Department of Informatics
Technical University of Munich

Master's Thesis in Electrical Engineering and Information Technology

Spiking Neural Network for Autonomous Navigation based on LiDAR Sensor

Spiking Neural Network für autonome Navigation basierend auf LiDAR-Sensor

Supervisor Prof. Dr.-Ing. habil. Alois C. Knoll

Advisor Genghang Zhuang, M.Eng.

Author Zhen Zhou, B.Sc.

Date January 28, 2023 in Garching

Disclaimer

I confirm that this Master's Thesis is my own work and I have documented all sources and material used.

Garching, January 28, 2023

(Zhen Zhou, B.Sc.)

Abstract

Deep Q-learning algorithms in reinforcement learning have been used in vehicle training, and this technology is relatively mature. However, artificial neural network (ANN) can suffer from long response times, high energy consumption, and high training costs due to their own connection methods. A new solution has been found in event-based spiking neural networks (SNN) that imitate biological neurons. In Reinforcement Learning tasks, SNNs can be trained to better control the direction of robot movement. This explores new ideas for the development of future autonomous driving technologies. In this thesis, a simulated lane tracking task is implemented using LiDAR. Firstly, the DQN algorithm and the Reward-modulated Spiking-Timing-Dependent-Plasticity (R-STDP) are compared by training on the same task. After coming up with the better R-STDP algorithm, the whole algorithm is optimized by different environment perception methods. What performs better among these is by combining Fully Convolutional Network (FCN) to predict the driving area and then obtain the lane edges by edge extraction as input state. Finally, the robustness of the algorithm is tested in scenarios of different complexity. Although the algorithm currently lacks the required reward prediction capability for more complex decision tasks, there are quite a few improvements in the adaptability to the environment and the response speed. Future optimization of this algorithm can be based on this algorithm to deal with more complex tasks in real-world situations.

Keywords: Deep Q-Learning (DQN), Artificial Neural Network (ANN), Spiking Neural Network (SNN), Autonomous Driving, LiDAR, Reward-modulated Spiking-Timing-Dependent-Plasticity (R-STDP), Fully Convolutional Network (FCN)

Zusammenfassung

Tiefe Q-Learning-Algorithmen im Bereich des verstärkenden Lernens wurden bereits für die Fahrzeugausbildung eingesetzt, und diese Technologie ist relativ ausgereift. Künstliche neuronale Netze (ANN) können jedoch aufgrund ihrer eigenen Verbindungsmethoden unter langen Reaktionszeiten, hohem Energieverbrauch und hohen Trainingskosten leiden. Eine neue Lösung wurde in ereignisbasierten spiking neural networks (SNN) gefunden, die biologische Neuronen imitieren. Bei Aufgaben des Reinforcement Learning können SNNs so trainiert werden, dass sie die Richtung der Roboterbewegung besser steuern. Dies eröffnet neue Ideen für die Entwicklung zukünftiger autonomer Fahrtechnologien. In dieser Arbeit wird eine simulierte Fahrspurverfolgungsaufgabe mit Hilfe von LiDAR implementiert. Zunächst werden der DQN-Algorithmus und der Reward-modulated Spiking-Timing-Dependent-Plasticity (R-STDP) durch Training an derselben Aufgabe verglichen. Nachdem der bessere R-STDP-Algorithmus gefunden wurde, wird der gesamte Algorithmus durch verschiedene Methoden der Umgebungswahrnehmung optimiert. Am besten funktioniert die Kombination von Fully Convolutional Networks (FCN) zur Vorhersage des Fahrbereichs und zur Ermittlung der Fahrbahnränder durch Kantenextraktion als Eingangszustand. Schließlich wird die Robustheit des Algorithmus in Szenarien unterschiedlicher Komplexität getestet. Obwohl der Algorithmus derzeit noch nicht die erforderliche Fähigkeit zur Belohnungsvorhersage für komplexere Entscheidungsaufgaben besitzt, gibt es einige Verbesserungen in der Anpassungsfähigkeit an die Umgebung und der Reaktionsgeschwindigkeit. Zukünftige Optimierungen dieses Algorithmus können auf diesem Algorithmus basieren, um komplexere Aufgaben in realen Situationen zu bewältigen.

Schlüsselwörter: Deep Q-Learning (DQN), Künstliches Neuronales Netz (ANN), Spiking Neural Network (SNN), Autonomes Fahren, LiDAR, Reward-modulated Spiking-Timing-Dependent-Plastizität (R-STDP) Fully Convolutional Network (FCN)

Contents

Chapter 1

Introduction

With the rapid development of artificial intelligence, Internet of Things, big data, and information and communication technologies, as well as the accelerated integration of automobiles with electronics, communications, and the Internet, the self-driving car industry will become a new global industry. Self-driving technology is a collection of many technologies, such as automatic control technology, artificial intelligence and vision processing. It is the product of highly developed computer science, pattern recognition and intelligent control technology. It is essentially an upgrade of the automotive industry and is an important indicator of a country's scientific research strength and industrial level[Lia20].

1.1 Autonomous Driving

Autonomous driving cars, also known as self-driving cars, are vehicles that require driver assistance or no control at all. As an automated vehicle, a self-driving car can sense its environment and navigate without the need for a human operator. This means that self-driving cars can autonomously collect and understand information about their surroundings, and make decisions and travel according to a set destination. The decisive role in this process is played by the autonomous driving system. An autonomous driving system consists of three main modules: sensing, decision making (localization and planning) and control[Lev11]. Roughly speaking, these three modules correspond to the eyes, brain and limbs of a biological system. The perception system (eyes) is responsible for understanding information about the surrounding obstacles and the road. The decision-making system (brain) decides the next action to be performed according to the surrounding environment and the set target. The control system (limbs) is responsible for executing these actions, such as steering, acceleration, braking, etc. Further, the perception system includes two tasks: environment perception and vehicle localization[Fay+20]. Environmental perception is responsible for detecting various moving and stationary obstacles (e.g., vehicles, pedestrians, buildings, etc.) and collecting various information on the road (e.g., driveable areas, lane lines, traffic signs, traffic lights, etc.), which requires the use of various sensors (e.g., cameras, LiDAR, millimeter wave radar, etc.). Vehicle positioning is based on the information obtained from environmental awareness to determine the location of the vehicle in the environment, which requires high-precision maps, as well as the assistance of inertial navigation (IMU) and global positioning system (GPS).

Because there is so much uncertainty in driving a vehicle on the road, there are also many challenges to achieve good robustness in autonomous driving. For example, LiDAR-based perception is susceptible to spoofing attacks[Sun+20]; or inaccurate locations of vehicles are matched to system maps due to unknown GPS noise[YCY03]. Errors like these can have seri-

ous consequences if they ever occur. So autonomous driving still has a long way to develop.

1.2 Sensors

Ultrasonic sensors, millimeter-wave radar, cameras, and LiDAR are indispensable sensors on self-driving vehicles today. For environmental sensing systems, different sensors have different physical principles, as well as their own advantages and disadvantages. Therefore, the following comparison will be based on the characteristics of the sensors and the usage scenarios.

1.2.1 Camera

The camera is the most commonly used sensor in sensing systems and has the advantage of being able to extract rich texture and color information, making it suitable for target classification. It is commonly used for obstacle classification, such as vehicles, pedestrians, lanes, traffic signs, etc. Commonly, front view cameras are used for lane recognition, traffic sign recognition. The rear view camera is used for parking assistance and recognition of rear obstacles. However, the disadvantage of relying on light to identify objects is that it has a weak perception of distance and is more affected by lighting conditions.

1.2.2 LiDAR

LiDAR has also been an important component of autonomous driving perception systems in order to obtain more accurate 3D information. It is commonly used for collision avoidance, pedestrian detection, semantic segmentation, etc. LiDAR data is a relatively sparse point cloud, which is very different from the dense grid structure of images, so algorithms commonly used in the image domain need to be modified to apply to point cloud data. The task of point cloud sensing can also be divided according to object detection, which outputs a 3D object border, and semantic segmentation, which outputs a semantic category for each point in the point cloud. To exploit algorithms in the image domain, point clouds can be transformed into a dense grid structure under Bird's Eye View or Range View. In addition, the Convolutional Neural Network (CNN) in deep learning can also be improved to make it suitable for sparse point cloud structures, such as PointNet or Graph Neural Network. shortcomings, making it suitable for target detection and ranging at medium to close distances. However, it has the disadvantages of higher cost, difficulty in mass production, limited sensing distance, and is more affected by extreme weather, such as heavy rain, snowstorm, etc.

1.2.3 Millimeter Wave Radar

Millimeter wave radar uses electromagnetic wave technology to transmit millimeter waves outward through the antenna and receive the reflected signal from the target. After processing, it quickly and accurately obtains information about the physical environment around the car body (such as the relative distance between the car and other objects, relative speed, angle, direction of motion, etc.). Then the target tracking and identification classification are carried out according to the detected object information, which is then combined with the dynamic body information for data fusion and finally processed intelligently by the central

processing unit (ECU). Millimeter wave radar has the characteristics of all-weather operation, can more accurately measure the speed and distance of the target, longer sensing distance, relatively low price, and is also widely used in automatic driving sensing systems. Millimeter wave radar data is also generally a point cloud, but it is more sparse and has lower spatial resolution than LiDAR, and has limited ability to sense stationary objects. Compared to cameras and LiDAR, the data density of millimeter wave radar is very low, so some traditional methods (such as clustering and Kalman filtering) do not perform much worse than deep learning, which are relatively less computationally intensive.

1.2.4 Ultrasonic Radar

Compared to millimeter wave radar relies on electromagnetic coupling propagation, ultrasonic radar relies on medium vibration transmission through mechanical waves. Ultrasonic radar is often referred to as "reverse radar" and is mainly designed for low-speed scenarios to detect obstacles within a few meters from the body. Ultrasonic radar uses sound waves back to detect the distance of obstacles. Therefore its distance sensing method is simple and low cost. However, it is susceptible to weather conditions and is only advantageous for short distance sensing[YXL16].

Sometimes, these sensors are also combined according to their characteristics. If they are combined together, more obstacles can be detected over a wide range. Using a multi-sensor fusion approach is a more mainstream solution for autonomous driving today. The results achieved are shown in the figure 1.1. The fused sensors can detect lane lines, vehicles, street signs, signals, pedestrians, and obstacles at the same time.

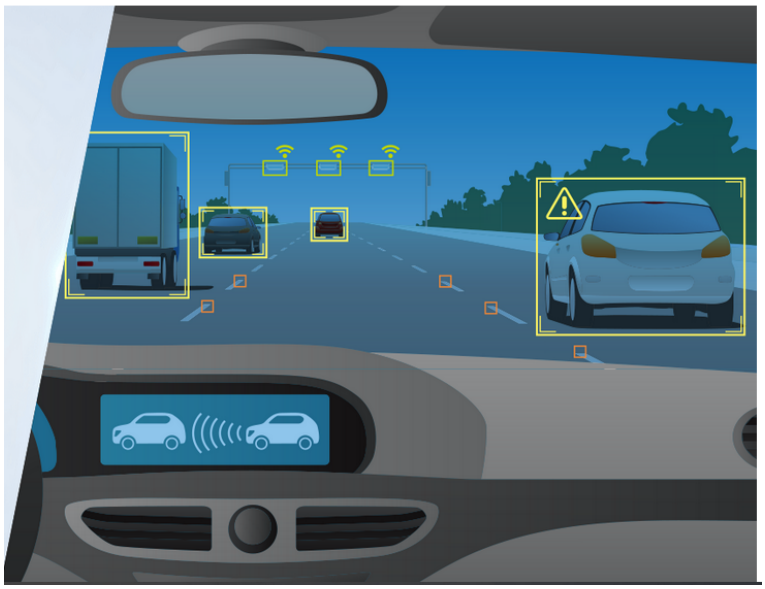

Figure 1.1: Sensor recognition

The automotive industry is a special industry with almost demanding requirements for safety and reliability, as any accident is unacceptable when the safety of passengers is involved. Therefore, the accuracy and robustness of sensors and algorithms are highly required in the research of self-driving cars. High-precision sensors facilitate accurate algorithm results, such as LiDAR. However, the cost and difficulty of manufacturing high-precision sensors is high, making them expensive. The cost of manufacturing sensors determines their market share. The sensors mentioned above are, in descending order of cost: ultrasonic radar, cam-

era, millimeter wave radar, and LiDAR. ultrasonic sensors have been widely deployed in parking assistance systems in modern cars due to their low cost, but other sensors are reserved for high-end features. The cost-performance trade-off may be the reason why car manufacturers (e.g., Tesla) have abandoned LiDAR [End17], but autonomous driving prototype developers (e.g., Google[DSB11] and Stanford[TT11]) tend to utilize all possible sensors. Autonomous driving cars will eventually be products for the average consumer, so manufacturers also need to control costs. The paradox of how to balance cost and performance has been difficult to resolve.

1.3 LiDAR Lane Keeping Task

As mentioned in the previous section, the hardware basis of an environment-aware system is the multiple sensors and their combinations, while the core of the software aspect is the perception algorithm. In general, perception algorithms perform two main tasks: object detection and semantic segmentation. The former obtains information about important targets in the scene, including position, size, speed, etc., a sparse representation, while the latter obtains semantic information about each location in the scene, such as drivable, obstacles, etc., a dense representation. The combination of these two tasks is called panoramic segmentation, which is a concept that has recently emerged in the field of autonomous driving and robotics[Yan+19]. For object targets (e.g. vehicles, pedestrians), panoramic segmentation outputs their segmentation Mask, category and instance ID; for non-object targets (e.g. roads, buildings), only their segmentation Mask and category are output. The ultimate goal of the environment sensing system is to get the panoramic segmentation results in the 3D space around the vehicle.

The explosion of autonomous driving technology this round largely comes from the breakthrough of deep learning in the field of computer vision, and this breakthrough starts with image classification and object detection in images. In autonomous driving environment perception, the first task in which traditional deep learning has been applied is object detection in a single two-dimensional image. The classical algorithms in this field, such as Faster R-CNN, YOLO, CenterNet, etc., have been the mainstream of visual perception algorithms[Shi+17] at different times. However, a vehicle cannot drive solely on the detection results on a single 2D image. Therefore, in order to meet the needs of autonomous driving applications, these basic algorithms need to be further extended, the most important of which is the fusion of temporal and 3D information. The former derives object tracking algorithms and the latter derives monocular/binocular/multi-object 3D object detection algorithms. By analogy, semantic segmentation encompasses image semantic segmentation, video semantic segmentation, and dense depth estimation. Specific algorithms for real-time semantic segmentation for autonomous driving are mentioned in[Sia+18] and will not be explored in depth in this paper.

In order to operate autonomous vehicles in urban environments with unpredictable traffic, multiple real-time systems must cooperate with each other, including environment sensing, localization, planning, and control. Traditional neural networks are limited by neurons, resulting in high computational costs for deep networks requirements are huge. Training is not only very time-consuming and labor-intensive, but often results in high response latency. If such a high memory is only used for the directional control of the vehicle, the car will not be able to drive properly in urban areas with complex environments. Therefore, it is necessary to increase the computational power of the network model. In the paper[GA09], the authors compare the computational power of a formal model of spiking neuron networks with other neural network models based on the computational power of McCulloch Pitts neurons (i.e.,

threshold gates) and sigmoid gates. The results show that the spiking neuron network has more computational power than these other neural network models in terms of the number of neurons required. A specific biologically relevant function is shown, which can be computed by a single spiking neuron (for biologically reasonable values of its parameters), but it requires hundreds of hidden units on a sigmoid neural network. On the other hand, it is well known that any function that can be computed by a small sine wave neural network can also be computed by a small network of bursting neurons.

In existing papers, SNNs are used to model the biological features and information processing capabilities of the neural structures present in the brain. In these neural networks, groups of neurons are connected by synapses, which can be enhanced or inhibited depending on the situation. This process of changing the strength of the connections (also called synaptic plasticity) is the learning process. Since the output of neurons is a temporal sequence of impulses, timing plays an important role in the output characteristics. There is evidence that synaptic adaptation between pre- and postsynaptic neurons depends on the difference in firing times between them, following a Herb learning behavior called STDP[Ger+14]. STDP can have different forms of synaptic plasticity, such as classical STDP, bidirectional STDP, and classical STDP with add-on (LTD)[BM10]. one of the most commonly used forms in SNN is the pair-based STDP model, in which the strength of synaptic connections is enhanced when presynaptic spikes appear before postsynaptic spikes, while synapses are inhibited if postsynaptic spikes appear before presynaptic spikes[Ger+14].STDP is an unsupervised learning method, as it changes synaptic weights based on the timing of specific spikes of presynaptic and postsynaptic neurons. In some cases, this learning technique is not sufficient and reinforcement learning strategies need to be applied to improve the performance of the application. Examples of this are in the field of robotics: line keeping vehicles[**bing2018end**], snake robots[Bin+19b], obstacle avoidance[Bin+19a] or movements using robotic arms to reach a target[Tie+19].

A reward-regulated spike-time-dependent plasticity (R-STDP)[Izh07], a learning rule that combines global reward signals and STDPs, has been recently proposed. In biology, the approach aims to mimic the function of those neuromodulators, which are chemicals released in the human brain, such as dopamine. Neurotransmitters such as dopamine act as reward prediction error signals, providing a mechanism for global modification of synapses. Thus, R-STDP is useful for robot control because it simplifies the requirement for external training signals, leading to more complex tasks and thus shorter reaction times. Autonomous driving is also essentially a process in which the car senses changes in the external environment through sensors (LiDAR, etc.) to make decisions and is trained repeatedly to eventually obtain a suitable autonomous driving algorithm. So this training process requires a combination of reinforcement learning and deep learning. For better comparison, the most commonly used Deep Q-Learning (DQN) is selected in this paper, which is compared with the combination of the above SNN transformed into R-STDP. The robotic car is placed in different scenarios for training to observe the learning efficiency of both.

1.4 Contributions

This paper focuses on exploring an SNN training algorithm based on the R-STDP learning rule and implementing it for end-to-end control in the robotics domain. Our main contributions are outlined below. First, build a simulated lane environment, in which the Pioneer robot loaded with dynamic radar sensors can input the radar signals detected by the vehicle, and use different algorithms, namely DQN and R-STDP for evaluation and comparison, and finally choose the more efficient R-STDP Continue to follow-up experiments. Secondly, we

improved the way of radar input, introduced FCN lane recognition network, changed the original simpler way to image recognition input, and compared whether it can improve the efficiency of training. All neurons in our network are fully connected with R-STDP synapses, and the network is directly trained to learn synaptic weights alone. The SNN reward for each motor is separately defined as a linear function of lane center distance. The network was implemented in NEST using the STDP dopamine synapse model and trained using rewards calculated from the distance between the robot and the lane. Finally, we analyze the simulation results of the event-based neural network to demonstrate the feasibility on different lanes.

1.5 Chapter Organization

As seen above, the first chapter provides an introduction to autonomous driving, which includes the autonomous driving system, sensors, and lane keeping algorithms. In following Chapter 2, the basic theoretical knowledge about autonomous driving is presented, which includes DQN, SNN, and the core component LiDAR sensors. Chapter 3 describes the main control algorithms used in this thesis. The experiment specific simulation environment as well as the experimental scenarios will be described in Chapter 4. Then in Chapter 5, we analyze the experimental results. Finally, we summarize the thesis and propose feasible improvements in Chapter 6.

