 45-degree turns, which facilitate the adaptation of the agent to the state during turns.

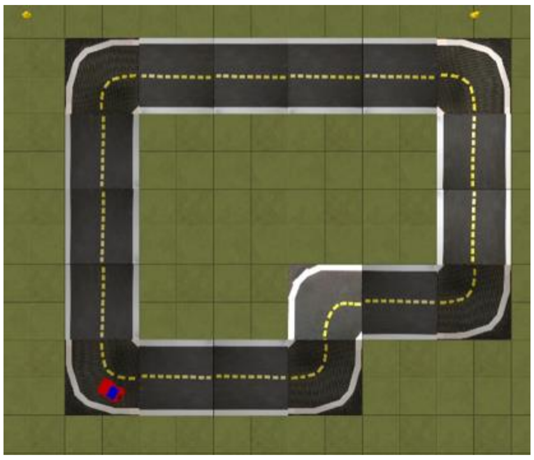


Fig. 4. 1: Larger-scale, relatively complex scenarios are used as training maps.

The multilayer perceptron strategy implemented in stable-baseline3 is utilized for this experiment due to the one-dimensional tensor input values, allowing for the use of a simple fully connected network. The architecture of the network is illustrated in Fig. 3.8 and Fig. 3.9.

To ensure the effectiveness of the training, meticulous consideration was given to the definition of both environment and network parameters. In this experiment, the learning rate is set to 0.0003, which controls the step size of the optimization process. The experience replay buffer size is set to 1000000, which stores the experiences of the agent for training. The sampling batch size is set to 256, which specifies the number of samples drawn from the experience buffer in each training iteration. The update frequency of the target network is set to 0.005, which controls the rate of updating the target network using the main network. The discount factor is set to 0.99, which controls the discount rate of the rewards. Finally, we set the maximum number of steps to 3000000, which specifies the length of the training process.

Tab. 2: Parameters for MlpPolicy.

Name	**Value**
Policy	MlpPolicy
Learning rate	0.0003
Buffer size	1000000
Batch size	256
Update frequency	0.005
Discount factor	0.99
Maximum time steps	3000000

Although the training process may extend over several hours, the computational speed is not significantly affected due to the one-dimensional tensor observations employed.

In the first training, the reward function equation is:

$$R = r_v + r_\psi(d, \Psi) \tag{14}$$

At present, each component of the reward function has an equal weight of 1.

4.2 Training Process of End-to-end Model

The second experiment involves the adaptation of one-dimensional observations into images, and training the model in the same scenario, as illustrated in Fig. 4.1. This process entails modifying not only the strategy and network structure but also entails longer training times. As previously mentioned, for this experiment, the "CnnPolicy" in Stable-Baseline3 was chosen as the optimal network structure and was trained using the sac algorithm [32].

"CnnPolicy" and "MlpPolicy" are two distinct policy networks within Stable-Baselines3, with differences in both their input values and network structures. The "MlpPolicy" is a multilayer perceptron neural network that can only accept one-dimensional data as input, processing it through fully connected layers. On the other hand, the "CnnPolicy" is a convolutional neural network that is capable of processing high-dimensional input data, such as images, using convolutional and pooling layers. Given these differences, the "CnnPolicy" policy network model is the more appropriate choice in this case.

Tab. 3: Parameters for Cnnpolicy.

Name	**Value**
Policy	CnnPolicy
Learning rate	0.0003
Buffer size	1000000
Batch size	256
Update frequency	0.005
Discount factor	0.99
Maximum time steps	3000000

During the initial stages of training, the carts execute random actions without employing any specific strategy, which leads to a high error rate. As a result, the carts frequently veer off the lane and travel only a few steps before encountering errors, resulting in a low bonus value. However, as the training progresses, the carts accumulate experience and continuously improve the updated model and driving strategy. Eventually, the agent becomes more proficient at completing the task and is capable of maximizing the reward function in the environment.

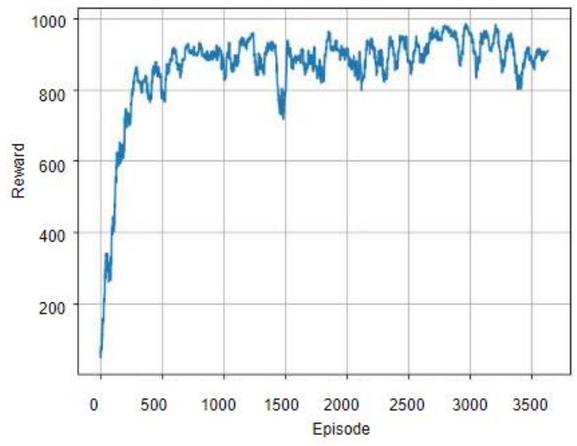
In the second training, the reward function equation is:

$$R = r_v + r_\psi(d, \Psi) \tag{15}$$

At present, each component of the reward function has an equal weight of 1.

Chapter 5 Result

To visualize the training progress, it is important to record the reward value change curve throughout the training process. Ideally, as the number of training iterations increases, the reward value should converge to a maximum value, indicating that the agent has learned an optimal strategy through accumulated experience that maximizes the reward value for a given state. The maximum number of time steps for training is set to three million, with a maximum of 500 steps within each episode. Therefore, according to the reward function formula, it can be calculated that the theoretical maximum reward value per episode is 1075.



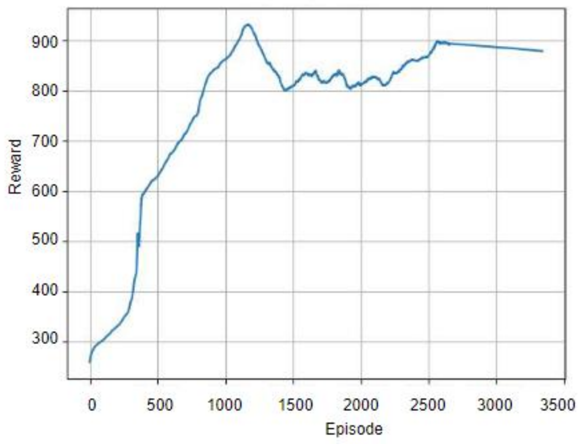


Fig. 5. 1: a: Reward function curve when trained a non-end-to-end model; b: Reward function curve when trained an end-to-end model. Due to the instability of the end-to-end model, a greater degree of smoothing was applied to (b) for better visualization.

Fig. 5.1 illustrates that the reward values of both models eventually converge to the highest achievable value. It is evident from the graph that the non-end-to-end model requires less training time and converges faster, indicating higher stability and learning efficiency during the training process. This is attributed to the fact that the low-dimensional data provided by the environment can be quickly fed into the neural network, and the model is stable and efficient to train since the neural network does not require extensive processing of the data, and the data can intuitively reflect the current agent state.

In contrast, the end-to-end model uses images as input to the neural network, which leads to longer processing times for the input values. Images alone do not offer an intuitive representation of the current agent state, causing the end-to-end model to be slower and less stable during training.