Chapter 2

Knowledge Background

This chapter focuses on explaining the theoretical foundations used in this paper.

2.1 Reinforcement Learning: Deep Q-Learning

Reinforcement learning is a learning process in which it learns how to map the environment to actions so that rewards are maximized. In the general case, an action can affect not only the immediate reward, but also the next state, and through the next state, the subsequent reward. These two features - trial and error and delayed reward - are the two most important distinguishing features of reinforcement learning [Ric18].

Beyond individuals and environments, reinforcement learning systems generally have four elements: a policy, a reward signal, a value function, and an optional environment model.

Strategies define how the learning individual behaves at a given time. In simple terms, a policy is a mapping from perceived environmental states to actions to take in those states. It corresponds to what is called a set of stimulus-response rules or associations in psychology. In some cases, the strategy can be a simple function or lookup table, while in other cases it can involve extensive computation, such as a search process. Policy is at the heart of reinforcement learning individuals, because it is sufficient in itself to determine behavior. In general, the policy can be random in terms of specifying the probability of each action.

The reward signal defines the goal of the reinforcement learning problem. At each time step, a single number sent by the environment to the reinforcement learning individual is called a reward. The sole goal of an individual is to maximize the total reward it receives over time. Thus, reward signals define good and bad relative to the individual. In biological systems, we might think of rewards as similar to pleasant or painful experiences. They are immediate and explicit features of the problems faced by individuals. The reward signal is the main basis for changing the policy, if the action chosen by the policy is followed by a low reward, the policy can be changed to choose some other action in that situation in the future. Typically, the reward signal can be a random function of the state of the environment and the action taken.

The value function specifies long-term benefits, while the reward signal only indicates immediate benefits. Roughly speaking, the value of a state is the total amount of benefits an individual can expect to accrue in the future from that state. While rewards determine the immediate, intrinsic value of an environmental state, value indicates the long-term value of each state after taking into account the states that may be followed and the rewards available in those states. For example, a state may always yield a lower immediate reward, but still have a higher value because subsequent states often yield high rewards. Or the

opposite. In a human analogy, rewards are a bit like pleasure (if high reward) and pain (if low reward), while value corresponds to our more precise and far-sighted judgment of whether our environment is happy or unhappy in a particular state .

An environment model is a simulation of the environment, or more generally, it makes inferences about the behavior of the environment. For example, given a state and action, the model can predict the next state and the next reward for the outcome. Models are used for planning, and we mean anticipating the future before taking action. Approaches to solving reinforcement learning problems using models and planning are called model-based approaches, rather than the simpler non-model-based approaches, which are almost seen as the inverse of planning, which learns by trial and error. In Chapter 8, we'll explore reinforcement learning systems, which learn by trial and error, learn models of the environment, and use the models for planning. Modern reinforcement learning has leaped from low-level, trial-and-error learning to high-level, planned learning.

The reinforcement learning system has four core elements: Policy, reward function (Reward Function), Value Function and Environment Model, of which the Environment Model is optional.

Policy: Defines how the agent behaves at a specific time. A policy is a mapping of environmental states to actions. Reward function: defines the objective in reinforcement learning problems. At each step, the environment sends the agent a scalar numerical value called payoff. Value function: Indicates what is good in the long run. The value of a state is an agent's expectation of the total future cumulative benefit starting from that state. Environment model: is a simulation of the reaction pattern of the environment, which allows inferences about the behavior of the external environment. Reinforcement learning is a computational approach to understanding and automating goal-directed learning and decision-making problems. It emphasizes that the agent learns through direct interaction with the environment without the need for replicable supervisory signals or full modeling of the surrounding environment, and thus has a different paradigm compared to other computational methods.

Reinforcement learning uses the formal framework of Markov decision processes to define the interaction process of a learning agent with the environment using states, actions, and rewards. This framework seeks to simply represent several important features of artificial intelligence problems, including the perception of causality, the perception of uncertainty, and the perception of the existence of explicit goals.

Value and value function are important features of reinforcement learning methods, and value function is very important for efficient search in policy space. Compared with evolutionary methods, which directly search the policy space guided by repeated evaluation of the complete policy, the use of value functions is the difference between reinforcement learning methods and evolutionary methods.

The Deep Q-Learning(DQN) algorithm is an effective reinforcement learning method, which can be regarded as an extension of the classic Q-learning algorithm proposed by Watkins in 1992[Chr92]. In order to understand the necessity of DQN, we need to understand the principle of Q-learning algorithm. First we will initialize a Q-table to record the value of the state-action pair, and each step in each episode will update the Q-table according to the following Equation 2.1:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t)] \tag{2.1}$$

After updating a certain episode, the final Q-table will converge to a matrix. The optimal solution can be found according to the larger value in the Q-table. For simple problems, where both state space and action space are small, Q-Learning works well. In practice, there are huge state spaces or action spaces in most of the problems. If you want to build a Q table,

the memory is absolutely not allowed, and the amount of data and time overhead also need to be considered.

To solve the problem of too large state space, Value Function Approximation [Vol13], also known as the "Curse of Dimensionality", was proposed. As Equation 2.2 shows, we need to derive ω, called the "weight", to fit a suitable function as the value function. Combined with some supervised learning algorithms in the machine learning algorithm, the features of the input state are extracted as input, the value function is calculated by Monte Carlo method(MC) [Ric98] or Temporal-Difference Learning(TD) [Dav12] as the output, and then the function parameter ω is trained until convergence. Here we mainly talk about regression algorithms, such as linear regression, decision trees, neural networks, etc.

$$\hat{Q}(s,a,\omega) \approx Q_{\pi}(s,a) \tag{2.2}$$

By expressing $Q(s,a)$ as a function instead of a Q-table, the convolutional neural network generates the target Q value, evaluates the Q value of the next state based on the target Q value in that state. The Q table of Q-Learning becomes Q-Network, which is Deep Q-Network, and the above-mentioned process of approximating the Q value through the value function to solve the Q-Network is also called Deep Q-Learning (DQN).

In fact, DQN is the combination of Q-Learning and neural network. So how to train this network to obtain and determine the network parameter ω is particularly important. First, we need a Loss Function; second, we need enough training samples. Training samples can be generated through the $\varepsilon - greedy$ strategy. In Q-Learning, the update of the Q table is iterated using the reward of each step and the current Q table as Equation 2.1. Then we can use this calculated Q value as the "label" of supervised learning to design the Loss Function. We use the following form, that is, the mean square error of the approximate value and the true value to get the Loss Function as follows 2.3.

$$L(\omega) = \mathbb{E}_{\pi}[(q_{\pi}(s,a) - \hat{q}(s,a,\omega))^2] \tag{2.3}$$

Using stochastic gradient descent (SGD) [Bot12] to solve it iteratively, we can obtain ω. The $q_{\pi}(s,a)$ in the above Equation 2.3 is expressed in the following form 2.4 in Q-Learning:

$$L(\omega) = \mathbb{E}_{\pi}[(R_{t+1} + \gamma \max \hat{q}(s_{t+1}, a_{t+1}, \omega) - \hat{q}(s,a,\omega))^2] \tag{2.4}$$

Furthermore, the optimization algorithm becomes a local minima rather than a global minima due to the correlation between subsequent empirical samples. This will cause DQN to fail to converge using neural network training. In order to break the association between data, Experience replay [Vol15] is necessary, which is a memory that stores the past experiences to randomly reuse them so that there is no continuity between samples. Experience replay technology improves the training speed and stability of DQN by caching the state, action, reward, and next state tuple of each step, and batch training multiple times after one round ends. In detail, it specifically implements maintaining a cache array of a specified size, and randomly replace the existing N in the cache pool with the newly generated N state, action, reward, and next state tuples each round, and then do several trainings after the round ends.

2.2 Spiking Neural Network

The human brain is a very complex system consisting of approximately 90 billion neurons. The brain's ability to deal with complex problems has inspired many researchers to explore processing functions and learning mechanisms. In the process of exploration, Spiking Neural

Network(SNN) came into being. SNN can deal with complex problems in a flexible way, and has great potential to model complex information processing as the brain. In the following content, SNN will be comprehensively introduced from three aspects: biological background, topological structure, and learning algorithm.

2.2.1 Biological Background

Neurons, which communicate by sending and receiving action potentials, are the most basic unit of information processing in the brain. Neurons are connected to each other through synapses. The strength of synaptic connections and their connection methods play an important role in the information processing of the nervous system.

The human brain is capable of performing amazing tasks (eg, recognizing multiple objects simultaneously, reasoning, controlling, and moving) with minimal energy expenditure. Although the human brain has not been exhaustively explored, from a neuroscience perspective, the extraordinary capabilities of the human brain can be attributed to three fundamental observations: extensive connectivity, structural and functional levels of organization, and time-dependent Neuronal synaptic connections.

Neurons are the computational primitive elements of the human brain, which exchange and transmit information through discrete action potentials or "pulses". The connections between nerve cells are called synapses. Synapses are the basic storage elements of memory and learning. The human brain has a network of billions of neurons, interconnected by trillions of synapses. One of the most important biological functions is synaptic plasticity, which is the ability to adjust the strength of nerve cell connections. Synaptic plasticity arises for a variety of reasons, such as: changes in the amount of neurotransmitters released in synapses, and how efficiently cells respond to neurotransmitters. Synaptic plasticity is considered to be an important neurochemical basis for memory and learning. As shown in Figure 2.1 a, during neuronal messaging, when a neuron is stimulated by a signal from the environment or another neuron, neurotransmitters stored in presynaptic vesicles can be released into the synaptic cleft, act on the corresponding receptors in the postsynaptic membrane, be captured by the postsynaptic receptors, and transmit the transmitter signal to the next neuron. As shown in Figure 2.1 b, a postsynaplic AMPA receptor of an excitatory synapse binds to glutamate (excitatory neurotransmitter) when sodium and potassium ions can flow across the membrane. Thus the process of one transmission at the synapse is complete.

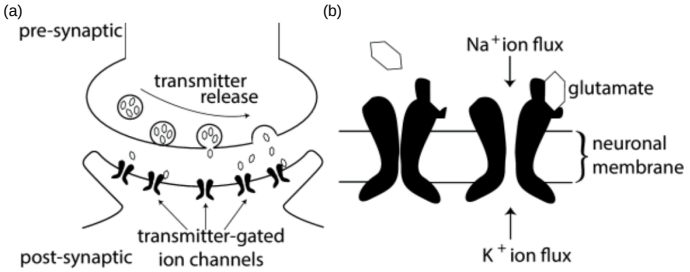

Figure 2.1: Synapses[Ger+14]: (a)Schema of synaptic transmission. (b) Schema of a postsynaplic AMPA receptor of an excitatory synapse.

In 1952, Hodgkin and Huxley [Ger+14] conducted experiments on giant axons in octopuses and built a 4-dimensional detailed point-to-point neuron model to reproduce the results of electrophysiological tests. However, due to the high complexity of this model, it

was eventually replaced by a more streamlined spiking neuron model, which was applied to the simulation algorithm of SNN.

2.2.2 Topological Structure

Like the traditional artificial neural network, the spiking neural network is also divided into three topological structures. They are feed-forward spiking neural network, recurrent spiking neural network and hybrid spiking neural network.

Feed-Forward Spiking Neural Network

As shown in Figure 2.2 (a), in the multi-layer feedforward spiking neural network structure, the neurons in the network are arranged in layers. The spike sequence of each neuron in the input layer represents the encoding of the input data of the specific problem, which is input to the next layer of the spiking neural network. The last layer is the output layer, and the pulse sequence output by each neuron in this layer constitutes the output of the network. There can be one or more hidden layers between the input layer and the output layer.

In addition, in the traditional feedforward artificial neural network, there is only one synaptic connection between two neurons, while the spiking neural network can adopt a network structure with multiple synaptic connections, as shown in Figure 2.2 (b). There can be multiple synapses between two neurons. Each synapse has different delays and modifiable connection weights. Different time delays at polysynapses allow the pulses input from presynaptic neurons to affect the firing of postsynaptic neurons over longer time scales. Multiple pulses transmitted by presynaptic neurons generate different postsynaptic potentials according to the size of synaptic weights.

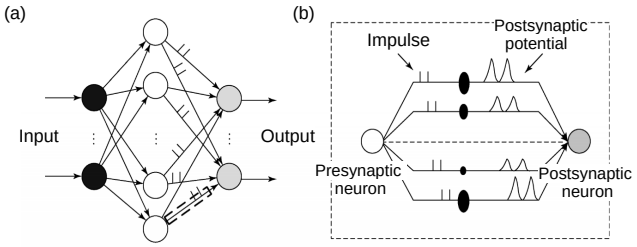

Figure 2.2: Multi-layer feedforward spiking neural network structure with multiple synaptic connections: (a) multi-layer feedforwardspiking neural network structure; (b) multiple synaptic connections between neurons

Recurrent Spiking Neural Network

Different from multi-layer feed-forward neural network and single-layer neural network, recurrent neural network has a feedback loop in the network structure, that is, the output of neurons in the network is a recursive function of the output of neurons on previous time steps, as shown in Figure 2.3. Recurrent neural networks can simulate time series to complete tasks such as control and prediction. On the one hand, their feedback mechanism enables them to express more complex time-varying systems; on the other hand, it also makes the design of effective learning algorithms and their convergence analysis more efficient. for difficulty. The two classic learning algorithms of the traditional recurrent artificial neural network are

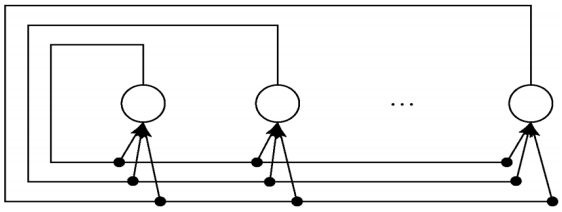

Figure 2.3: Global recurrent spiking neural network structure

the real-time recurrent learning algorithm and the backpropagation through time algorithm, which both calculate the gradient recursively.

Recurrent spiking neural network refers to a spiking neural network with a feedback loop in the network. Because its information coding and feedback mechanism are different from traditional recurrent artificial neural network, it is difficult to construct a learning algorithm and analyze the dynamics of the network. Recurrent spiking neural networks can be used to solve many complex problems, such as language modeling, handwritten digit recognition, and speech recognition. Recurrent spiking neural networks can be divided into two categories: fully recurrent spiking neural networks; the other is locally recurrent spiking neural networks.

Hybird Spiking Neural Network

Hybrid spiking neural network includes both feed-forward structure and recurrent structure. The following is a description of two typical hybrid networks, synchronous firing chain and liquid state machine.

Synchronous firing chain[Ger+12] is considered as an important mechanism to represent the relationship between different delay events. It is a multi-layer structure in which the neurons in the structure deliver a sequence of impulses that can be passed sequentially from one layer to the next. The feed-forward structure is used between the sub-layers of the synchronous firing chain. The recursive structure is used within the sub-layers, as shown in Figure 2.4. Based on such structural characteristics, synchronous firing chain is defined as a hybrid structure.

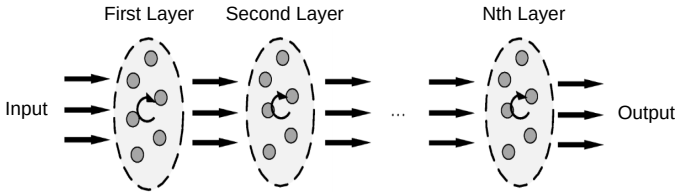

Figure 2.4: Multi-layer structure of synchronous firing chain

Maass et al[Maa11] proposed a new kind of neural network computational framework, called liquid state machine, whose structure is shown in Figure 2.5. Liquid state machine exploits the advantages of recurrent neural networks to improve the performance of neural networks for solving complex problems. Liquid state machine consists of a very large sparsely connected recurrent neural network forming a neural microcircuit. In biological nervous systems, the connections between neurons exhibit sparse recurrent structures, which are

called neural microcircuits[Coh14]. The recursive connection weights between the neurons in the neural microcircuit are fixed sound and are simply trained with appropriate output connection weights to obtain the desired target pulse sequence.

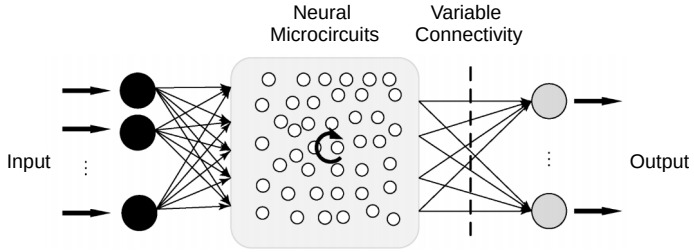

Figure 2.5: Liquid state machine structure

2.2.3 Neuron Model and Learning Algorithm

To describe the proposed structure, we review how neurons generate spikes, how synapses store learning, and how reward signals are generated.

Spiking Neuron

When an input stimulus is provided to a neuronal cell, shaped as an input current, the potential v_m of the membrane increases. Once the threshold voltage v_th is exceeded, the neuron generates a spike, and then it immediately resets its membrane potential to a reset voltage v_reset. The neuron cannot fire again until a certain refractory period has elapsed. Some differential equation models account for these neural dynamics with high biological plausibility, but with excessive computational cost, such as the Hodgkin-Huxley[NR98] or Izhikevich models[Izh04]. However, it is possible to approximate spikes of the same characteristics of biological neurons by reducing the amount of computation for membrane potentials, and thus reducing the accuracy. Therefore, the most commonly used model is the Leaky Integrate and Fire (LIF) model[SK20], which sacrifices some accuracy but has a small computational cost.

The kinetic equations of the LIF neuron model are as follows Equations 2.5:

$$\tau_m \frac{\mathrm{d}v_m(t)}{\mathrm{d}t} = -(v_m(t) - E_r) + R_m I_{syn}(t) \tag{2.5}$$

where $v_m(t)$ is the membrane potential of neuron, R_m represents the membrane resistance, $\tau_m = R_m C_m$ is the neuron's membrane decay time constant for $v_m(t)$, in which C_m is the neuron's membrane capacitance, E_r is the membrane resting potential of neuron, $I_{syn}(t)$ is the sum of the injected currents provided by the input synapses, which is calculated as follows:

$$I_{syn}(t) = W \cdot S(t) \tag{2.6}$$

where $W = [\omega_1, \omega_2, \ldots, \omega_n]$ is the weight vector as the synapse strength value between a presynaptic ith neuron and a postsynaptic jth neuron, $S(t) = [s_1(t); s_2(t); \ldots; s_n(t);]$ is the spatial-temporal pulse sequence, $s_i(t)$ is calculated as:

$$s_i(t) = \sum_f \delta(t - t_i^f) \tag{2.7}$$

where t_i^f is the fth pulse of the ith input pulse sequence, $s_i(t)$ represents the ith synapse, $\delta(t)$ is the Kronedecker delta function, which $\delta(x) = 1$ for $x = 0$ and $\delta(x) = 0$ for $x \neq 0$.

Each time a spike arrives at the neuron, $I_{syn}(t)$ increases. On the other hand, if no spikes arrive at the neuron, the current decays. This phenomenon is described by LIF conductance-based model [Lu+21], composed of Equations 2.5 and 2.8:

$$\tau_m \frac{\mathrm{d}I_{syn}(t)}{\mathrm{d}t} = -I_{syn}(t) + C_m \sum_i^N \omega_{ij} \delta(t - t_i^f) \tag{2.8}$$

As for each postsynaptic neuron, there can be N presynaptic neurons connected, t_i^f is then a vector with firing times from each of the N presynaptic neurons in Equation 2.8. It assumes all presynaptic spikes have been produced at time t. For each time a new spike happens, $t_i^f = t$, therefore, $\delta(t - t_i^f) = 1$. Once the neuron threshold voltage v_th is overpassed, the neuron spikes, emitting a pulse of magnitude v_spike, then, the neuron resets to a reset potential $v_reset = E_r$ and it starts integrating again. And it remains at the resting level for the duration of the non-response period t_ref. Figure 2.6A shows the LIF structure model[Jua+22], while its spiking activity for a given fixed and variable input current is shown at Figures 2.6B,C, respectively.

Synapse Based on Dependence

We have defined how neurons generate spikes and we will describe how synaptic strengths are tuned. Spike-timing-dependent plasticity (STDP) learning [Khe+18] is a SNN unsupervised learning algorithm based on the dependence between pre- and postsynaptic spikes and is applicable to learning event-based networks. It describes how synaptic weights are strengthened or weakened according to neural spike activity. First, a synaptic weight value is randomly assigned to each defined synapse. Then, the time difference between the pre- and postsynaptic firing times $\Delta = t_post - t_pre$ is calculated, which determines the rate of change Δw on the synaptic weight w as Equation 2.9 and 2.10:

$$W(\Delta t) = \begin{cases} A^+ e^{-\frac{\Delta t}{\tau_{post}}}, & \Delta t \geq 0 \\ -A^- e^{\frac{\Delta t}{\tau_{pre}}}, & \Delta t < 0 \end{cases} \tag{2.9}$$

$$\Delta\omega = \sum_{t_{pre}} \sum_{t_{post}} W(\Delta t) \tag{2.10}$$

Here, A^+, A^- are scaling constants depicting whether our synaptic weight has been incremented (Long-Term Potentiation LTP) or decremented (Long-Term Depression LTD). τ_{pre} and τ_{post} are positive time constants representing decay time. Δt is defined as the time difference between the pre- and post-synaptic spikes. This difference is then used in Eq. 9, where A_pre together with A_post (constant values) define the increase or decrease of the synaptic weight. Therefore, Δw is the change of the synaptic weight. In a given synapse, when a postsynaptic spike occurs within a specific time window after a presynaptic spike, the weight of this synapse increases. If the postsynaptic spike occurs before the presynaptic spike, the weight decreases assuming an inverse dependence between the presynaptic and postsynaptic spikes. The intensity of the weight change is a function of the time between the presynaptic and postsynaptic spike events[Iak+15]. The learning curve is shown in Figure 3 2.7.

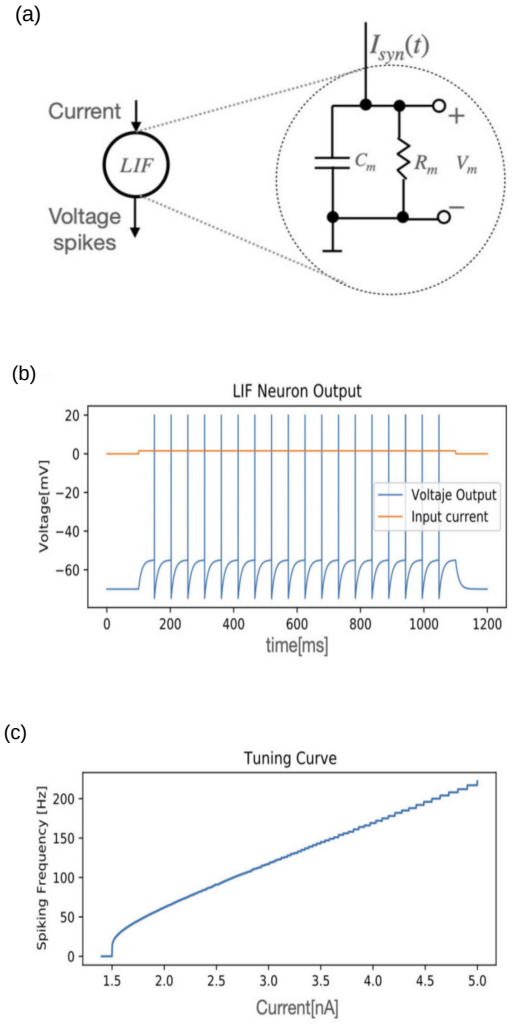

Figure 2.6: LIF neuron model[Jua+22]: (a) diagram of a spiking LIF neuron. (b) Spiking activity for a LIF neuron with $I_{syn} = 1.51nA$. (c) Tuning curve of the neuron model.

Firing and Action

Suppose there are two presynaptic neurons j=1, 2, both of which send spikes to postsynaptic neuron i. Neuron $j = 1$ fires spikes at $t_1^{(1)}$, $t_1^{(2)}$, ..., and similarly neuron $j = 2$ fires at $t_2^{(1)}$, $t_2^{(2)}$, ... Each spike evokes a postsynaptic potential ϵ_{i1} or ϵ_{i2}, respectively. As long as there are only few input spikes, the total change in potential is approximately the sum of the individual postsynaptic potentials(PSPs), as shown in the Equation 2.11:

$$u_i(t) = \sum_j \sum_f \epsilon_{ij}(t - t_j^f) + u_r est \tag{2.11}$$

That is, the response of the membrane potential to the input spikes is linear; see Figure 2.8b.

On the other hand, if too many input spikes arrive in a very short period of time, the linear response is interrupted. Once the membrane potential reaches a critical value of θ, its trajectory shows a completely different behavior from the simple summation of the PSP: the membrane potential exhibits a pulse-like excursion with an amplitude of about 100 mV. This short voltage pulse will propagate along the axon of neuron i to the synapse with other

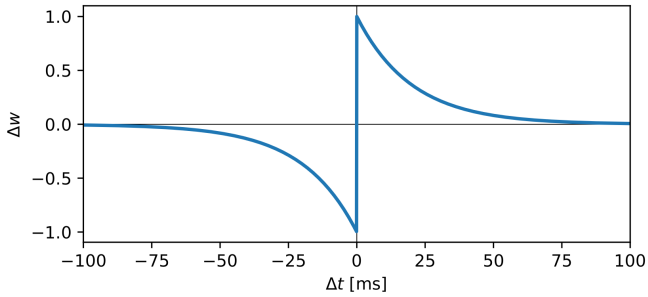

Figure 2.7: Learning curve of the STDP mechanism with $A^+ = 1.0$ and $A^- = 1.0$. τ_{pre} and τ_{post} are set to 20 ms. The same parameters were used for the simulations in this work as well.

neurons. After the pulse, the membrane potential does not return directly to the resting potential, but for many neuron types goes through a hyperpolarization phase below the resting value. This hyperpolarization is referred to as the "spike-afterpotential"[Ger+14].

The process by which most neurons fire and transmit spikes is shown in Figure 2.8: (a) Each presynaptic spike evokes an excitatory postsynaptic potential (EPSP) that can be measured with electrodes as the potential difference $u_i(t) - u_{rest}$. The time course of the EPSP elicited by the spike of neuron $j = 1$ is $\epsilon_{i1}(t - t_1^f)$. (b) Shortly after the spike from neuron $j = 1$, an input spike from a second presynaptic neuron j=2 causes a second postsynaptic potential that is summed to the first potential. (c) If $u_i(t)$ reaches the threshold value θ, an action potential is triggered. As a result, the membrane potential begins a large positive pulse-like excursion (arrow). The peak of the pulse is out of bounds on the voltage scale in the figure. After the pulse, the voltage returns to a value below the resting potential u_{rest}.

R-STDP

In the previous section, we described the basic theory of STDP, which can be briefly summarized as follows: if two neurons fire their pulses at closer times, then the stronger the connection between these two neurons. Since pair-based STDP relies only on the relationship between pre- and postsynaptic firing times to update the weights, it is an unsupervised learning technique. The Reward-modulated Spike-Timing-Dependent Plasticity (R-STDP) algorithm, first proposed by Izhikevich in 2007[Izh07], is an example of the STDP algorithm, which is a variant of STDP algorithm and applies reinforcement learning ideas on the basis of STDP algorithm. The basic principle of R-STDP can be briefly summarized as: correct/incorrect decisions lead to STDP/Anti-STDP. That is, the reward/penalty signal generated by the behavioral outcome of the neural network is used to exert influence on the weight update of neurons[Moz+18]. The core technique of the R-STDP algorithm is a classic technique in the field of reinforcement learning: Eligibility Trace. The basic principle of Eligibility Trace can be simply summarized as follows: the most recent activity is considered more important.

However, in a reinforcement learning system, an agent performs certain actions in the environment that may lead to a reward. The overall goal is to learn how to achieve the maximum number of rewards. In biology, this analogy can be made with dopamine, whose function consists in rewarding when certain results are achieved (external learning signal). Thus, in an R-STDP signaled by dopamine, there is an eligibility to track STDP events, representing the activation of an enzyme that is important for plasticity[Izh07].

Equations 2.12 and 2.13 [Izh07] describe the behavior of eligibility tracking and synaptic

dynamics. c is the eligibility tracking, w is the synaptic strength, τ_c is the decay constant, and $s_{spre/post}$ is the spike time of the pre- or postsynaptic neuron. As defined in [Izh07], the eligibility track c decays in exponential time over a range of $\tau_c = 1s$. The change in synaptic weight w is generated by the extracellular concentration d of dopamine gating the eligibility trace c (specific to each synapse). Each time the reward signal is generated, the global concentration of dopamine shared by all synapses in the network increases, and it decays with an exponential time constant τ_d toward the baseline value defined by DA(t). Equation 2.14 shows this mechanism[Izh07].

$$\dot{c} = -\frac{c}{\tau_c} + W(\Delta t)\delta(t - s_{pre/post}) \tag{2.12}$$

$$\dot{\omega} = cd \tag{2.13}$$

$$\dot{d} = -d/\tau_d + DA(t) \tag{2.14}$$

The learning process can be stopped if the concentration of dopamine in the network is zero, as the synapse plasticity depends on the spiking activity of the neurons and the concentration of the dopamine in the neural structure[QPG22].

2.3 LiDAR Sensor

LiDAR, also known as optical radar (Light Detection and Ranging), is the abbreviation of laser detection and ranging system. It analyzes the reflected energy on the surface of the target object, amplitude, frequency and phase of the reflection spectrum and other information by measuring the propagation distance between the sensor transmitter and the target object, so as to present the precise three-dimensional structure information of the target.

Since the invention of lasers in the 1960s, LiDARs have been developed on a large scale. At present, LiDAR manufacturers mainly use laser transmitters with wavelengths of 905nm (nanometers) and 1550nm. Light with a wavelength of 1550nm is not easy to transmit in the liquid of the human eye, which means that the power of LiDAR using a laser with a wavelength of 1550nm can be quite high, without causing retinal damage. Higher power means longer detection distance, and longer wavelength means it is easier to penetrate dust haze. However, due to cost reasons, the production of LiDAR with a wavelength of 1550nm requires the use of expensive gallium arsenide materials. Manufacturers choose to use silicon materials to manufacture 905nm LiDARs close to the wavelength of visible light, and strictly limit the power of the transmitters to avoid permanent damage to the eyes.

LiDAR is mainly used in ranging, positioning and three-dimensional rendering of surface objects. As an important sensor, it is currently being widely used in the field of automatic driving and unmanned aerial vehicles.

Such as Velodyne's launch of the velodyne VPL-16 LiDAR, , which stacks 16 lasers vertically to make the entire unit rotate many times per second. There is a physical rotation in the transmitting system and the receiving system, that is, by continuously rotating the transmitter, turning the laser point into a line, and arranging multiple laser transmitters in the vertical direction to form a surface to achieve 3D scanning and receiving information.

In Coppeliasim (vrep), the velodyne VPL-16 LiDAR consists of four VisionHandle forms. In fact, each Handle corresponds to 90 degrees, and together it is the entire 360-degree space.

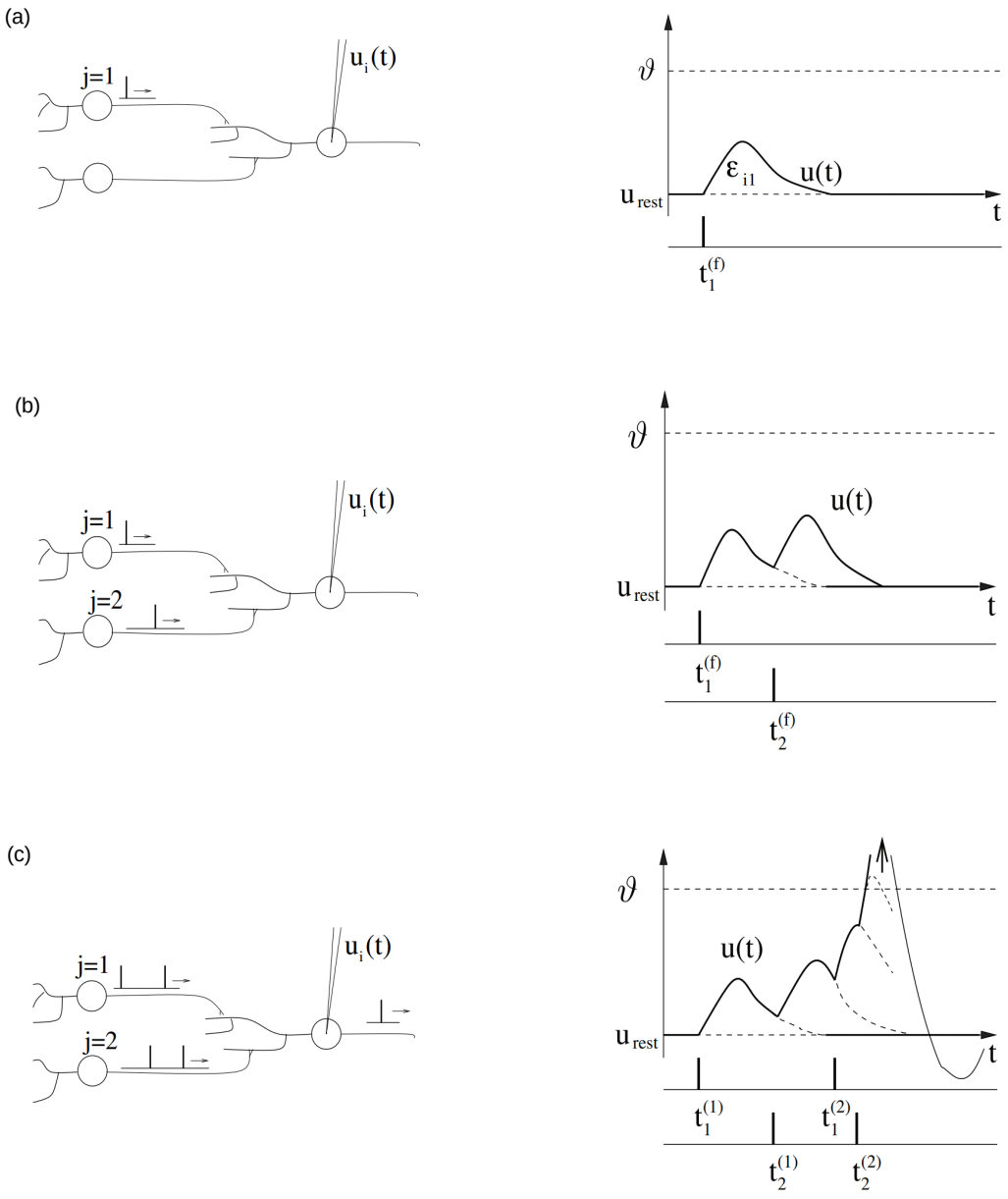

Figure 2.8: A postsynaptic neuron i receives input from two presynaptic neurons $j = 1, 2$ [Ger+14].

Chapter 3

Methodology of Lane Keeping Based on SNN

In this chapter, the core algorithms used in the paper are mainly introduced. The structure and parameters of the algorithms used in the thesis will be introduced in detail. Networks trained with these core algorithms could map some state inputs to control corresponding action outputs. When training successfully, the network should be able to perform simple tasks such as obstacle avoidance, target arrival, lane tracking.

In the first part of this chapter, the perception algorithm and the implementation of the results into state inputs are presented. In the second part, here is shown that SNN control based on Hebbian learning is implemented. In the last section, how reinforcement learning can be combined with SNN is presented.

3.1 Control Algorithms

In the previous section, simulation environment was presented, including a model of the Pioneer robot and the LiDAR device. It was also shown how to use the ROS theme to control the simulation for incoming motor speed and reset commands, and for outgoing LiDAR data and position information. In this section, several algorithms are discussed to learn to use ANN as well as SNN to control the robot for lane tracking tasks.

3.1.1 Control Algorithms 1: DQN

Main Task

(DQN) Learning how to follow the right lane in a classical reinforcement learning environment will be discussed in this section. Therefore, it is first explained how the task of following the right lane is transformed into a Markov decision process (MDP) in the deep Q-learning (DQN) algorithm, which is the main task of this thesis. Then, some details about the algorithm implementation are given.

A Markov decision process is usually defined as a five-tuple (Sec. 2.1) consisting of action, state, transition probability, reward, and discount factor. While transition probabilities can be ignored when using model-free reinforcement learning algorithms such as Q-learning, the other components of the MDP must be carefully chosen to ensure fast and stable learning.

1) Action

Figure 3.1 shows the three discrete actions the robot can choose from in this task. It can go straight, with both left and right motors running at the same speed. It can also turn to

the left or right by controlling the both motors to increase and decrease the corresponding speed, according to the desired direction. That is, when the left motor speed is greater than the right motor speed, the vehicle turns to the right; when the left motor speed is less than the right, the vehicle turns to the left. As mentioned earlier, the goal of the vehicle r obot is to move along the right lane using only the data from the LiDAR.

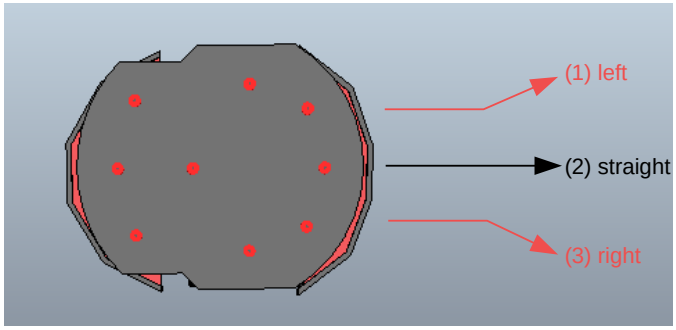

Figure 3.1: Three discrete actions in lane following task: (1) Turn left: Set the speed of the left engine to $v - v_t$ and right motor speed to $v + v_t$. (1) Go straight: Set left and right motor speed to v. (2) Turn right: Set left motor speed to $v + v_t$ and right motor speed to $v - v_t$.