In terms of training time, the non-end-to-end model requires approximately four hours per million steps, while the end-to-end model takes sixteen hours for the same number of training steps.

In practice, several training attempts were required before achieving a well-trained model. The initial training failed due to various factors such as unreasonable reward function values and hyperparameters. Continuous debugging was performed to adjust the neural network hyperparameters and architecture, which helped to improve the learning efficiency and convergence rate. Additionally, the images were standardized by preprocessing them from their initial size of 480×640 to 32×32. The training environment was also adjusted to address the problem of unstable reward function curves.

Chapter 6 Test and Evaluation

In the testing phase of the end-to-end autonomous driving simulation based on a deep reinforcement learning algorithm, the environment used for testing is not identical to the training environment. The main change is in the selection of the map, which is intended to test the generalization ability of the model, as shown in Fig. 6.1. The pre-trained model is imported into the testing environment for evaluation. Testing is performed mainly for the end-to-end model, and the non-end-to-end model is used only for comparison.

The end-to-end model will undergo testing in two distinct scenarios, and each test will be initiated with the agent at a random location within the environment. After executing the maximum number of steps, the test will conclude, and the environment will record the specific data obtained during the test. The recorded data includes speed, the relative distance (lateral deviation), and the relative angle (orientation deviation) between the agent and the right lane's centerline. These data will be averaged, and the average will be printed out. The entire testing process will be repeated five times, and the results from each test will be aggregated for comparison. These tests aim to assess the model's performance such as adaptability and stability, and ability to maximize the reward value.

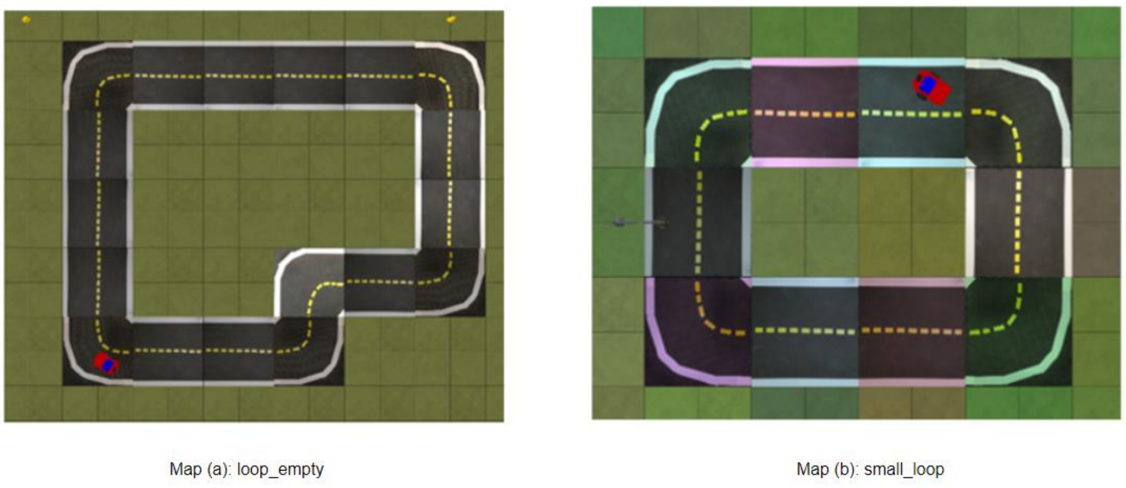


Fig. 6. 1:a shows the 'loop_empty' map, b shows the 'small_loop' map.

During the testing phase, the number of steps in each test iteration needs to be adjusted based on the scenario being tested. In this study, to comprehensively evaluate the performance of the trained model, 1000 steps are set per test iteration for map (a). For map b, which is smaller in size, 800 steps are used per test iteration. The purpose of this adjustment is to ensure that the model is tested thoroughly and accurately under different scenarios.

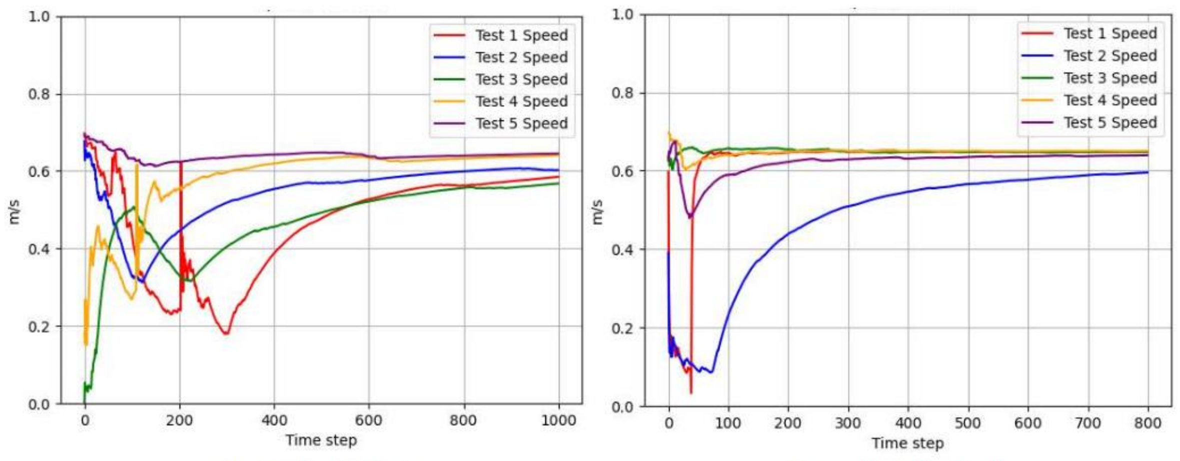
Tab. 4: Environmental parameters in both scenarios.

Name	Loop_empty (a)	Small_loop (b)
Scope	large	small
Number of tests	5	5
Number of iterations per test	1000	800
Start position of the agent	Randomization	Randomization
Maximum speed [m/s]	0.65	0.65

6.1 Evaluation of End-to-end Model

As mentioned previously, during the evaluation process, the trained model will undergo testing five times, with 1000 iterations in map (a) and 800 iterations the map (b) for each test, environmental parameters are shown in Tab. 4. Several important parameters will be recorded during testing, including speed, the relative distance, and the relative angle between the agent and the right lane's centerline, to evaluate the performance of the model.