2) State

Although in the previous section, the LiDAR data has been filtered to only the lane portion. However, due to the large size of the laser data, it could not be used as a direct state input to the MDP. First of all, the area in the upper middle of the lane can also be cropped without consideration due to its far distance from the vehicle. At the same time, the resolution of the lanes will be decreased in order to reduce the computational complexity of the task. This allows the network to make meaningful decisions more quickly. As shown in the right side of Figure 3.2, this is shown by taking the original 200×400 lane LiDAR map, that is, [-1m,1m] for left and right, [0,4m] for front and back, and then dividing it into small 40×20 (Varies by algorithm, here means in DQN) pixels and counting each point that appears in that pixel. The final result is shown in the lower left corner of Figure 3.2, and the lane alignment can be clearly seen. Although this is a significant reduction in dimensionality, it is still a challenging enough problem. To further improve the performance of the algorithm, the final DQN state input s_{MN} used in this thesis is a binary version of the state input i_{MN}, containing only 1 and 0. As shown in Equation 3.1, that is, when the state input is greater than 0, the final input to the DQN has a state value of 1. When the state input is 0, the final input to the DQN has a state value of 0.

$$s_{M,N} = \begin{cases} 0, & i_{m,n} = 0 \\ 1, & i_{m,n} > 0 \end{cases} \tag{3.1}$$

In the simulation, the LiDAR data is computed and published every 50 ms. On the other hand, the actions in the MDP are executed every 500 ms. Since the processing time of the LiDAR data is in the range of 100-300 ms, only the lastest LiDAR lane map is acquired each time. This will then be converted to the final state input.

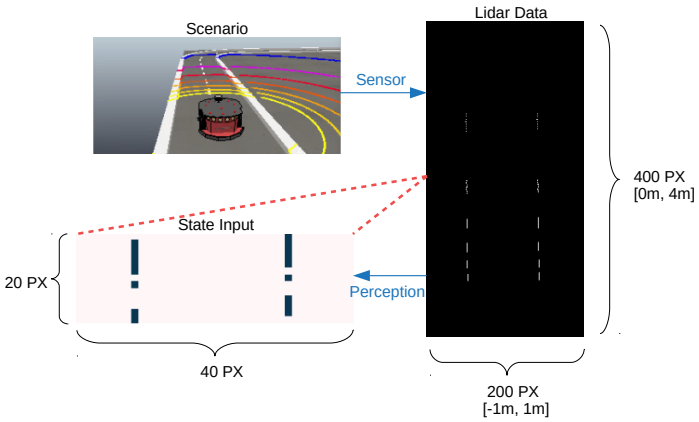

Figure 3.2: Conversion of LiDAR data into state input for reinforcement learning

3) Reward & Discount Factor

Reward plays a crucial role in reinforcement learning, which defines the goal of an agent. In this work, the robotic vehicle should learn to follow a lane as close to the center as possible. Figure 3.3 shows the reward at each time step of the MDP. It is defined as a Gaussian distribution function of the distance to the center of the lane. Since the model-free DQN algorithm learns from an empirical sample with one step ahead, it is beneficial to use rewards that are well distributed in the state space and monotonically growing toward the goal. This ensures that the robotic vehicle will learn to navigate in the direction of the target. even if it has not yet reached the direction of the goal. In addition to the LiDAR data, the simulator also publishes the position data of the robotic vehicle every 50 ms as well. Through a mathematical model of the lane center in both directions, this data is used to calculate the exact distance from the robotic vehicle to the lane center and the exact distance from the robotic vehicle to the lane center and the resulting reward. Similarly, this is done every 500 ms using the latest published position data The latest published position data. If the robot vehicle reaches a position where its distance to the center of the lane is greater than 0.15 m, the training event is terminated and a reset message is sent, causing the simulator to place the robot vehicle at the starting position in the opposite lane. Allowing the robot vehicle to alternate between the two lane directions increases the experienced state of the robot vehicle and leads to a more general strategy after learning. In reinforcement learning, the extent to which an agent takes expected future rewards into account is typically controlled by a discount factor. Although the lane tracking task does not necessarily require many steps forward, the discount factor is set to 0.99, so it is possible to solve tasks that involve more foresight.

Algorithm Process

In Figure 3.4, the communication of the DQN controller modules is shown. All modules are running on Ubuntu 18.04, and the controller, environment handler, and experience buffer are written in Python 2.7. The environment handler manages all communication between the DQN controller and the CoppeliaSim simulation discussed in Section 3.1. When the controller is implemented at the Markov Decision Process(MDP) level, the environment handler transitions between ROS messages and status, actions, and rewards. That is, messages are exchanged through topic publishers and subscribers. Discrete MDP actions are translated into left and right motor speeds and published in the respective topics (Figure 3.5). Also at the end of each training period, the environment handler publishes a Boolean message to reset

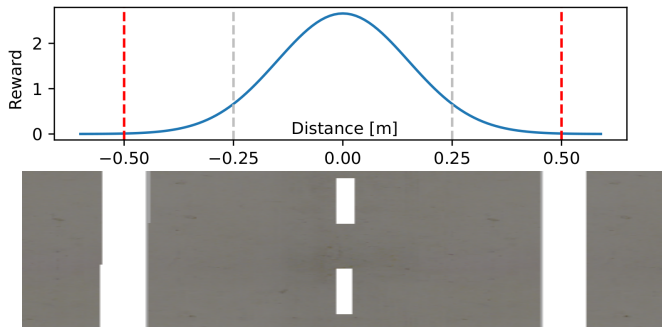

Figure 3.3: Reward for the lane following task

the robot and place it in the initial position in the corresponding direction. In addition, since the Pioneer P3-DX Robot is prone to nose up after a period of operation, which affects the recognition area of the LiDAR. Therefore, a CoppeliaSim software restart is performed every 30 periods in the code.

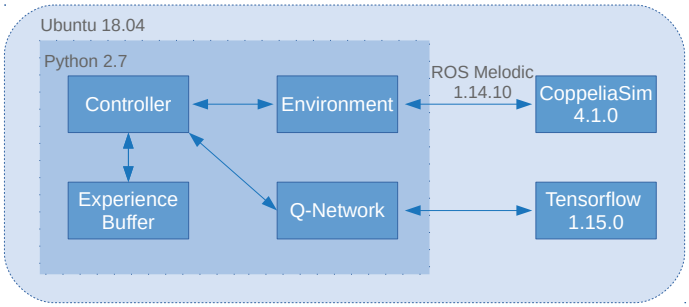

Figure 3.4: DQN Controller

In addition, it subscribes to the LiDAR data and defines a callback function with the data stored. When the environment is called to perform a discrete action, the stored LiDAR data is converted into a binary state image by road edge recognition and fed back to the controller. At the same time, the environment subscribes to the robot's position data. With the predefined model in the code, the distance between the robot vehicle and the center of the lane can be calculated, and the resulting reward is shown in Figure 3.7 and fed back to the controller.

One of the main improvements of DQN over earlier algorithms is the addition of an experience buffer, which stores the transformations of the MDP and samples random experiences for the stochastic gradient descent step. Firstly, this makes the data uncorrelated and prevents the algorithm from falling into local minima. Secondly, by fixing the size of the buffer and dropping the old experience samples, it alleviates the conflict between early stochastic exploration and later knowledge exploitation.

The neural network used by the DQN controller is implemented by Tensorflow[Rao+17], an open source machine learning software library developed by Google.Tensorflow provides a Python interface for advanced processing of neural networks, while the core functionality of the computation is developed by Google. advanced processing of neural networks, while the core functions are computed using an efficient kernel implementation. In addition, combined with CUDA[Gar+08], a parallel computing platform developed by Nvidia, it provides

a simple tool for fast GPU computation. Convolutional neural networks (CNNs) have become the network architecture of choice for image data in many previous classification tasks as well as control tasks using DQNs. This is due to the fact that CNNs exploit the spatial information of the input data, which can lead to considerable performance gains in many tasks.

In this paper, we chose a fully connected feedforward network structure using rectified linear units (ReLU) as activation. The network takes a binary state image as input and generates $32 \times 16 = 512$ input neurons. It consists of 2 hidden layers with 200 neurons each and 3 output neurons representing discrete actions. The training was performed using the stochastic optimization algorithm Adam[KB14].

The following part describes the detailed process of the DQN algorithm and the communication with external components. At the beginning, the action network is initialized with random weights and copied to the target network. Each cycle of the training procedure will start with the robot reset to its starting position, changing lanes each cycle. Thus, the initial state input is a vector of zeros. At each time step, the actions are chosen according to a "greedy" policy. This means that the probability of e is [0; 1] and the agent will choose an action at random. Otherwise, it will choose the action with the highest action value. At the beginning, it's set to 1 to ensure purely exploratory actions. After a predefined number of time steps, it is then linearly reduced to a final value close to zero. The selected action is sent to the environmental handler, which communicates with the simulator to obtain the reward, the next state image and the distance to the center of the lane. In addition, each transition is stored in an experience buffer. Every N steps, the actual training step is performed by randomly drawing transitions from the experience buffer. The target network is used to calculate the update target and then a loss function is constructed in order to perform a stochastic gradient descent step on the action network. At the end of each training step, the weights of the target network are slowly updated to the weights of the action network, $\tau \in [0, 1]$ while $\tau \ll 1$. The training ends if the robot exceeds the maximum distance to the center of the lane, or if it reaches the maximum number of steps in a period. The latter mechanism guarantees that the robot will experience both directions of the road, even if it has learned a good strategy along the lane. The whole training process ends after a predefined number of sets or total training steps.

3.1.2 Control Algorithms 2: SNN

The DQN algorithm introduced in the previous section processes the road edge data by storing LiDAR data and processing it at each step of the MDP. It is evident from the longer training process that this approach is not very efficient in learning and does not take full advantage of the high resolution of LiDAR. In addition, the computation of traditional ANNs is generally more expensive, especially with 512 neurons as input. In this way it also increases the delay of the control system and cannot respond quickly during the actual driving process. However, spiking neural network can solve these concerns very well. The lower computational cost allows the network to process the LiDAR data fast enough to keep pace with the LiDAR refresh frequency. Spiking neural network is able to take into account the exact timing of spikes rather than just the average rate. At the same time, brain-like neural-inspired design can be used as an innovative idea in software, which also lays the foundation for developing more efficient hardware in the future. Therefore, ideally, we should feed the LiDAR data directly into the SNN and use some reinforcement learning algorithms to train the corresponding weights. The output of the SNN is then used to control the speed of the vehicle engine. However, in practice, the output of SNN is prone to jumping. And the speed change of the vehicle driving on the lane is a slow process, such that the SNN output is not suitable for directly controlling the engine of the robotic vehicle. A suitable vehicle steering wheel model

is needed at this point. In further research we found that Kaiser et al[Kai+16] proposed an agonist-antagonist muscle system based steering wheel model for the lane following task that is more suitable for vehicle control. In this section, different algorithms for sensing LiDAR data are first discussed. Then it explores the implementation of a steering wheel model for vehicle controllers. Finally, it shows how this approach can be combined with SNN to enable it to learn how to drive along a lane according to R-STDP rules instead of setting the weights manually.

Thresholding Algorithms

In reinforcement learning, lanes need to be used as STATE input to determine the left and right steering angles of the car while driving. Since LiDAR detects the information of all objects around the vehicle in the form of x-y-z coordinate points, this kind of lane information cannot be directly obtained from 3D LiDAR. Therefore, the LiDAR data needs to be processed to get a clear lane edge for describing the lane, then to be used as state input for reinforcement learning. Three different methods will be described to detect the lane edge below according to the degree of simplicity to complexity.

1) Height Thresholding

Since the height of the lane edge is fixed, based on a fixed z-axis height we can determine the lane edge points between the LiDAR detection points. In other words, reading the x-y coordinates corresponding to a particular value of height z is the lane edge point we need.

In V-REP, since the LiDAR is placed higher than the vehicle, we take z between -0.25m and -0.23m. The information thus obtained is shown below.

2) Gradient Thresholding

The core idea is to find the edge of the vehicle by gradient descent. The specific idea is to a) divide the road scanned in front of the vehicle into a certain number of grids, b) find the maximum height and the minimum height (that is, the z-axis) in each grid, c) calculate the height difference, and determine whether this grid contains the lane edge. When the height difference value is greater than a certain threshold value, then it can be determined that the lane edge exists in this grid.

3) FCN Recognition

After having the previous method, the lane areas can also be recognized by neural network training. We use supervised learning.After selecting a network model suitable for road recognition, the network is trained by inputting a LiDAR map and a training set with corresponding road labels. The trained parameters are used for real-time output of road areas. This not only ensures a certain accuracy in vehicle driving, but also improves the speed of lane recognition. The detailed steps are described below.

a) Data Processing

Since the data acquired directly from the radar are in 3D, which is not conducive to subsequent data processing, 3D data conversion to 2D data is required.

Firstly, the radar point cloud is filtered for points in the forward direction [0, 4m] and left and right [-1m, 1m], so that only lane related points can be retained by removing surrounding objects. This step is also needed in Height Thresholding and Gradient Thresholding, so it will not be repeated above.

Secondly the point cloud position values are converted to x,y pixel position values based on the resolution and then the pixels are shifted so that the minimum value is (0,0). Clip the height value between the minimum and maximum height, here [-0.3m, -0.2m] and rescale the size of the original height value to expand to [0, 255]. Now initialize an empty image array, and fill the pixel values in the image array with the x,y pixels we just got. Here the default background is black and the pixels to be filled are set to white. Finally, the numpy array is converted to a pil image and saved. At last the 3D data is successfully converted to 2D image as shown in Figure 3.5.

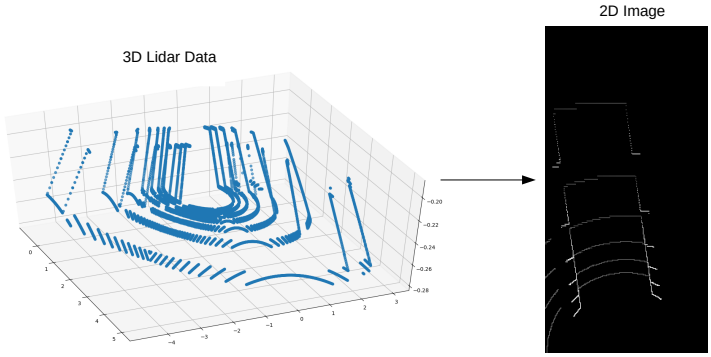

Figure 3.5: 3D data conversion to 2D image

b) Semantic Segmentation - FCN8 based on VGG16

From the image obtained above, we can see that although the edge of the road can be clearly seen, it still needs to be further processed to obtain the feasible driving area and finally the edge of the road.The acquisition of the travel area then requires a suitable semantic segmentation network to handle. The acquisition of the travel area then requires a suitable semantic segmentation network to handle. After searching, it was found that the mainstream ones are: Fully Convolution Networks (FCNs)[LSD15], SegNet[BKC17], U-Net[RFB15] and so on. Considering the accuracy of segmentation results and inference time, FCNs are finally chosen. FCNs improve the classification network Visual Geometry Group 16 (VGG16) by replacing the fully-connected layer with a convolutional layer to train the end-to-end semantic segmentation network. Where, VGG16 is due to containing 13 convolutional layers and 3 fully connected layers, totaling 16 weight layers, because both convolutional and fully connected layers have weight coefficients. The pooling layers don't involve weights, therefore are not part of the weight layers and not counted. Among the FCNs, FCN8 has a higher accuracy, so I finally decided to use FCN8 as the model framework for training.

As shown in Figure 3.6, conv1 to pool5 use the VGG16 network structure to shrink the image by 32 times, and FCN replaces the fully connected layer [class * (h/32) * (w/32)] with the convolutional layer [4096 * (h/32) * (w/32)] to complete the classification of each pixel. FCN32 upsamples the convolutional result directly by 32 times to recover the original size (class * h * w). FCN16 upsamples the convolution result and fuses the pool4 result, and then does 16x upsampling to get the original size. FCN8 upsamples the result of FCN16, fuses it with the result of pool3, and upsamples it by a factor of 8 to obtain the original size of the feature map. At this point, the feature map is restored to the original size, thus completing the semantic segmentation task.

The training process of the model is described below. Since FCN8 is a supervised learning neural network. Therefore, the training set and the corresponding groundtruth are needed to train the network. At first, the radar map in the Kitti dataset is used and cropped. 279 images like the two left ones in Figure 3.7 are selected and input to the FCN8 network for

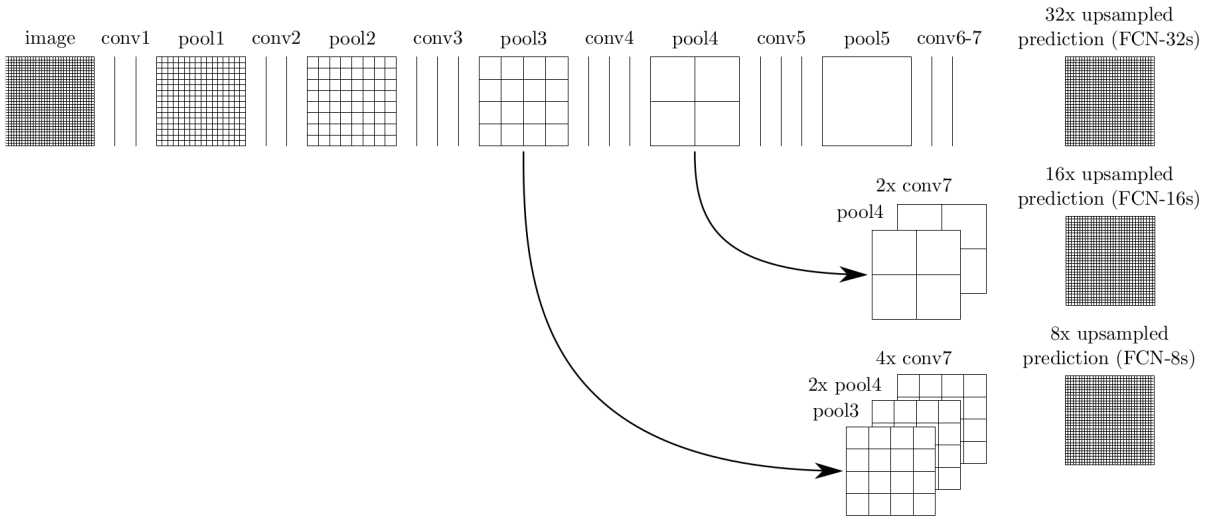

Figure 3.6: FCN8 network structure

training to get an initial model[Cal+17]. The plot on the right of Figure 3.7 represents the test results of the model. The green color indicates the traveling area, and it can be seen that the scene segmentation has a high correct rate.

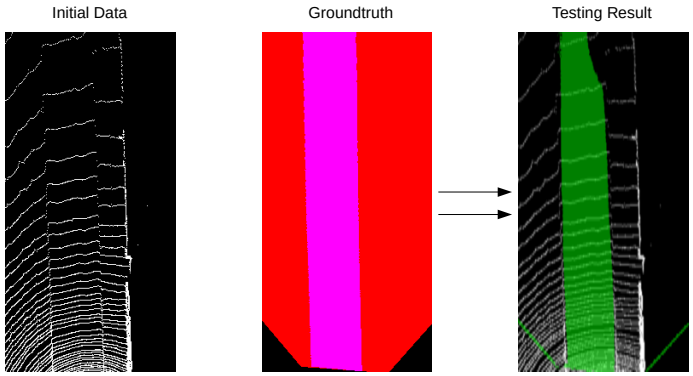

Figure 3.7: First training based on Kitti

In order to fit the scene in Coppeliasim, the corresponding groundtruth is drawn and cropped by reading the coordinates and angles of the vehicle while it is driving. As shown in Figure 3.8, the picture on the left is the groundtruth of the whole scene, reading the vehicle coordinates and the four elements of xyzw in ROS at a certain moment, converting the four elements into Euler angles-roll,pitch,yaw. Based on the resolution, the groundtruth of the whole scene is properly cropped according to the position and angle (forward direction [0,4m], left and right direction [-1m, 1m]). Finally, the two maps in the middle of Figure 3.8 are obtained. Here, about 300 lidar images and the corresponding groundtruths are selected for training to obtain the final trained model.

The output results obtained by inputting to the final trained network model are shown in the right image of Figure 3.8. It can be seen that the semantic segmentation results are also ideal, and the green area is the driving area. It is worth noting that the test images selected here do not exist in the actual training set, which also ensures the accuracy of the test results.

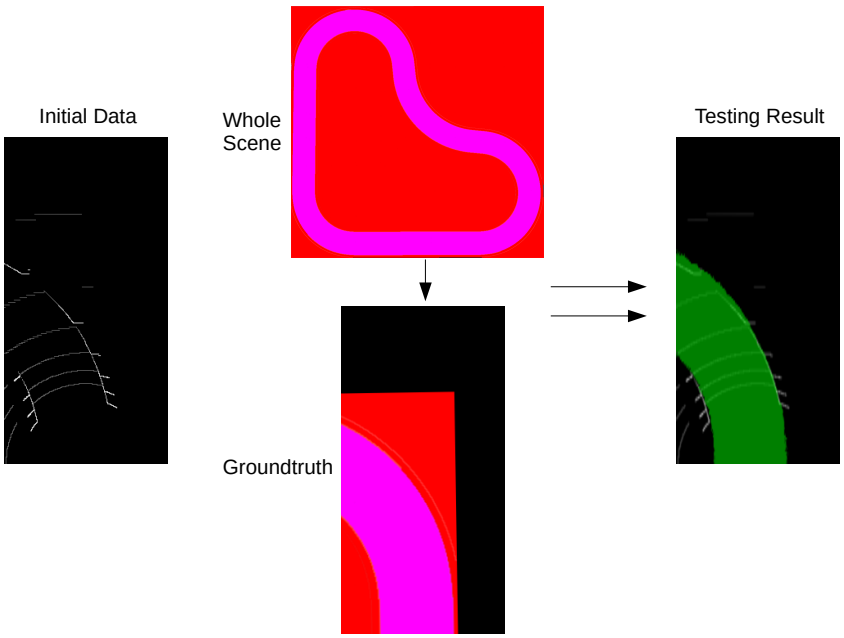

Figure 3.8: Second training based on Coppeliasim scene

c) Edge Extraction

There are two methods of edge extraction: Grid and OpenCV. Grid method is by dividing the whole image into some rows according to the resolution. At this point, the part of the rows away from the vehicle head can be discarded according to the situation.Each row is traversed and the maximum and minimum y are read out, which also correspond to the pixel value coordinates of the lane edges. Finally filling into a new matrix, the final lane edge is obtained, and this is also used as input state for subsequent vehicle training.

OpenCV uses the Laplance operator for edge extraction, which is based on the mathematical principle that the value of the second-order derivative is zero at the point of maximum change, i.e., the edge is zero. Calculating the second-order derivative of an image can be used for gradient calculation, edge extraction, and edge detection.

In the edge part, the pixel values show "jumps" or large changes. This feature can be used as a method to detect the edges of an image because the first-order derivative of the edge will show the extreme value, and the second-order derivative of the edge will show the extreme position of the first-order derivative and the second-order derivative will be zero. However, the zero values of the second-order derivatives are not only found at the edges (they can also be found at meaningless locations), but these points can be filtered out.

The Laplacian operator is defined by the following Equation 3.2. Since the image is 2-dimensional, we need to derive it in both directions. This can be done automatically by using the Laplacian operator directly, so there is no need to calculate the two directions separately.

$$Laplace(f) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{3.2}$$

Laplacian extract edge process: a. Gaussian blur - GaussianBlur(); b. Convert grayscale - cvtColor(); c. Laplacian - second order derivative calculation Laplacian(); d. Take absolute value - convertScaleAbs() - here we can get the edge image; e. Binarization thresholding - threshold() - enhance edge features, edge images are more visible.

The whole FCN8 processing process is shown in Figure 3.9. It is worth noting that only

the lower half of the image is finally intercepted as the state input, because this can avoid the false recognition of the upper part and thus bring disturbing factors to the training.

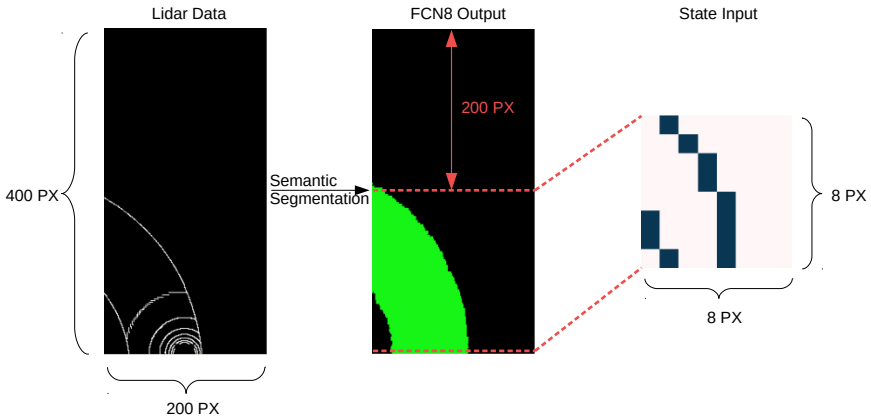

Figure 3.9: FCN8 process

R-STDP

The general idea is that neural activity, stimulated by either the left road edge or the right road edge from the LiDAR data, will cause the left or right neuron to strengthen (and the other side to weaken subsequently). At this point the robot moves away from the direction of neural activity, which will cause the left or right motor to increase its speed. Figure 3.10 shows the R-STDP control structure. The LiDAR data presented in the previous section is subjected to three different perception methods to obtain lane edges, which are scaled and used for the excitation of Poisson neurons. This time a single SNN network with $8 \times 8 = 64$ input neurons (minimum can be $4 \times 4 = 16$) is used. The input layer was connected to two LIF output neurons in an "all-to-all" fashion using R-STDP synapses. The number of output spikes n_t^{left} and n_t^{right} is calculated for each motor neuron in each simulation step and used as input to the steering wheel model introduced in the previous section. The network for this work draws on the basic network structure of the Braitenberg vehicle controller proposed by Kaiser et al[Kai+16]. The difference is that in this paper two static synapses are merged into one so that the other previously existing synaptic connection can be interpreted as a zero-weighted connection. And the weights are automatically updated during training by walking the lane according to the R-STDP rules, without having to be set manually.

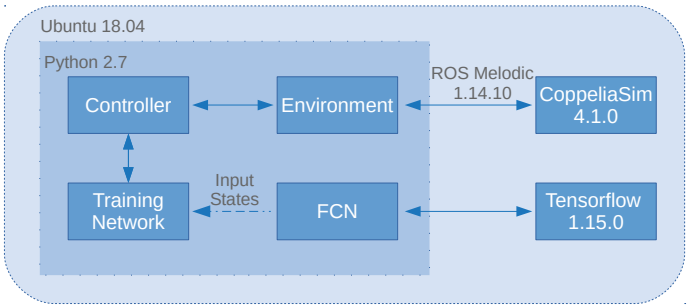

Figure 3.10: R-STDP controller

The network is implemented in NEST using the simulator's own stdp dopamine synapse model for all connections. Based on studies by Izhikevich[Izh07] and Potjans et al[PMD10], a low-pass filtered version of the user-specific neuronal pool constitutes the dopaminergic signal that modulated spike timing-dependent plasticity[Flo07]. To reduce the complexity of the control task, the reward signal in this work is set directly at each simulated time step, rather than indirectly by first stimulating the pool of dopaminergic neurons. While this approach may not be biologically sound, it simplifies the control task without changing the underlying dynamics of the problem, which makes it a good starting point. Figure 3.11 shows the reward signal given at each simulation time step. It is a linear signal of opposite sign defined for each motor based on the distance of the robot to the center of the lane. When the robot is to the right from the center of the lane and should turn left to the center, the connection leading to the ignition of the right motor neuron is strengthened and the connection leading to the ignition of the left motor neuron is weakened. On the other side of the center of the lane, this process is reversed. Over time, the robotic car can learn to associate certain input stimuli with left or right turns and act accordingly. These considerations lead to the following rewards for left and right motor neuron connections, where d is the distance to the center of the lane and c_r is a constant for the scaling reward:

$$r_{left/right} = -/+(d \cdot c_r) \tag{3.3}$$

The reward is defined separately as a linear function of the lane center distance, scaled by a constant c_r. The lane marker is 0.25 m away from the lane center. If the robot will leave the center of the lane more than 0.25 m, the period will automatically end and the robot bot will be positioned at its starting position to start a new period.

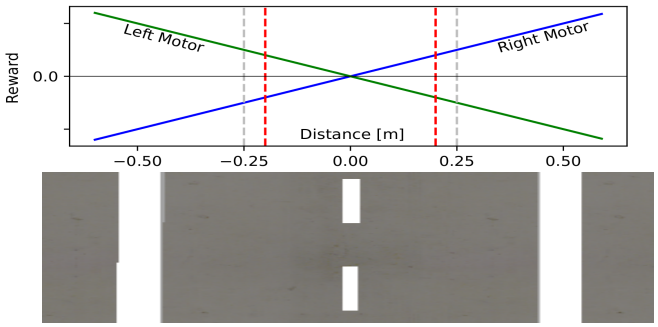

Figure 3.11: Reward given by the R-STDP controller

Steering wheel model

In order to make the robot car drive in the lane with smoother speed changes. In this work a steering wheel model based on the agonist-antagonist muscle system proposed by Kaiser et al[Kai+16] was used to control the speed of the car. A slight difference, however, is that instead of the steering angle, the turning speed is calculated and the left and right motors are added and subtracted (Figure 3.1). First, the output spike count is scaled by the maximum possible output:

$$m_t^{left/right} = \frac{m_t^{left/right}}{n_{max}} \in [0,1],\ with\ n_{max} = \frac{T_{sim}}{T_{refrac}} \tag{3.4}$$

where T_{sim} sim denotes the simulation time step and T_{refrac} describes the length of the refractory period of the LIF neuron. Based on the difference between the normalized activities m_t^{left} and m_t^{right} and the turning constant c_turn, the turning speed is defined as

$$S_t = c_{turn} \cdot a_t, \; with \; a_t = m_t^{left} - m_t^{right} \in [-1, 1] \tag{3.5}$$

Furthermore, in order to ensure slower speed in turns, the overall speed is controlled according to

$$V_t = -|a_t| \cdot (v_{max} - v_{min}) + v_{max} \tag{3.6}$$

where v_{min} and v_{max} are predefined speed limits. Since controlling a car is generally a continuous process, overall and turn speed are smoothed based on the activities:

$$v_t = c \cdot V_t + (1 - c) + v_{t-1} \tag{3.7}$$

$$s_t = c \cdot S_t + (1 - c) + s_{t-1} \tag{3.8}$$

$$with \; c = \sqrt{\frac{(m_t^{left})^2 + (m_t^{right})^2}{2}} \tag{3.9}$$

Finally, the control signals for the left and right motor were computed by

$$v_t^{left} = v_t + s_t \tag{3.10}$$

$$v_t^{right} = v_t - s_t \tag{3.11}$$

Then the turn speeds v_t^{left} and v_t^{right} obtained above in the environment processor are published to the simulator CoppeliaSim, which enables continuous control of the robot vehicle orientation.

Chapter 4

Experimental Process

In this section, the simulation environment is presented, including models of the Pioneer robot and the LiDAR device. It was also shown how to use the ROS theme to control the simulated incoming motor speed and reset commands, as well as the outgoing LiDAR data and position information.

4.1 Simulation Environment

The robot simulator CoppeliaSim (formerly V-REP) with integrated development environment is based on a distributed control architecture: each object/model can be controlled individually via embedded scripts, plug-ins, ROS or BlueZero nodes, remote API clients or custom solutions. This makes CoppeliaSim very versatile and ideal for multi-robot applications. The controller can be written in C/C++, Python, java, Lua, MATLAB. CoppeliaSim is used for rapid algorithm development, factory automation simulation, rapid prototyping and verification, robotics-related education, remote monitoring, safety double-checking, digital twins, and more. CoppeliaSim is controlled using the Lua language[Ier06]. The language is small, fast in execution and high in performance. Lua is not restricted to programming paradigms, that means it's a multi-programming paradigm programming language. Lua provides only a small collection of features to meet the needs of different programming paradigms, rather than providing elaborate feature support for a specific programming paradigm. By providing these basic meta-features Lua language can be freely adapted as needed. Compared to Gazebo, CoppeliaSim is a more intuitive and user-friendly simulator, and includes more features. In addition, CoppeliaSim requires less hardware than Gazebo[Luc14]. At the same time, the educational version of CoppeliaSim is free for university students and all features are available in it. Therefore, CoppeliaSim was finally chosen as the simulator for this thesis.

Since real robots usually implement more than a single function, this means that many different software and hardware will be integrated into the same system. Then a standard interface and functionality becomes especially important in robot simulation. Currently the most widely used is the Robot Operating System (ROS)[Qui+09]. It is an open source meta-level operating system (post-OS) that provides services similar to an operating system, including hardware abstraction description, underlying driver management, execution of common functions, inter-program messaging, and program distribution package management. CoppeliaSim also provides a ROS interface plugin that can be used to create ROS nodes for publishing and subscribing to topics. In this thesis, the communication between the CoppeliaSim environment and the controller is done through ROS messages, which are defined in non-threaded subscripts attached to the robot model, written in the Lua language mentioned earlier. As for the controller, it is written in python language. In Table 4.1 the

simulation setup of CoppeliaSim used in this thesis is shown.

`Dynamics engine:`	Bullet V2.78
`Dynamics settings:`	Accurate
`Simulation time step:`	50 ms

Table 4.1: CoppeliaSim simulation settings

4.1.1 Pioneer P3-DX Robot

As Shown in Figure 4.1, Pioneer P3-DX Robots is the world's most popular intelligent mobile robot for education and research. Its versatility, reliability and durability make it the platform of choice for advanced intelligent robots. Pioneer robots are pre-assembled, customizable, upgradeable, and rugged enough for years of use in labs and classrooms.The Pioneer P3-DX Robot features a differential drive two-wheeled robot with two independent DC motors controlling the movement and orientation of the Pioneer P3-DX. It also has a caster wheel to support the Pioneer Robot's chassis. The robot's length, width and height are 45.5 cm, 39 cm and 23.7 cm, respectively. The Pioneer P3-DX wheeled robot is equipped with sixteen ultrasonic sensors. These sensors read obstacles from approximately 2 cm to 4 m[Pan+20]. Through remote API functions, we can also extract real-time sensor data, the x-axis and y-axis positions of the Pioneer P3-DX Robot, etc.

Figure 4.1: Pioneer P3-DX two-wheeled robot

Attached to the robot model in CoppeliaSim is a subscript that handles the communication between the simulator and the controller. In order to control the robot, the following subscribers are defined in the script that executes the callback function. For motor control, the "/leftMotorSpeed" and "/rightMotorSpeed" messages are set directly to the target speeds of the two motors. In addition, the Boolean messages posted under the "/resetRobot" topic will place the robot in a different starting position depending on the value. This is important for the robot to be able to travel correctly in two different directions. We also need to determine the size of the reward based on the position of the vehicle relative to the lane. Then the "/transformData" topic is needed to determine the exact position and direction of the robot vehicle. By calculating the relative position, the position of the vehicle with respect to the lane is obtained. This allows the reinforcement learning environment to interact with the simulator, thus allowing the training to be complete.

4.1.2 LiDAR Sensor

Velodyne-16 The distance values resolved by the LiDAR at each angular resolution in the scanning area are connected in sequence so that the outline of the surrounding objects can be seen very visually through the polar representation. A diagram of the lid scanning range and performance metrics can be found in Figure 4.2.

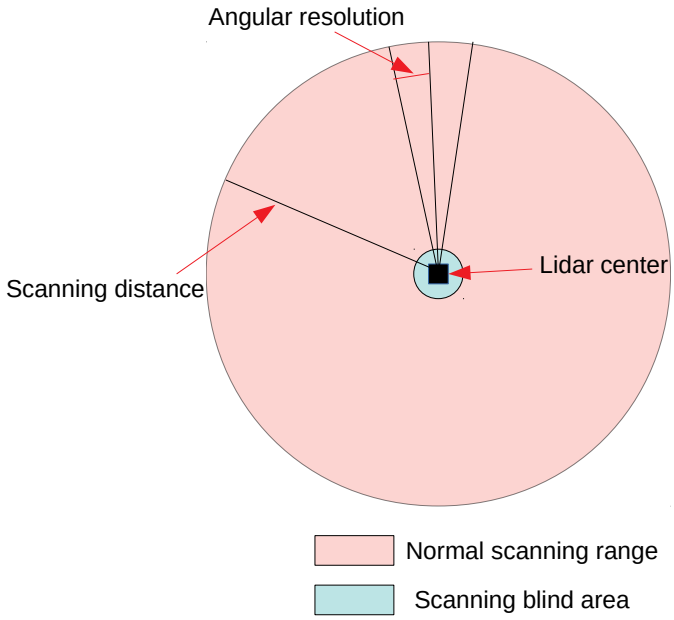

Figure 4.2: LiDAR scanning range

Four general performance specifications exist for LiDAR: range resolution, scanning frequency (or sometimes scanning period), angular resolution, and scanning range. The range resolution measures the accuracy of the range at a given distance, and usually differs from the true value by 5-20mm, which is usually the default value without adjustment; the scan frequency specification how fast the LiDAR completes a complete scan, usually at 5Hz and above; the angular resolution directly determines how many sample points the LiDAR can return in a complete scan; the visual range refers to the complete scan of the LiDAR. The visual range refers to the wide angle of the complete LiDAR scan, outside the visual range is the blind area.

There are many LiDAR sensors to choose from in CoppeliaSim, such as Hokuyo URG, Sick S300, Velodyne and so on. However, since most of them are 2D LiDAR sensors, we only choose between two 3D LiDAR sensors, Velodyne HDL-64E S2 and Velodyne VLP-16. For computer performance reasons, the Velodyne VLP-16 was finally chosen to conduct the experiment in order to ensure a smooth flow. velodyne VLP-16 LiDAR maintains Velodyne's groundbreaking and important features in LiDAR: real-time transceiver data, full 360-degree coverage, 3D distance measurement, and calibrated reflectometry.

The velodyne VPL-16 model LiDAR in CoppeliaSim is composed of four VisionHandles, in fact each Handle corresponds to 90 degrees of the coordinate system and combined together is the entire 360 degree space. According to the above LiDAR performance specifications, the specific parameters of the velodyne VPL-16 LiDAR are shown in Table 4.2. The data of the LiDAR intercepted in the experiment are, forward direction (vertical): [0,4]; left-right direction (horizontal): [-1,1]. Therefore the maximum scanning range of 5 meters is

sufficient. The LiDAR has a fixed scanning angle range in the vertical direction, $[-30°, 30°]$. The Velodyne VPL-16 LiDAR installed on the Pioneer P3-DX robot is shown in Figure 4.3. The final intercepted LiDAR map is shown below in Figure 4.4. The actual number of point clouds according to the above parameters is around 16000.