6.1.1 Speed Tests in Two Scenarios



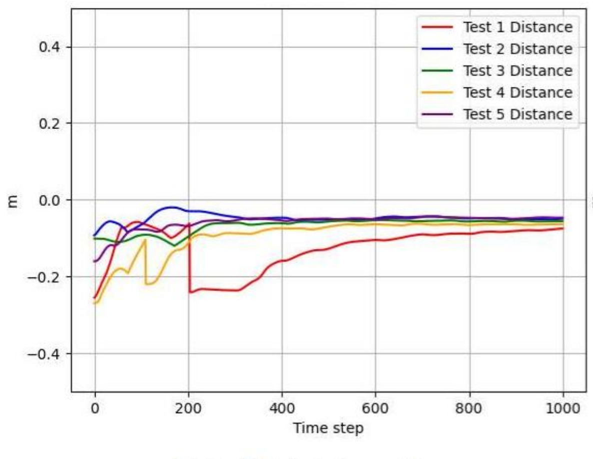
A. Speed test in the map (a)

B. Speed test in the map (b)

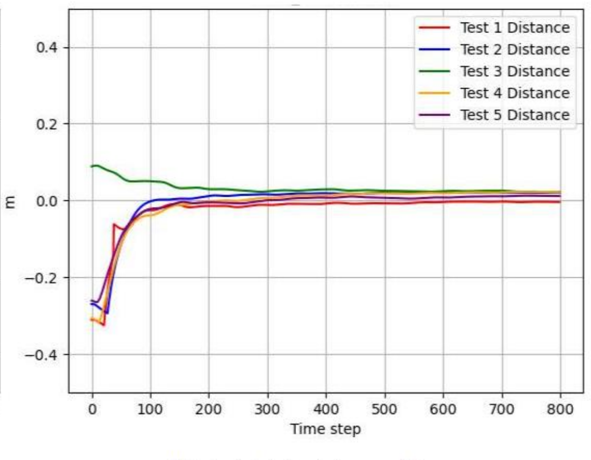
Fig. 6. 2: Speed variation of different scenarios.

In different scenarios, the speed of the agent converges to a value that is as close as possible to the maximum speed limit for that environment. As discussed in Chapter 4, since the weights of all parts of the reward function take a value of 1, the agent needs to consider the stability of the right lane following while achieving a higher speed. The figure above shows that at the beginning of the test, a large oscillation in speed occurs due to the randomization of the cart's position, causing it to constantly adjust its position and correctly drive on the right lane. During this process, the agent needs to accelerate, decelerate or slightly steer, which is the primary reason for the oscillation in the speed profile.

6.1.2 Lateral Deviation between the Agent and the Right Lane's Centerline



A. Lateral deviation in the map (a)



B. Lateral deviation in the map (b)

Fig. 6. 3: Lateral deviation in different scenarios.

The relative distance between the agent and the centerline of the right lane is a crucial metric in assessing the agent's ability to drive within the designated lane. Recording this data during the test facilitates the evaluation of the localization component of the reward function for agent policy learning. From the above figure, it is evident that in both scenarios, the distance between the agent and the lane median converges to the position of 0. This indicates that the cart has learned to drive within the right lane and follows the right lane median. Since the initial position of the agent for each test is random, the agent spends the beginning of the test searching for the lane median position, which reflects the impact of the relative distance of the localization component on the agent's learning of the right lane following strategy.

6.1.3 Orientation Deviation between the Agent and the Right Lane’s Centerline

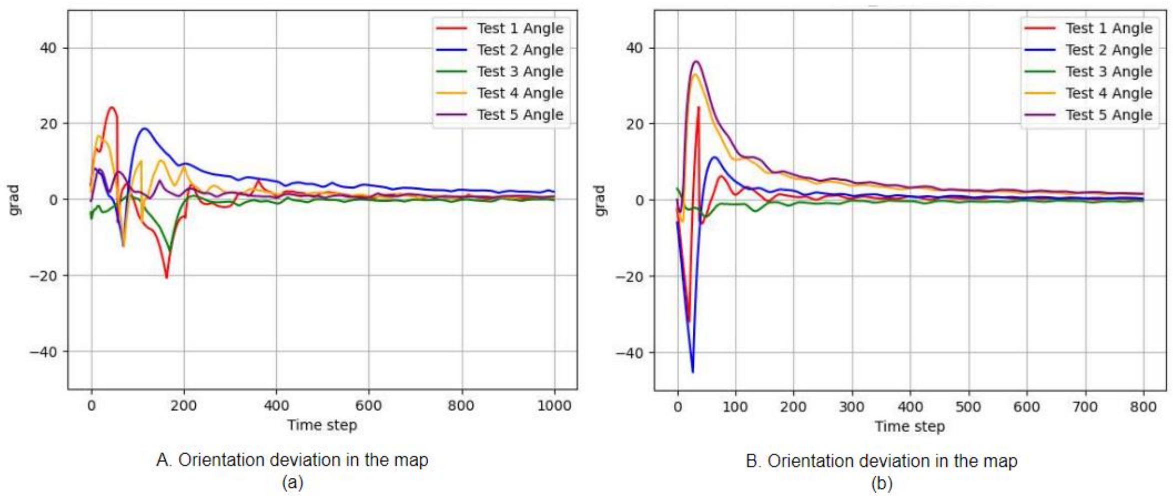


Fig. 6. 4: Orientation deviation in different scenarios.

The relative angle represents the orientation deviation between the agent and the right lane's centerline. A smaller relative pinch angle indicates that the agent can more closely follow the tangent vector of the right lane median, which is beneficial for the agent to learn the right lane following strategy more efficiently. Similarly to the relative distance, the initialization randomness causes the agent to continually adjust its position during the beginning of the test to minimize the relative pinch angle and maximize the reward value of the localization part. This demonstrates the impact of the relative pinch angle on the agent's learning of the right lane following strategy.

Tab. 5: Mean metrics of each test for two scenarios.

Name / Test_num	Mean speed [m/s]		Lateral deviation [m]		Orientation deviation [°]	
	Map a	Map b	Map a	Map b	Map a	Map b
1	0.46	0.61	-0.12	0.02	0.69	2.38
2	0.54	0.52	0.05	-0.02	4.64	-4.65
3	0.59	0.63	0.07	0.01	0.07	-0.20
4	0.58	0.63	0.09	-0.02	0.09	3.12
5	0.64	0.60	0.06	0.01	1.41	3.45

In summary, the end-to-end model takes images as input and learns the correct lane following policy to control the agent. For larger-scale maps, the environment becomes more complex, leading to decreased stability of the agent compared to the simpler environment. However, the agent is still able to attempt to drive along the runway to avoid mistakes that would cause the environment to reset. As a result, the driving speed of the cart may not remain stable in some states.

6.2 Comparison of the End-to-End Model to Two Baselines in Simulation

To evaluate the performance of the end-to-end model, two baselines are compared with the model. The first baseline is the non-end-to-end model, which is used as a benchmark to compare the performance of the end-to-end model. The second baseline is provided by the environment itself and is based on classical control theory using the relative distance and angle of the robot to the centerline of the lane. This baseline operates by controlling the robot to align itself towards a point on its intended path in the future and computing wheel velocities using a proportional-derivative (PD) controller, which is based on the orientation error of the robot [14]. Comparing these two baselines will provide a better understanding of the effectiveness of the end-to-end model in achieving the lane-following task.