Scanning Frequency:	5 Hz
Angular Resolution(Vertical):	2
Scanning Range:	5 m

Table 4.2: LiDAR settings

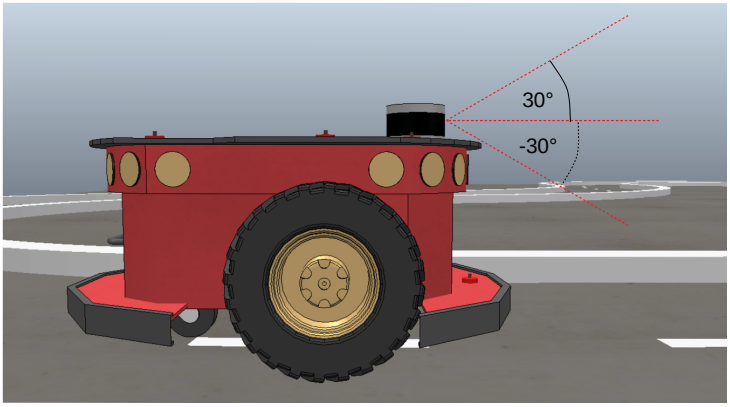

Figure 4.3: Pioneer P3-DX robot attached with LiDAR

4.2 Lane Scenarios

4.2.1 Scenario 1

Scenario 1 Figure 4.5 is shown below. The scenario consists of a circular route with two lanes. The lanes consist of two solid lines and a uniform dashed line in the middle. This constitutes a counterclockwise outer lane and a clockwise inner lane. Each side lane can be divided into 6 segments. From the starting position, the outer lane goes through (A) straight, (B) left turn, (C) straight, (D) left turn, (E) right turn, and (F) left turn, respectively. Similarly, the inner lane went through (C) straight section, (B) right turn, (A) straight section, (F) right turn, (E) left turn, and (D) right turn. During training, the robotic vehicle will switch between the inner and outer lanes at each reset. Thus, it will experience left and right turns equally. In addition, there are different radius curves in the same side lane. Therefore it will also experience turns of different radii.

4.2.2 Scenario 2

Figure 4.6 shows the Eight-Shape of Scenario 2. To further verify the robustness of the algorithm, we augmented another type of map, the figure-of-eight map. In this map, this scenario can effectively verify how the robot vehicle handles when facing continuous curves as well as intersections.The scenario consists of two 3/4 circles and two straight intersections.

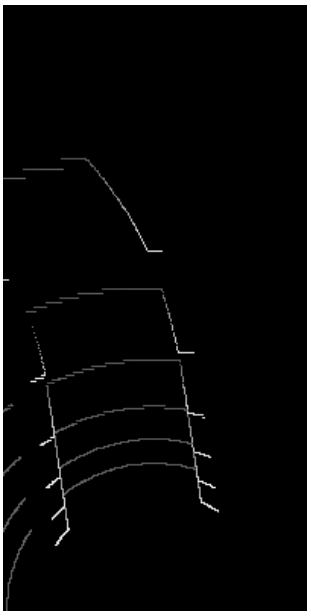

Figure 4.4: Intercepted Velodyne VPL-16 LiDAR map

In this design, we specify that the lanes are single lanes in a fixed direction with five segments. From the starting position, the lanes go through (A) counterclockwise -1/4 circle, (B) straight ahead, (C) clockwise -3/4 circle, (D) straight ahead and (E) counterclockwise -3/4 circle. In this map, the radius of the turns is consistent. The robot vehicle needs to judge whether to go straight or turn at the intersection.

4.2.3 Scenario 3

Figure 4.7 shows the Square-Shape of Scenario 3. The scenario consists of four 1/4 circles and four straight lane routes. The lanes consist of two solid lines and a unified dotted line in the middle. This constitutes a counterclockwise outside lane and a clockwise inside lane. Each side lane can be divided into 8 segments. From the starting position, the outside lane goes through (A) straight ahead, (B) left turn, (C) straight ahead, (D) left turn, (E) straight ahead, (F) left turn, (G) straight ahead, and (H) left turn. Similarly, the inside lane goes through (C) straight, (B) right, (A) straight, (H) right, (G) straight, (F) right, (E) straight, and (D) right. During the training period, the robot vehicle will switch between the inside and outside lanes at each reset. Therefore, it will experience the same left and right turns. It will experience the same turn radius for all turns.

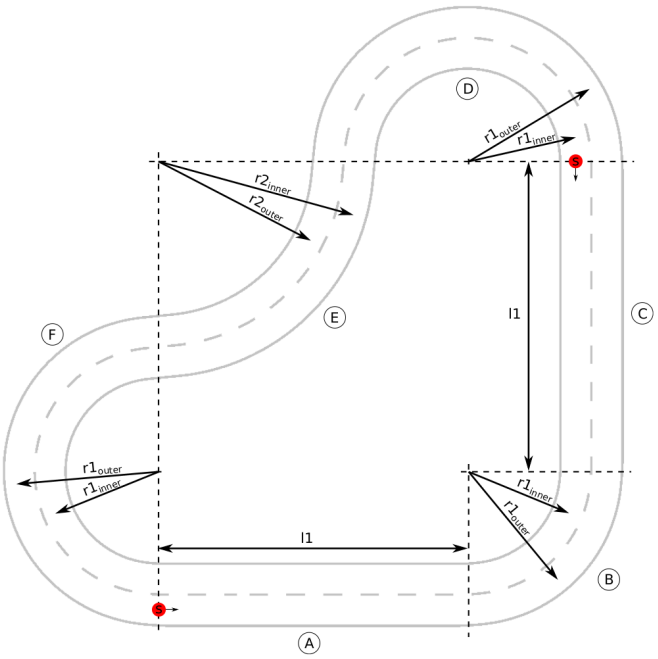

Figure 4.5: CoppeliaSim Scenario 1: Standard shape. Dimensions: $r1_{inner}$=1.75m, $r2_{inner}$=3.25m, $r1_{outer}$=2.25m, $r2_{outer}$=2.75m, $l1$=5.0m

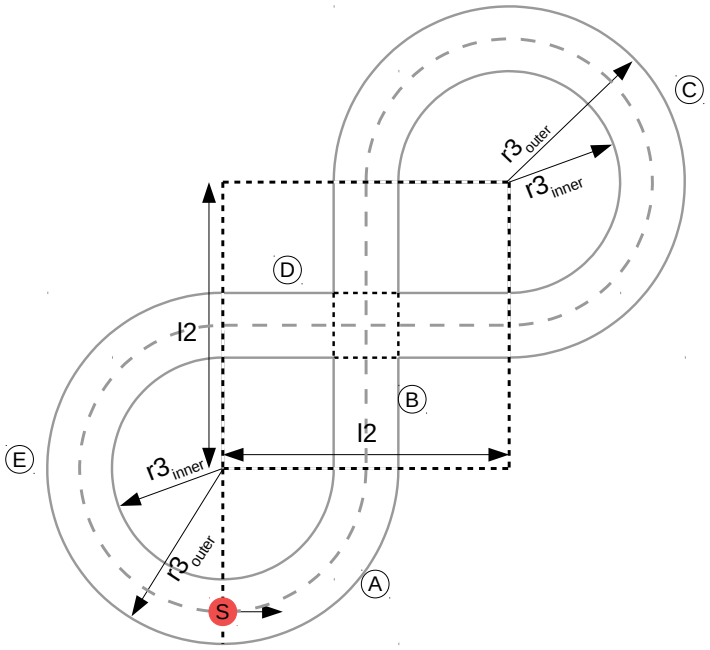

Figure 4.6: CoppeliaSim Scenario 2: Eight Shape. Dimensions: $r3_{inner}$=1.5m, $r3_{outer}$=2.5m, $l2$=4.0m

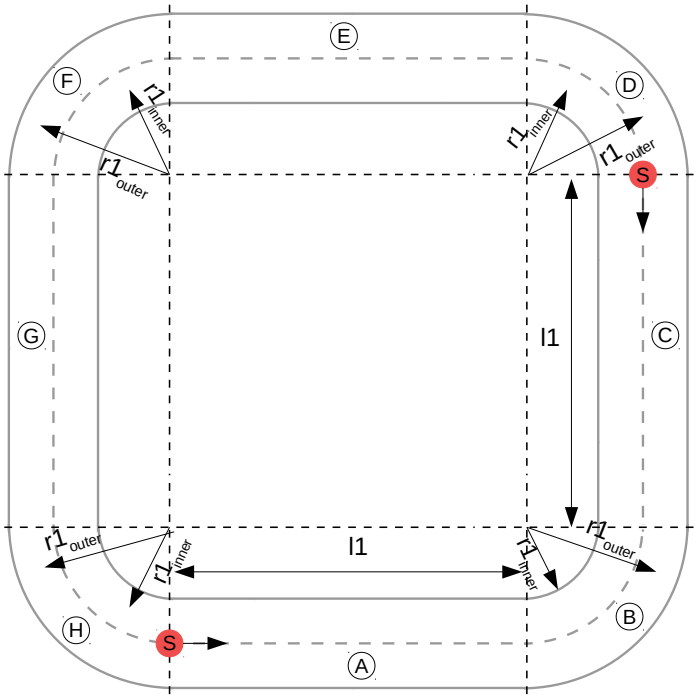

Figure 4.7: CoppeliaSim Scenario 3: Square Shape. Dimensions: $r1_{inner}$=1.75m, $r1_{outer}$=2.25m, $l1$=5.0m

Chapter 5

Results and Analysis

5.1 Controller 1: DQN

The training progress of the DQN algorithm in Scenario 1 is shown in Figure 5.1. The training parameters are given in Table A.1. Since it is used as a comparison experimental group, this controller has only been applied in scenario 1. At the beginning, the robot randomly chooses actions. Once these random actions cause the robot to exceed the lane center distance threshold of 0.25 m, the set is terminated. Thus, the steps and rewards at the beginning are randomly distributed at a low level. Although the "greedy" strategy continuously increased the chance of selecting the action with the highest action value, the robot did not show any learning effect until the 1000th episode. At around episode 500, the steps and rewards are actually decreasing, because the policies followed by the robot are not yet optimal. After 2000 steps, the cumulative reward for 1000 steps or more is still slowly increasing, approaching the reward maximum. After about 2050 episodes, the robot has learned to follow lanes without causing a reset. To ensure experience in both lanes, the robot kept driving in both directions alternately. Until finally the number of steps and the reward stabilized at the maximum value.

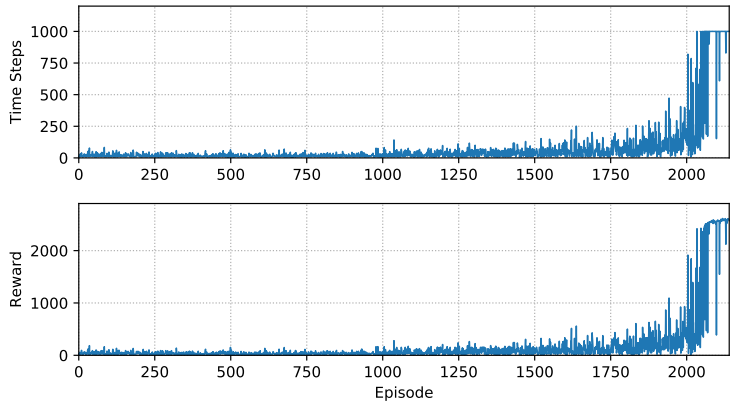

Figure 5.1: Scenario 1(DQN): Time steps and rewards for each training episode of the DQN controller 170,000 simulation steps.

5.2 Controller 2: R-STDP

If the value of the reward constant is too low, learning will take too much time and it may be difficult to see any progress. On the other hand, if the constant is too high, learning will become increasingly unstable and the robot will not learn anything. Through repeated experiments, the reward factor was determined to be 0.01. In addition, the initial weight setting should also be worth noting in order to ensure efficient learning. The motor neurons controlling the motors on both sides need to be excited at the beginning so that they can continue to cause weight changes according to the R-STDP learning rule, which requires weights greater than zero. At the same time, the initial weight values can be made as close as possible to the final weights in order to achieve training faster. The initial weights are therefore slightly larger, being set to 200, while the maximum value of the weights is limited to 3,000, so that only excitatory synaptic connections are allowed. Other detailed parameters are given in Table A.2 in appendix. It is worth noting that the parameters will change slightly due to different scenarios and methods, which will be mentioned subsequently.

5.2.1 Scenario 1

In the following, the training results will be shown according to the different perception methods in scene 1.

Height Thresholding

The following will present the training results for perceiving the lane surface by fixing the height. Due to the perception method, the same event may be repeated several times. Here we set the number of events during each step for maximum poisson frequency to 130. The resolution of the input image is 4x4.For the final weights, Figure 5.2 shows the 4x4 weights for each of the left and right sides. It can be clearly seen that the left side of the left weights and the right side of the right weights are larger and therefore the neurons are more active. Probably because these two are closer to the edges of the lane on the left and right, the neurons increase their firing frequency.

Figure 5.3 shows the training process according to the height-aware R-STDP controller. The upper part of the Figure 5.3 represents: the final position where the robot vehicle stays at the end of each training period. It is worth noting that the reset mechanism is triggered when the vehicle is more than 0.25 m from the center of the lane or when the training steps in a single period exceed 10,000 steps. It can be seen that within the first approximately 10,000 steps, the robot vehicle makes trial and error in the initial straight and curved lanes. Then after a few attempts to successfully turn the corner into the next straight lane, it is finally able to lane follow correctly at 12,000 steps until it has completed a full lane. The bottom half of the Figure 5.3 then indicates how the 4x4 total of 16 weights for each side of the motor changed during the training process. The left and right motors each have a line with larger weights in the plot, which is consistent with the values in the yellow cells in the above Figure 5.3, indicating that closer to the edge of the lane, the neurons are more active. As for the other weights, there is a clear upward expression starting at 10,000 steps. This is because at this point the robot vehicle learns to turn and is able to go straight in the next section of the lane, which leads to a high reward during this time. The high reward leads to a significant change in the value of the weight connection. The fluctuation is less at 20,000-70,000. As for the increase after 70,000, it may be an overall improvement of the previous weight values. The green line in the figure also suffers from large fluctuations and

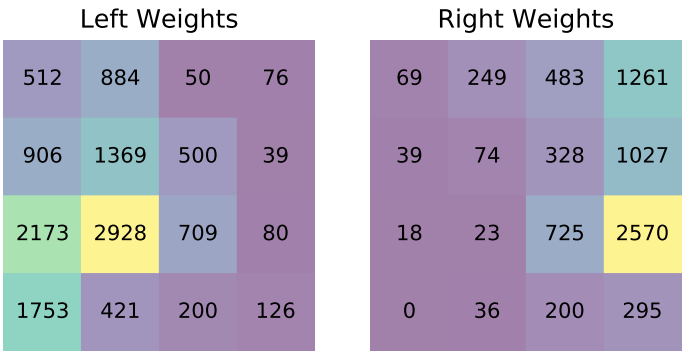

Figure 5.2: Scenario 1(Height perception): Learned connection weights to the left and right motor neuron of the R-STDP controller after 100,000 simulation steps.

inconsistency with the changes of other weights. This is a side indication that this method is less stable and not the best choice.

The number of steps required for each period of the training process is indicated in Figure 5.4. It can be seen that at the 7th period, a complete lap is already possible. In Figure 5.5, it shows the use of trained weights to test the distance from the center of the lane when the robot car runs for one lap. The average deviation is $e = 0.023m$. The data performed well overall, with only large fluctuations near the arrival at the end. The height-aware approach, therefore, still has a certain degree of accuracy. However, the limitation is the instability caused by the high fluctuation of the weights during training.

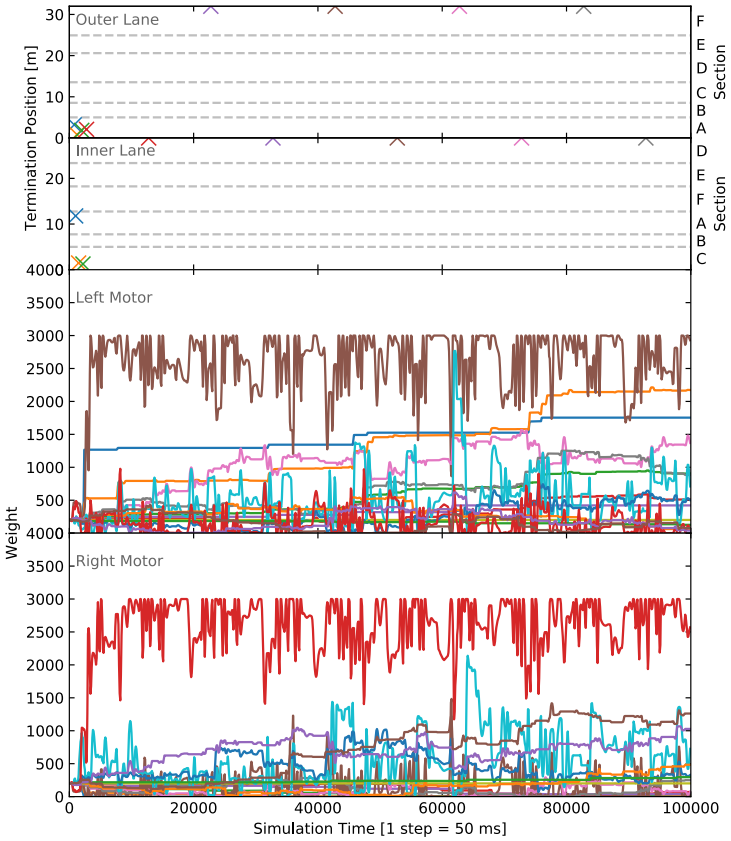

Figure 5.3: Scenario 1(Height perception): Learning progress of the R-STDP controller.

Gradient Thresholding

The following will present the training results of road perception by grid. Here we set the number of events during each step for maximum poisson frequency to 20. The resolution of the input image is 4x8. Figure 5.2 shows the final weights of 4x8 for each of the left and right sides. Here it is seen that the lower left corner of the left weights and the lower right corner of the right weights are larger, basically forming two road edges. As mentioned before, the neurons here are more active and the neurons increase their firing frequency to better follow the lane.

Figure 5.7 shows the training process of the R-STDP controller based on grid perception. In the upper part of Figure 5.7 it can be seen that the real stable keeping is done after 40,000 steps. The bottom half of Figure 5.7 shows, on the other hand, the variation of a total of 32 weights per side motor 4x8 during the training process. It is obvious to see that the weights in the figure are more unstable compared to the height-awareness. However, the red line for the left motor starts to show a significant upward trend at about 20,000 steps, which is a result of the robot vehicle successfully achieving lane keeping and thus high reward.

In Figure 5.8 it is possible to see that at the 10th period, a complete lap is already possible. However, it does not remain stable until the 14th period. In Figure 5.9 the average deviation is $e = 0.069m$. The overall fluctuation of the data is high. Several of the above images illustrate its relative instability compared to height perception. Although it was able to achieve a correct perception of the road and to follow the lane for a full circle, the algorithm needs to be improved.

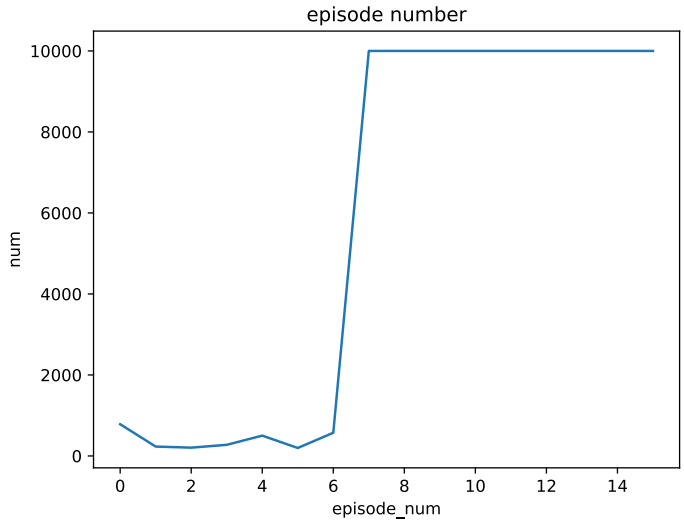

Figure 5.4: Scenario 1(Height perception):Number of steps required in each period

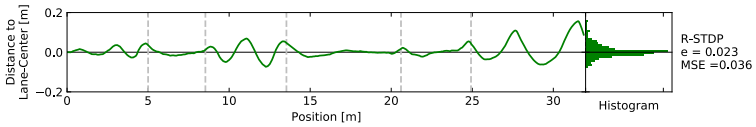

Figure 5.5: Scenario 1(Height perception):Performance distance on the outer lane

FCN recognition

In this part, we will present two different methods of obtaining states based on FCN perception.

1) FCN Grid

The following will present the training results of road perception by FCN Grid. Here we set the number of events during each step for maximum poisson frequency to 5.3. At this time the maximum number of steps for a single training cycle is set to 1,300, and the total training step length is 17,000. The resolution of the input image is 8x8, because after several attempts, the best results are presented when the resolution is 8x8. (However, the minimum resolution that can be achieved is 4x4.) Figure 5.10 shows the final weights of 8x8 for each of the left and right sides. It can be clearly seen that the larger weights are concentrated on the right side of the left weight and on the left side of the right weight, i.e., the middle part. Combining with the algorithm, we know that when extracting the edges of the image output from the edge fcn as input state, more accurate edge points and fewer points as input will make the final weights more evenly distributed.

Figure 5.11 shows the training process of the perception R-STDP controller according to FCN Grid. In the upper part of Figure 5.11 it can be seen that there is a lot of trial and error here before the real stable keeping comes at around 6000 steps. The bottom half of Figure 5.11 shows the variation of a total of 64 weights for each side of the motor 8x8 during the training process. It is clear that all weights are gradually increasing or decreasing during the first 6,000 steps. This is also the learning process, which is expressed as enhancement or inhibition on the neurons, and stabilizes between 6,000 and 10,000 steps. At about 10,000 steps, it may fall into the trap of local minimum, and all the data show a phenomenon of

Figure 5.6: Scenario 1(Grid perception): Learned connection weights to the left and right motor neuron of the R-STDP controller after 100000 simulation steps.

decreasing and then increasing (or increasing and then decreasing). At this point, there is a secondary improvement learning process to get a global optimal solution, so as to escape the trap. Thereafter it continues to stabilize until the end. In terms of the overall weights change, its stability is significantly better than the first two perception algorithms.

In Figure 5.12 it can be seen that for the first 15 periods it tries near the starting point. At the 16th period, the robotic vehicle is close to the end. And as mentioned above, the weights are greatly optimized at this point. It then remains stable from the 18th period. In Figure 5.13, the average deviation is $e = 0.025m$. The curve also shows that the ride is extremely smooth and close to the center of the lane. The stability and learning efficiency are greatly optimized compared to the first two perception algorithms.

2) FCN OpenCV

In order to find the most suitable algorithm, we searched for a similar method of lane edge acquisition, FCN OpenCV, to compare with the above method. In the following, we present the training results of road perception by FCN OpenCV. Here the number of events during each step for maximum poisson frequency is set to 5.3. The maximum number of steps in a single training cycle is set to 1,300, and the total number of training steps is 17,000. The resolution of the input image is 8x8. These parameters are consistent with FCN Grid. Figure 5.14 shows the final weights of 8x8 for each of the left and right sides. It is more obvious that the final weights are not evenly distributed. The larger weights on the left weights are concentrated in the upper right corner, while the distribution of the right weights is not concentrated.

Figure 5.15 shows the training process of the perceptual R-STDP controller according to FCN OpenCV. In the upper part of Figure 5.15 it can be seen that the stable arrival at the end point does not occur until 10,000 steps. The bottom half of Figure 5.15 then shows that the total of 64 weights for each side of the motor 8x8 changes during the training process. It is obvious that most of the weights increase or decrease gradually from 0 to 10,000 steps, and then tend to stabilize. A small number of weights will increase and decrease from the beginning to the end. The whole learning process does not converge, which may lead to

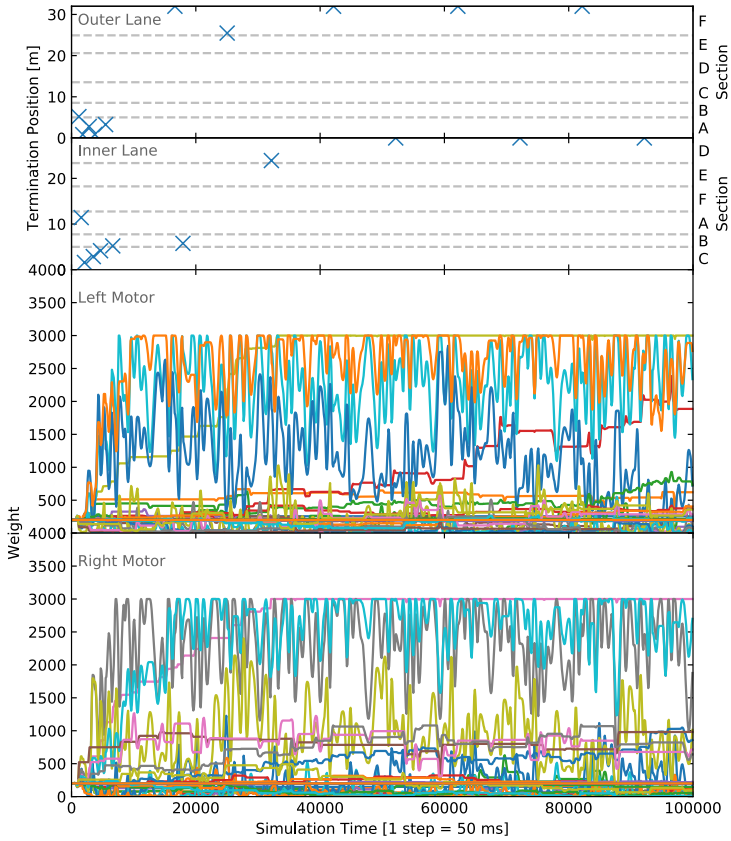

Figure 5.7: Scenario 1(Grid perception): Learning progress of the R-STDP controller.

overtraining.

In Figure 5.16 it can be seen that the robotic vehicle does not reach the end of the curve until the 24th period in a stable manner. Previously, it was stuck on consecutive curves or near curves. In Figure 5.17, the average deviation is $e = 0.031m$. The curve shows a relatively smooth driving process. However, compared to the FCN Grid, several of the above results are slightly inferior to the FCN Grid method, which shows a slightly weaker stability and learning efficiency. But compared with the first two perception algorithms, it is still a big improvement.

In summary, way1 performs better, so way1 is used later to train other different scenarios.

Right-Hand Lane Keeping in Bidirectional Road

The original scenario placed the robotic vehicle in the center of the lane. In order to be closer to the real road driving scenario, the robot vehicle is set to drive on the right. Compared to FCN Grid, the biggest parameter changes are: Maximum Poisson firing frequency is reduced from 300 to 225, number of events during each step for maximum poisson frequency is set to 5.5, reset distance is changed to 0.15, and the maximum number of training steps in a single period was changed to 2,000 due to the longer lanes. Since the vehicle was too close to the right edge of the lane at this point, the radar originally placed on top of the vehicle could not detect the right edge very well. For this reason, we moved the LiDAR to the bottom of the robotic vehicle. This results in a change in what the LiDAR detects and thus the FCN network needs to be retrained. Figure 5.18 shows the 8x8 final weights for each of the left and right sides. The larger weights on the left side are concentrated on the right side and the larger weights on the right side are concentrated on the left side, which is consistent with the FCN

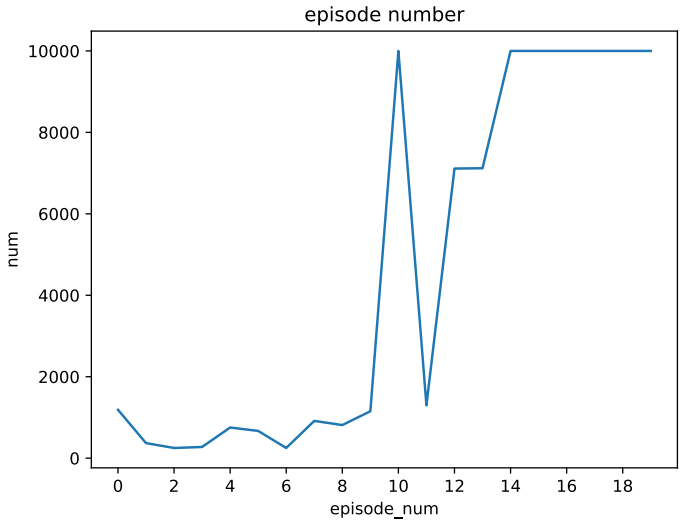

Figure 5.8: Scenario 1(Grid perception):Number of steps required in each period

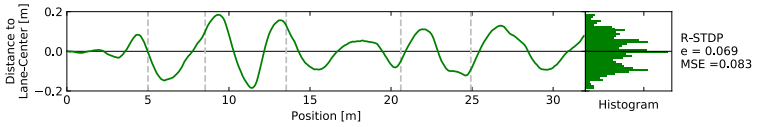

Figure 5.9: Scenario 1(Grid perception):Performance distance on the outer lane

Grid described above. Based on the previous experience, concentrating the weights on one side can control the direction better.

Figure 5.19 shows the training process of right driving with the R-STDP controller in Scenario 1 according to the FCN Grid. In the upper part of Figure 5.19 it can be seen that the stable arrival at the end does not occur until 12,000 steps. The bottom half of Figure 5.19 then shows that the total of 64 weights for each side of the motor 8x8 changes during the training process. It is clear that the weights gradually increase or decrease from 0 to 12,000 steps. After this step, it tends to be stable. The whole training process is progressive and presents a good result.

In Figure 5.20, it can be seen that the robot vehicle does not reach the end of the line until the 11th period. Before that, it was substantially closer to the end. In Figure 5.21 the average deviation is $e = 0.016m$, which is also the smallest value so far. The curve shows that the driving process is extremely smooth. Overall, although the training process is longer, this training inherits the characteristics of the FCN Grid and even outperforms it in the final result.

5.2.2 Scenario 2

To test the robustness of the algorithm, a new eight-shape scenario was introduced for testing. Compared to the FCN Grid, the parameters have been changed: the maximum Poisson firing frequency has been reduced from 300 to 225. The maximum number of training steps in a single period has been changed to 2,000 due to the longer lanes. Figure 5.22 shows the final weights of 8x8 for each of the left and right sides. Again, the larger left weight is concentrated on the right side and the larger right weight is concentrated on the left side, which is consistent with the above FCN Grid.

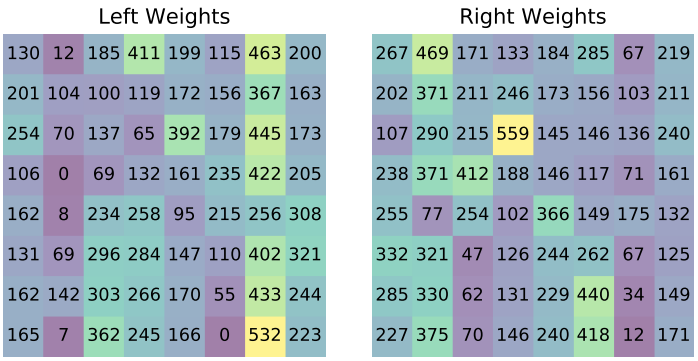

Figure 5.10: Scenario 1(FCN Grid): Learned connection weights to the left and right motor neuron of the R-STDP controller after 17000 simulation steps.

Figure 5.23 shows the training process of the R-STDP controller in Scenario 2 according to the FCN Grid. In the upper part of Figure 5.23 it can be seen that it is not until 8000 steps that a stable arrival at the end point occurs. The bottom half of Figure 5.23 shows, the variation of a total of 64 weights for each side of the motor 8x8 during the training process. It is clear that the weights gradually increase or decrease from 0 to 8,000 steps. After this step, the weights tend to be stable. However, for some weights of the right motor, the weights only increase in the first 1,000 steps and remain constant thereafter. There are also individual weights that do not converge for both the left and right motors, which may be related to the fact that the intersection in the scene cannot be identified.

In Figure 5.24, it can be seen that the robotic vehicle does not reach the end point until the 16th period. However, it is also learning gradually before that. In Figure 5.25, the average deviation is $e = 0.042m$. The curve shows some fluctuations in the driving process, and this fluctuation occurs at the intersection. Overall, although the presented results are not as stable, they are able to overcome the obstacle of directional uncertainty at intersections and further extend the robustness of the algorithm for different scenarios.

5.2.3 Scenario 3

Square scenarios were also introduced to test the utility of the algorithm. Compared to the FCN Grid, the parameters have been changed in that the maximum number of training steps in a single period has been changed to 1,300 due to the longer lanes. Figure 5.26 shows the final weights of 8x8 for each of the left and right sides. Similarly, the larger left weight is concentrated on the right side and the larger right weight is concentrated on the left side, which is consistent with the above FCN Grid.

Figure 5.27 shows the training process of the R-STDP controller in Scenario 2 according to the FCN Grid. In the upper part of Figure 5.27 it can be seen that it is not until 8,000 steps that a stable arrival at the end point occurs. The bottom half of Figure 5.27 shows, the variation of a total of 64 weights for each side of the motor 8x8 during the training process.

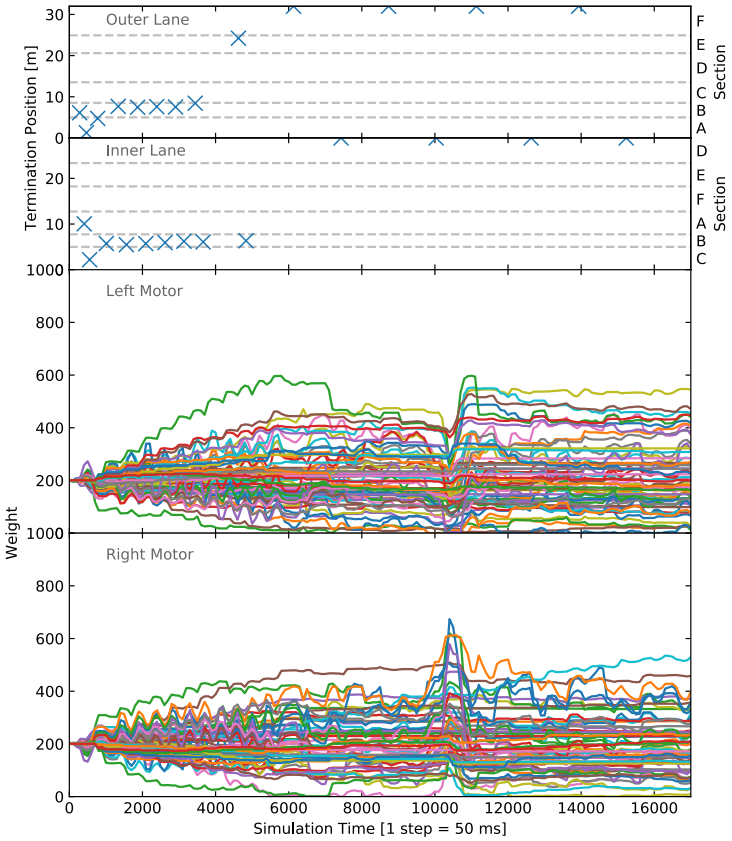

Figure 5.11: Scenario 1(FCN Grid): Learning progress of the R-STDP controller.

It is clear that the weights increase or decrease gradually from 0 to 6,000 steps. This is still followed by a trapezoid-like fluctuation. This change occurs when the inner and outer lanes alternate. The reason for this may be the difference in weights between the inner and outer lanes, especially in the turning amplitude, which leads to repeated fluctuations in weights.

In Figure 5.28 it can be seen that the robotic vehicle does not reach the end of the line stably until the 13th period. However, the previous learning progress is not obvious. The average deviation in Figure 5.29 is $e = 0.016m$, which is the smallest value, as in the previous right lane. The driving process is particularly smooth as seen in the curve, which basically follows the center of the road. Overall, although there is some fluctuation in weights, it is interesting to note that the final test driving driving results performed surprisingly well.

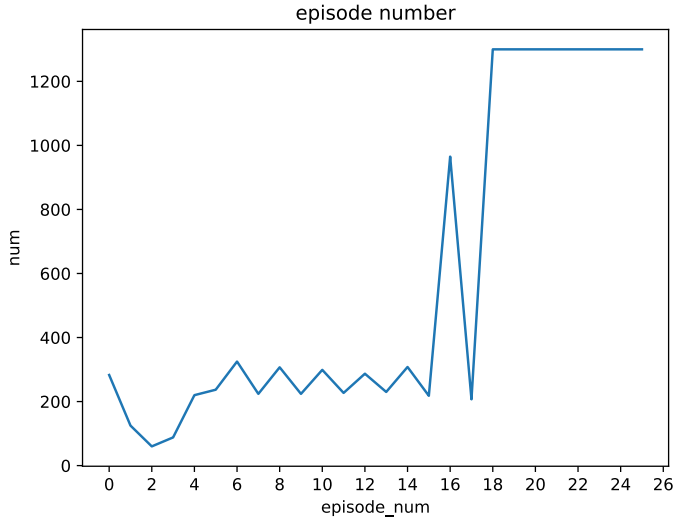

Figure 5.12: Scenario 1(FCN Grid):Number of steps required in each period

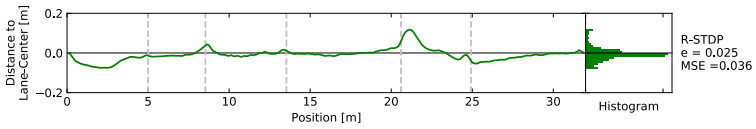

Figure 5.13: Scenario 1(FCN Grid):Performance distance on the outer lane

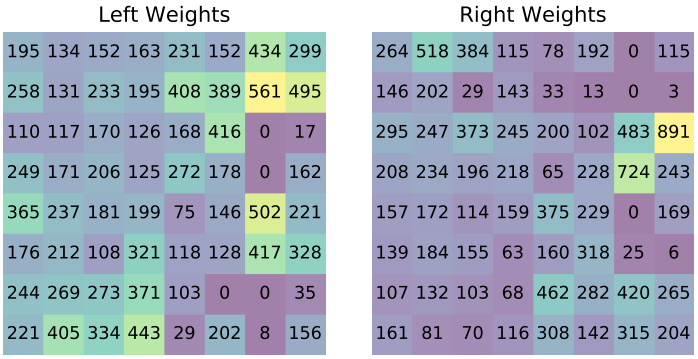

Figure 5.14: Scenario 1(FCN OpenCV): Learned connection weights to the left and right motor neuron of the R-STDP controller after 17000 simulation steps.