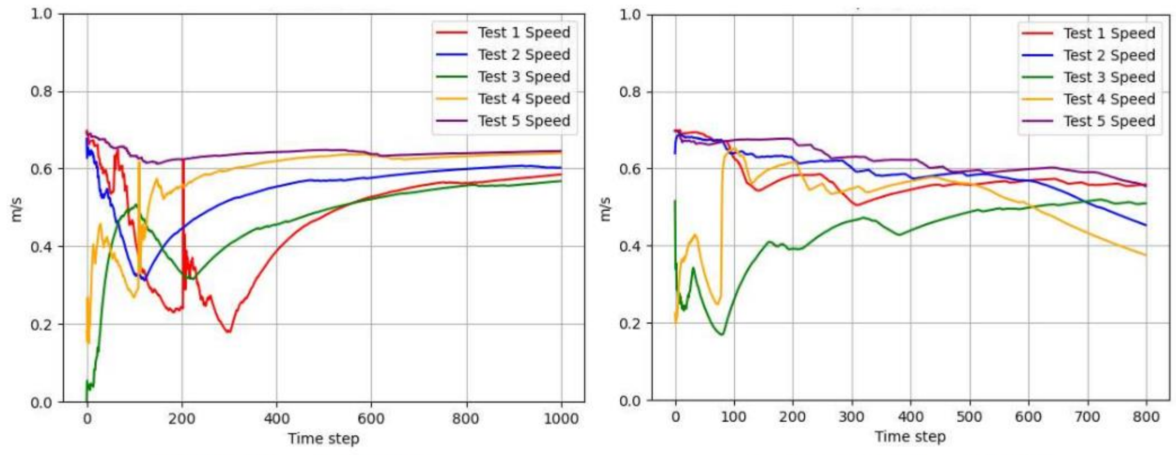
6.2.1 Comparison with Non-end-to-end Baseline

The non-end-to-end model takes one-dimensional processed data provided by the environment as input, and therefore cannot be strictly considered an end-to-end learning method. Models trained using this approach exhibit better performance, faster convergence rates, and higher learning efficiency. To compare the performance of the end-to-end and non-end-to-end models, tests are conducted in different environments, with specific monitoring data being recorded. To illustrate the differences between the two models, the test is conducted on map (a) and map (b). The environmental parameters are shown in Tab. 6:

Tab. 6: Environmental parameters

Name	End-to-end model	Non-end-to-end model
Env	Map (a), Map (b)	
Number of tests	5	5
Start position of the agent	Randomization	Randomization
Maximum speed [m/s]	0.65	0.65

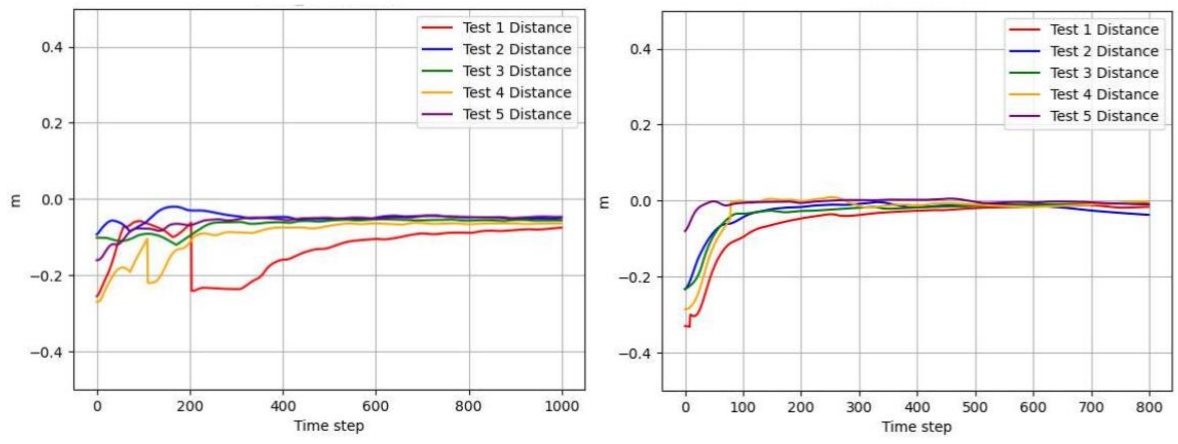
6.2.1.1 Comparison with Non-end-to-end Baseline in the Map (a)



A. Speed test of end-to-end model in the map (a)

B. Speed test of non-end-to-end model in the map (a)

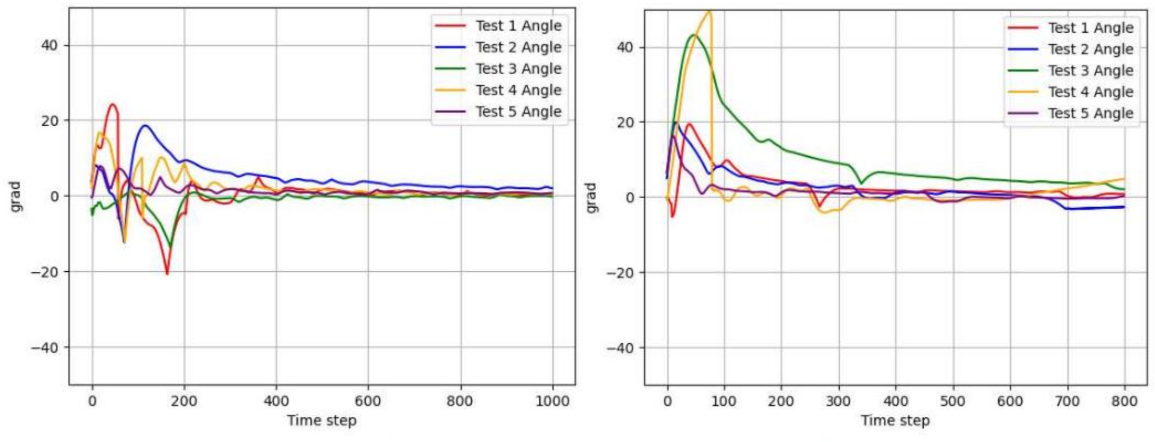
Fig. 6. 5: Comparison of speed variation in the map (a).



A. Lateral deviation of end-to-end model in the map (a)

B. Lateral deviation of non-end-to-end model in the map (a)

Fig. 6. 6: Comparison of Lateral deviation in the map (a).



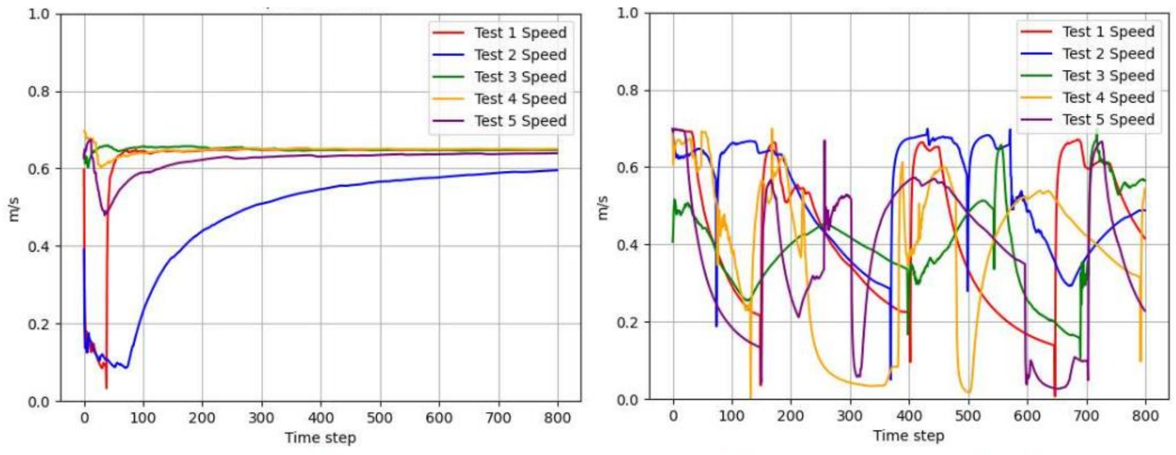
A. Orientation deviation test of end-to-end model in the map (a)

B. Orientation deviation test of non-end-to-end model in the map (a)

Fig. 6. 7: Comparison of orientation deviation in the map (a).

By analyzing the various metrics of the two models, it was observed that the end-to-end model outperforms the non-end-to-end baseline in terms of speed control as well as lateral and orientation deviations of the agent. The end-to-end model enables the agent to achieve and maintain the desired speed more rapidly and consistently, while the lateral and directional deviations converge to zero at a faster rate, indicating that the end-to-end model performs better than the non-end-to-end model in the lane following task.

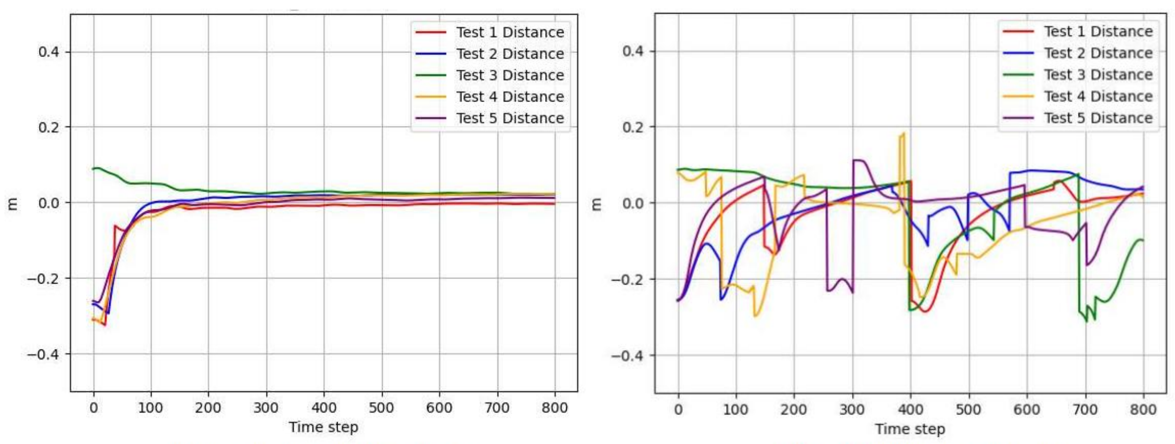
6.2.1.2 Comparison with Non-end-to-end Baseline in the Map (b)



A. Speed test of end-to-end model in the map (b)

B. Speed test of non-end-to-end model in the map (b)

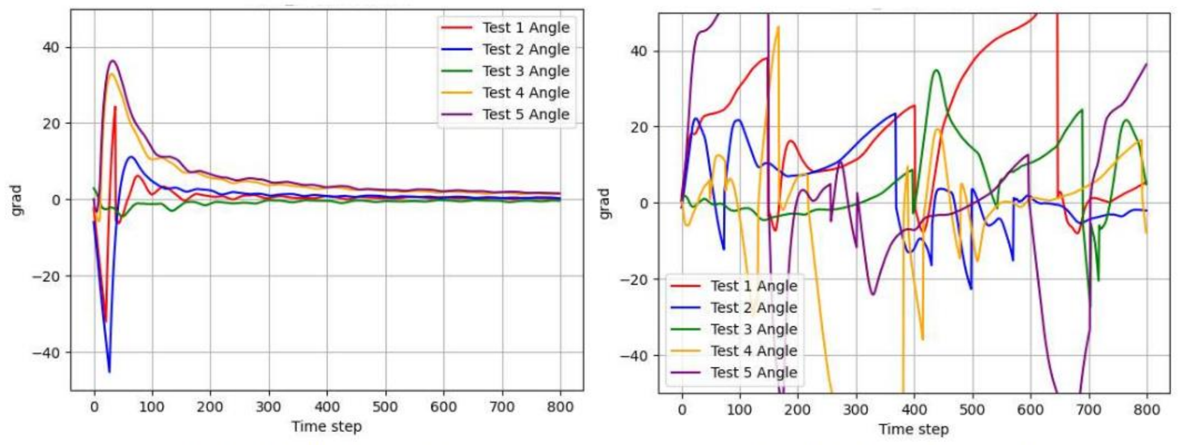
Fig. 6. 8: Comparison of speed variation in the map (b).



A. Lateral deviation of end-to-end model in the map (b)

B. Lateral deviation of non-end-to-end model in the map (b)

Fig. 6. 9: Comparison of Lateral deviation in the map (b).



A. Orientation deviation of end-to-end model in the map (b)

B. Orientation deviation of non-end-to-end model in the map (b)

Fig. 6. 10: Comparison of orientation deviation in the map (b).

As previously stated, the models are trained on map (a), which is larger and more complex than map (b). Models trained on map (a) also tend to perform better on the simpler map (b). After changing to map (b), the end-to-end model continues to perform relatively well, maximizing its speed while keeping the angle and distance to the right lane centerline close to 0. On the other hand, the non-end-to-end model performs very poorly on this new map. This indicates that the end-to-end model, trained using an integrated approach, has better adaptive capabilities and can perform more consistently even in changing environments.

One possible explanation for this issue is the limited input values of the non-end-to-end model, which result in an incorrect representation of the current state after changing the map, and affect the model's performance due to changes in environmental noise. This leads to overfitting of the model to the training map, which can impact the model's robustness in testing when the map changes.

In summary, while the non-end-to-end model may learn more efficiently during training and have better stability and effectiveness compared to the end-to-end model, it does not perform as well in testing as the end-to-end model. This is because the non-end-to-end model is overfitted to the training map. Conversely, the end-to-end model exhibits better robustness, as it can perform well even after changing the test environment. Its ability to maintain speed stability and perform the lane-following control task more accurately demonstrates this point.

6.2.2 Comparison with PD Baseline

The PD baseline is a test script provided by the platform itself, which serves as a control group for comparison with the end-to-end model. In contrast, the end-to-end model does not have direct access to the agent's dynamics parameters from the environment. This allows for a systematic comparison between the two models in an experimental setting.

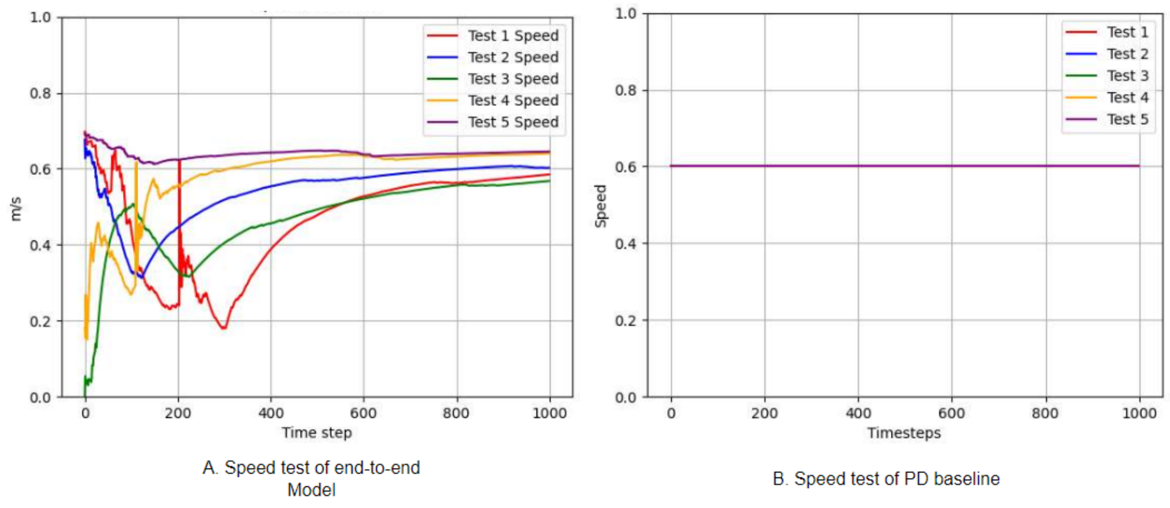
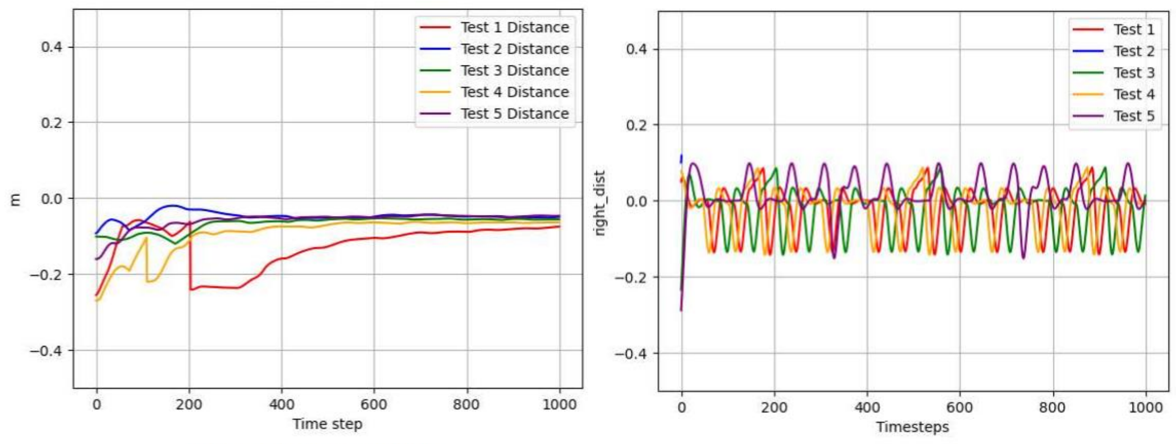


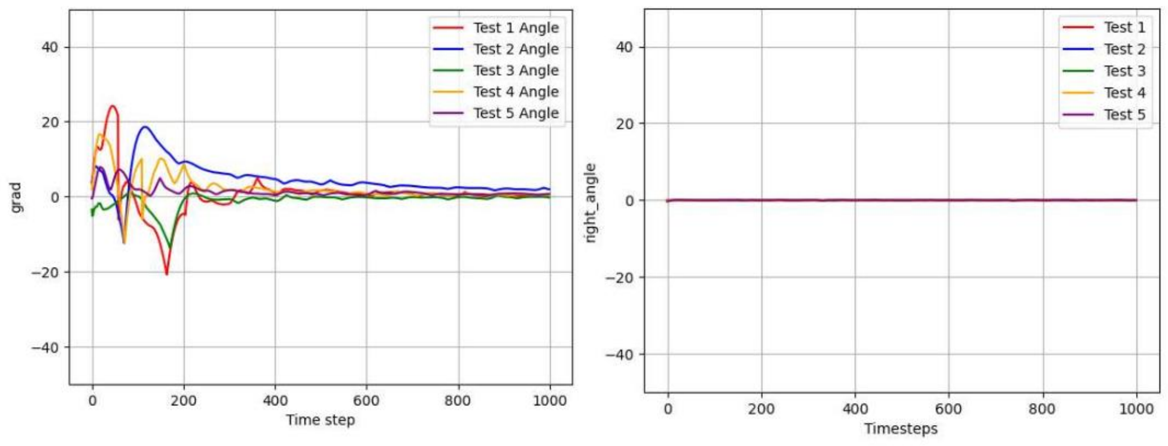
Fig. 6. 11: Speed test comparing the end-to-end model with PD baseline.



A. Lateral deviation of end-to-end Model

B. Lateral deviation of PD baseline

Fig. 6. 12: Lateral deviation comparing the end-to-end model with PD baseline.



A. Orientation deviation of end-to-end Model

B. Orientation deviation of PD baseline

Fig. 6. 13: Orientation deviation comparing the end-to-end model with PD baseline.

This set of control experiments will be conducted in the map (a). The PD baseline, which is provided by Gym-duckietown, serves as a standard that primarily reflects the cart's state when it completes the lane following task under ideal conditions. The PD baseline mainly controls the agent's distance and direction from the right lane's centerline, thereby keeping the speed constant. The change in relative distance is primarily due to fluctuations caused by frequent curves, which require the agent to maintain cornering stability under varying conditions. The relative angle is almost zero, which is the ideal state data.

By comparing the end-to-end model with the PD baseline, we find that although there are differences between the two, the model quickly achieves stability and meets the driving requirements, indicating its good performance. The end-to-end model can also control the cart's speed based on different states, leading to more stable driving and avoiding swaying from side to side in the lane.

Tab. 7: Comparison of the end-to-end model to two Baselined in map (a).

Mean metrics over 5 tests	End-to-end model	Non-end-to-end baseline	PD baseline
Speed [m/s]	0.56	0.54	0.6
Lateral deviation [m]	0.05	0.03	0.07
Orientation deviation [°]	1.99	4.17	0.15

6.3 Test the Effect of Different Weights Assignments on the End-to-end Model.

The reward function used in this study is formulated by Equation (14), which consists of two components: the orientation part and the speed part. This section aims to investigate the impact of varying the weights of the localization and velocity components on the performance of the agent. To conduct this investigation, multiple end-to-end models are trained, and their performance is analyzed by comparing the changes in velocity, relative distance, and relative angle. Therefore, it is necessary to adjust the weights of the reward function and evaluate the resulting changes in model behavior through testing and comparison. This approach will provide insights into the impact of reward functions on model learning and their ability to achieve the lane-following task.

The environmental parameters and the reward functions chosen for this experiment are listed in the following table:

Tab. 8: Choice of reward function for each model and choice of environment

Model / Name	Larger speed weighting model	General model	Larger orientation weighting model
Env	Map (b)		
Reward function formula	$R_1 = 2r_v + r_\psi$	$R_2 = r_v + r_\psi$	$R_3 = r_v + 2r_\psi$

Following the same procedure as the previous test, each model was tested five times with a maximum of 800 steps per test, and the changes in speed, lateral deviation, and orientation deviation were recorded for analysis.

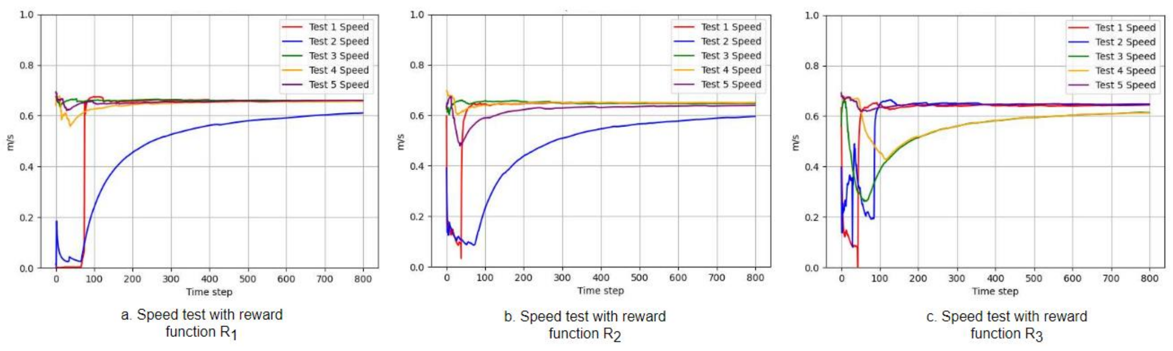
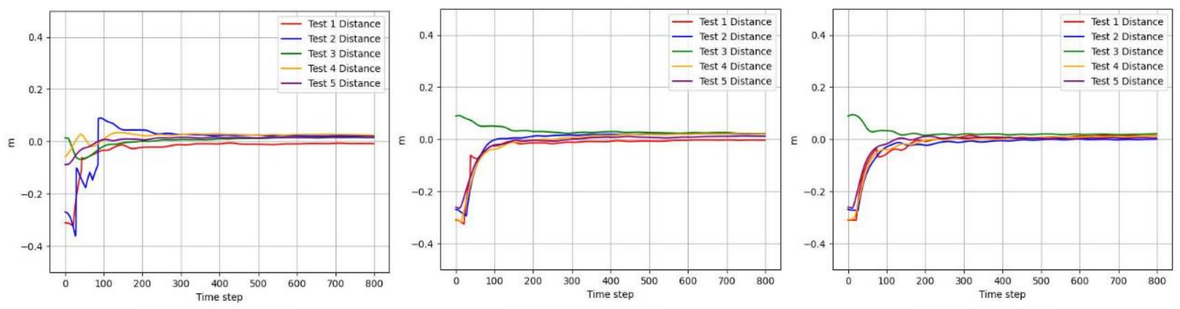


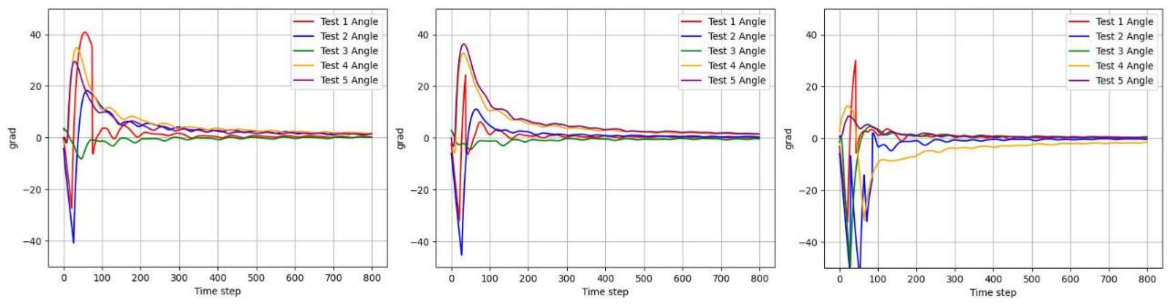
Fig. 6. 14: Speed curves for each model.

By comparing the speed changes of the three models during testing, it can be observed that the model with a higher weight on the speed component exhibits a greater ability to adjust the velocity. Specifically, this model can quickly increase the velocity to the maximum value after initialization. On the other hand, the other two models took longer to adjust the velocity and exhibited lower stability in doing so. Therefore, increasing the weight of the speed component can help promote the agent's ability to travel at a more stable and faster speed.



a. Lateral deviation with reward function R_1 b. Lateral deviation with reward function R_2 c. Lateral deviation with reward function R_3

Fig. 6. 15: Lateral deviation curves of each model.



a. Orientation deviation with reward function R_1 b. Orientation deviation with reward function R_2 c. Orientation deviation with reward function R_3

Fig. 6. 16: Orientation deviation curves of each model.

Based on the analysis of Fig. 6.15 and Fig. 6.16, it is observed that increasing the weight of the positioning component to 2 results in faster and more accurate convergence of the lateral deviation and directional deviation of the agent from the right lane centerline to 0. This implies that increasing the weight of the positioning component can enable the agent to drive more proficiently along the right lane centerline, thereby enhancing the safety and stability of the agent's driving and preventing the agent from driving into the reverse lane or swerving out of the lane.

Table. 9 shows the average value of each metric for different weighting reward functions for the five tests. It can be visualized that the change in weights has a subtle effect on the model performance:

Tab. 9: Mean metrics over 5 tests for 3 models with different reward functions.

Mean metrics over 5 tests	R_1	R_2	R_3
Speed [m/s]	0.61	0.60	0.59
Lateral deviation [m]	0.02	0.02	0.01
Orientation deviation [°]	3.21	2.76	2.00

Overall, adjusting the weights of the model can have an impact on its performance, but the model is still capable of completing the control task, just better at a particular capability.

Chapter 7 Conclusion

This paper investigates the performance of end-to-end and non-end-to-end models trained using the sac algorithm for policy networks in a Gym-duckietown environment for the lane following task, achieved through a deep reinforcement learning approach for agent control. The study demonstrates the sac algorithm's effectiveness for training end-to-end and non-end-to-end models for this task.

For the end-to-end model, the observation space is optimized for computational efficiency, requiring preprocessing of input images. The action space uses the box in the gym to define two continuous action outputs, turn angle, and velocity. Selecting the neural network and setting the reward function is challenging. In this study, a CNN network is used as the feature extractor for the policy network of the end-to-end model, with a linear layer mapping feature values to the output values. The reward function is divided into the orientation part the and speed part. The orientation and speed parts are crucial for the agent to complete the lane-following task. Specific parameters and weights of each part need to be selected through extensive training and continuous improvement.

Compared to non-end-to-end models, end-to-end models exhibit better robustness and are less prone to overfitting. Since the input to an end-to-end model is an image, adding noise helps to prevent overfitting. Furthermore, end-to-end models perform well in dynamic environments. In contrast, the non-end-to-end model trained on one map cannot perform the control task properly on another map due to differences in input locations. Although the performance of the end-to-end model is comparable to that of the baseline model provided by the Gym-duckietown platform and even superior in some respects, its stability in complex environments needs further improvement. Future research can explore using both image and environmental information as inputs to enhance the model's exploration of the environment, speed up training time, and improve the model's stability.

Reference

[1] Anderson J M, Nidhi K, Stanley K D, et al.: *Autonomous vehicle technology: A guide for the policymaker.* Rand Corporation, 2022.

[2] Buckley C.: *Beijing's Electric Bikes, the Wheels of E-Commerce, Face Traffic Backlash.* New York Times, 2016.

[3] World Health Organization.: *Global Health Observatory Data: Number of Road Traffic Deaths,* 2010.

[4] Eckstein L, Dittmar T, Zlocki A, et al.: *Automatisiertes Fahren—potenziale, Herausforderungen und Lösungsansätze.* ATZ- Automobiltechnische Zeitschrift, 2018, 120(3): 58-63.

[5] Mnih V, Kavukcuoglu K, Silver D, et al.: *Playing Atari with deep reinforcement learning*. arXiv preprint arXiv: 1312.5602, 2013.

[6] Paull L, Tani J, Ahn H, et al.: *Duckietown: an open, inexpensive and flexible platform for autonomy education and research*. 2017 IEEE international Conference on Robotics and Automation (ICRA). IEEE, 2017: 1497-1504.

[7] Pomerleau D A.: *Alvinn: An autonomous land vehicle in a neural network.* Advances in neural information processing systems,1998,1.

[8] Krizhevsky A, Sutskever I, Hinton G E, et al.: *Imagenet classification with deep convolutional neural networks*. Advances in neural information processing systems, 1097-1105, 2012.

[9] Mnih V, Kavukcuoglu K, Silver D, et al.: *Human-level control through deep reinforcement learning.* Nature, 2015, 518(7540): 529-533.

[10] Xia W, Li H, Li B.: *A control strategy of autonomous vehicles based on deep reinforcement learning.* International Symposium on Computational intelligence and Design (ISCID). IEEE, 2016, 2: 198-201.

[11] Jaritz M, De Charette R, Toromanoff M, et al.: *End-to-end race driving with deep reinforcement learning*. International Conference on Robotics and Automation (ICRA). IEEE, 2018: 2070-2075.

[12] Mnih V, Badia A P, Mirza M, et al.: *Asynchronous methods for deep reinforcement learning*. International conference on machine learning. PMLR, 2016: 1928-1937.

[13] Vitelli M, Nayebi A, Carma.: *A deep reinforcement learning approach to autonomous driving*. Tech. Rep. Stanford University., 2016.

[14] Kalapos A, Gór C, Moni R, et al.: *Sim-to-real reinforcement learning applied to end-to-end vehicle control*. International Symposium on Measurement and Control in Robotics (ISMCR). IEEE, 2020: 1-6.

[15] Huang Z, Zhang J, Tian R, et al.: *End-to-end autonomous driving decision based on deep reinforcement learning*. International Conference on Control, Automation and Robotics (ICCAR). IEEE, 2019: 658-662.

[16] Lillicrap T P, Hunt J J, Pritzel A, et al.: *Continuous control with deep reinforcement learning*, arXiv preprint arXiv: 1509.02971, 2015.

[17] Haarnoja T, Zhou A, Abbeel P, et al.: *Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor*. International conference on machine learning. PMLR, 2018: 1861-1870.

[18] Goldberg D E, Holland J H.: *Genetic algorithms and machine learning*. Machine Learning 3, 1988: 95-99.

[19] McCulloch W S, Pitts W.: *A logical calculus of the ideas immanent in nervous activity*. The bulletin of mathematical biophysics, 1943, 5: 115-133.

[20] Rosenblatt F.: *Perception simulation experiments*. Proceedings of the IRE, 1960, 48(3): 301-309.

[21] Rumelhart D E, Hinton G E, Williams R J.: *Learning internal representations by error propagation*. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[22] Ramstedt S.: *Deep Reinforcement Learning for continuous control*. Bachelor Thesis, Technical University of Darmstadt, 2016.

[23] Sharma S, Sharma S, Athaiya A.: *Activation functions in neural networks*. Towards Data Sci, 2017, 6(12): 310-316.

[24] Maas A L, Hannun A, Ng A Y: *Rectifier nonlinearities improve neutral network acoustic models*. Proc. Icml. 2013, 30(1): 3.

[25] Sewak M.: *Deep reinforcement Learning*. Singapore: Springer Singapore, 2019.

[26] Sutton R S, McAllester D, Singh S, et al.: *Policy gradient methods for reinforcement learning with function approximation*. Advances in information processing systems, 1999.

[27] Silver D, Huang A, Maddison C J, et al.: *Mastering the game of Go with deep neural networks and tree search*. nature, 2016, 529(7587): 484-489.

[28] Van Hasselt H, Guez A, Silver D.: *Deep reinforcement Learning with a double q-learning*. Proceedings of the AAAI conference on artificial intelligence, 2016, 30 (1).

[29] Williams R J.: *Simple statistical gradient-following algorithms for connectionist reinforcement learning*. Reinforcement learning, 1992.

[30] Aflaki S C.: *Teaching a Virtual Duckietown Agent to Stop*. University of Amsterdam, 2021.

[31] Haarnoja T, Zhou A, Hartikainen K, et al.: *Soft actor-critic algorithms and applications.* arXiv preprint arXiv: 1812.05905, 2018.

[32] Raffin A, Hill A, Ernestus M, et al.: *Stable baselines3*. https://github.com/DLR-RM/stable-baseline., 2019.

Acknowledgement

I would like to thank Mr. Li Dianzhao for his guidance and assistance with my thesis from November 2022 to May 2023.

Declaration of authorship

Declaration on my honor, I hereby declare on my honor that I have prepared the present work myself. The thoughts taken directly or indirectly from external sources are marked as such. The work has not yet been submitted to any other reviewing authority and has not yet been published. I am aware that an untrue statement will have legal consequence

Dresden,