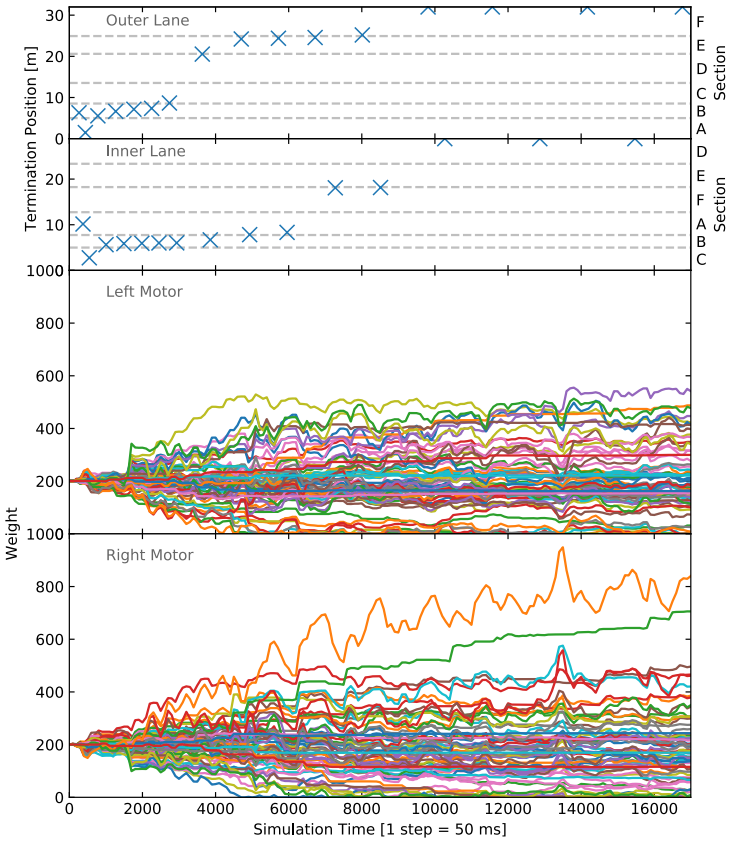

Figure 5.15: Scenario 1(FCN OpenCV): Learning progress of the R-STDP controller.

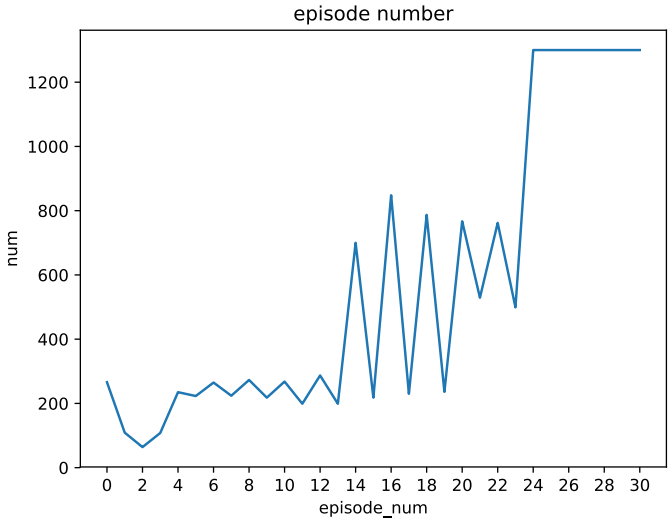

Figure 5.16: Scenario 1(FCN OpenCV):Number of steps required in each period

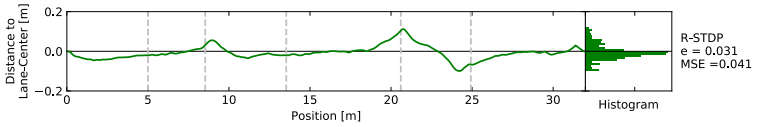

Figure 5.17: Scenario 1(FCN OpenCV):Performance distance on the outer lane

Figure 5.18: Scenario 1(Right lane): Learned connection weights to the left and right motor neuron of the R-STDP controller after 17000 simulation steps.

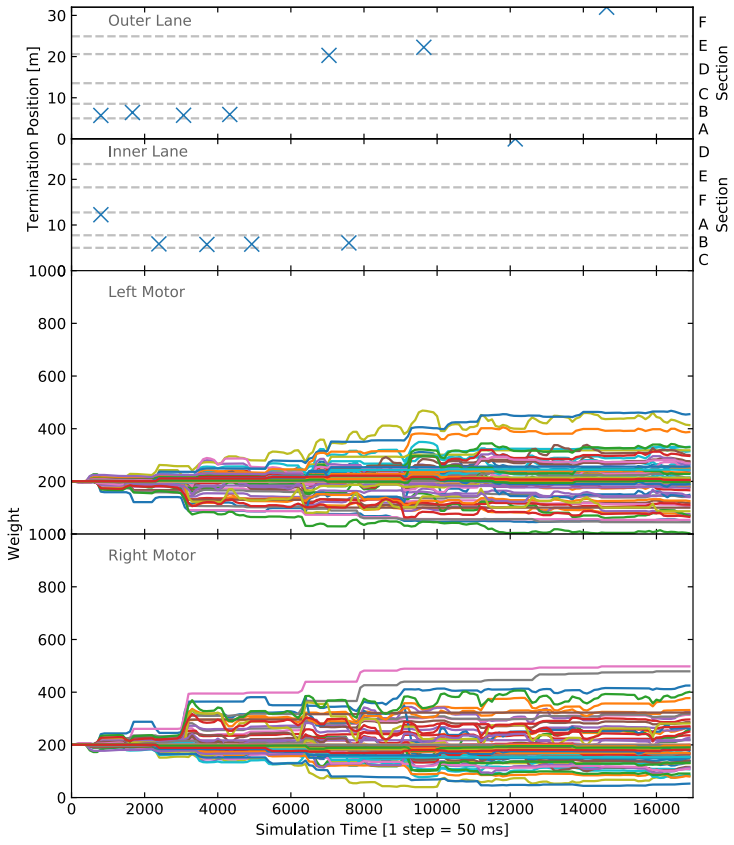

Figure 5.19: Scenario 1(Right lane): Learning progress of the R-STDP controller.

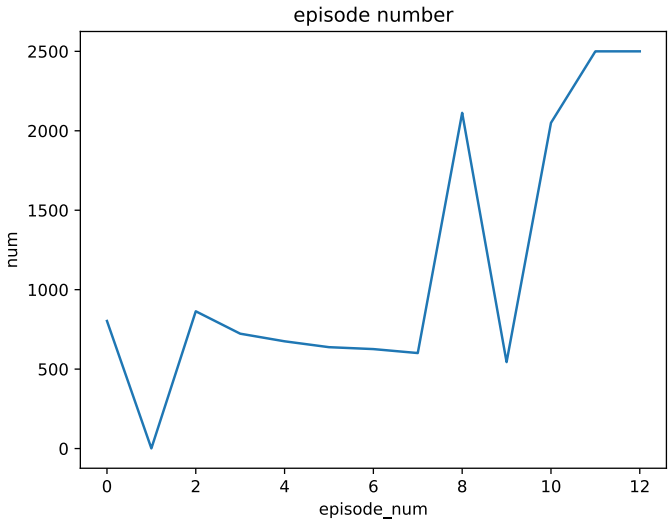

Figure 5.20: Scenario 1(Right lane):Number of steps required in each period

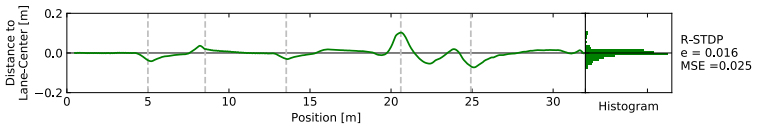

Figure 5.21: Scenario 1(Right lane):Performance distance on the outer lane

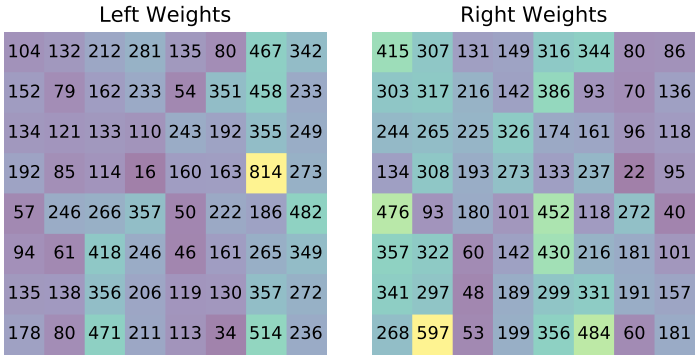

Figure 5.22: Scenario 2(FCN Grid): Learned connection weights to the left and right motor neuron of the R-STDP controller after 17000 simulation steps.

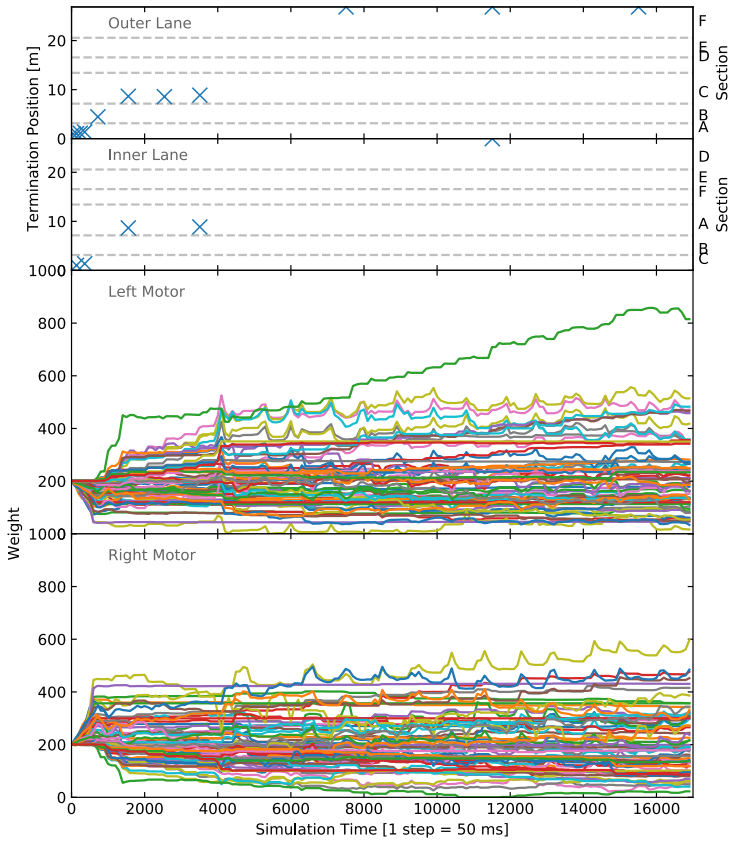

Figure 5.23: Scenario 2(FCN Grid): Learning progress of the R-STDP controller.

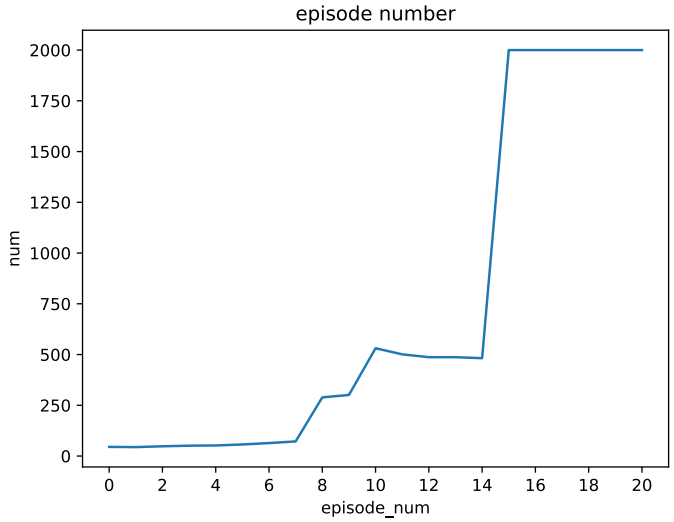

Figure 5.24: Scenario 2(FCN Grid):Number of steps required in each period

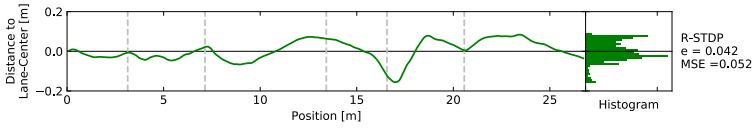

Figure 5.25: Scenario 2(FCN Grid):Performance distance on the outer lane

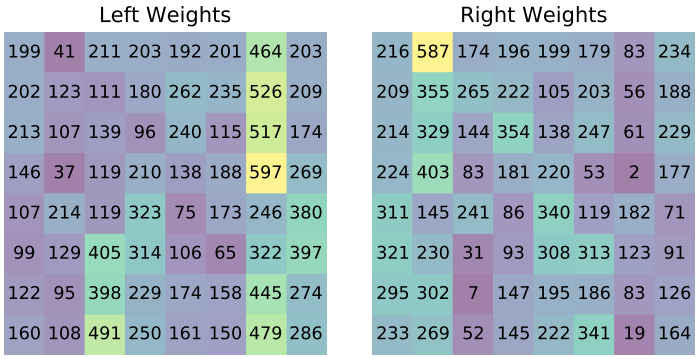

Figure 5.26: Scenario 3(FCN Grid): Learned connection weights to the left and right motor neuron of the R-STDP controller after 17000 simulation steps.

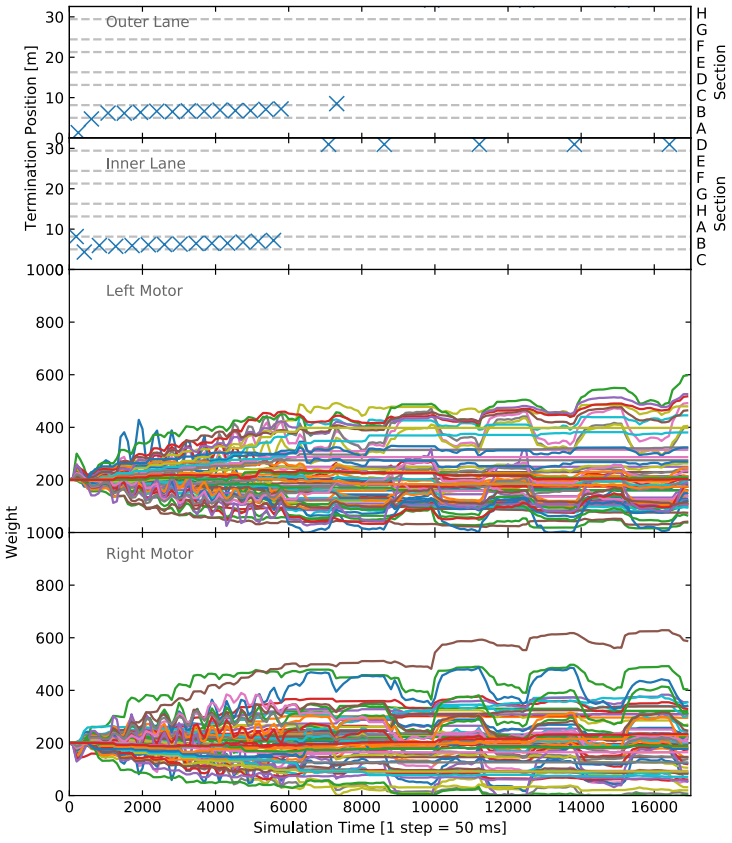

Figure 5.27: Scenario 3(FCN Grid): Learning progress of the R-STDP controller.

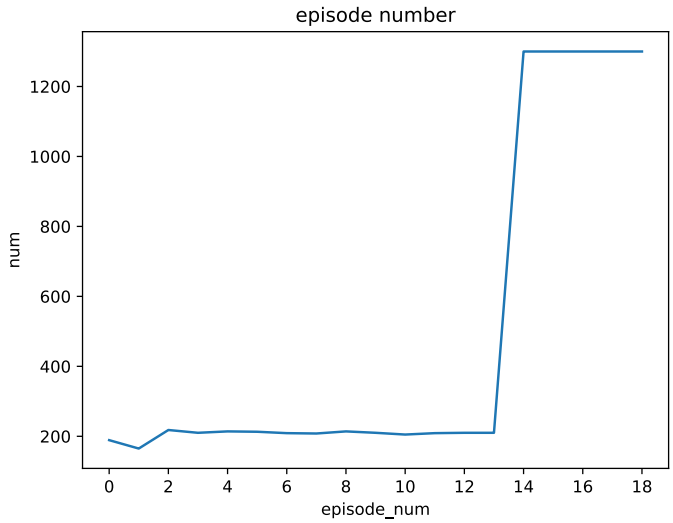

Figure 5.28: Scenario 3(FCN Grid):Number of steps required in each period

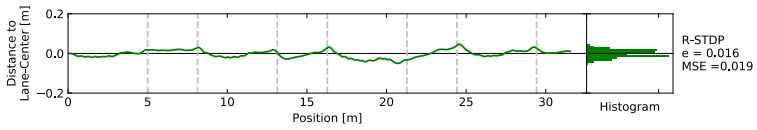

Figure 5.29: Scenario 3(FCN Grid):Performance distance on the outer lane

So far, this is all the scenarios that were trained in CoppeliaSim. AS shown in Table 5.1, FCN Grid has a good ability to adapt to the scenarios. Although there are also some flaws, it could be ported to more scenarios through subsequent parameter optimization.

Scenario-Methode	Mean Error(e)	Mean Square Error(MSE)
Scenario 1-Height perception	0.023	0.036
Scenario 1-Grid perception	0.069	0.083
Scenario 1-FCN Grid	0.025	0.036
Scenario 1-FCN OpenCV	0.031	0.041
Scenario 1-Right lane	0.016	0.025
Scenario 2-FCN Grid	0.042	0.052
Scenario 3-FCN Grid	0.016	0.019

Table 5.1: Error Comparison

Chapter 6

Conclusion and Future Outlook

In this work, two different approaches were proposed at the beginning to compare DQN and R-STDP. While both approaches achieve successful control strategies, it is clear that DQN is trial and error over a long period of time, and it is far less efficient in learning than R-STDP. The R-STDP-based perception approach was then improved. It can be found that height-based perception is more effective than grid perception. However, the scenarios where height perception is used have limitations because it requires a fixed height. Grid perception is not limited by the scenario, but it lacks stability and the actual test results are not as good as height perception. This can be improved later by changing the size of the grid and controlling the gradient variation within each grid. However, using both perception methods has a problem that the weights fluctuate too much and cannot converge well to a stable value. But the FCN perception approach can solve this problem well. When comparing the two edge perception approaches based on FCN, Grid's approach is better than OpenCV's, as its weights change more smoothly and can be traced to converge well. The reason for this is also attributed to the fact that the FCN perception results are more accurate and the variation between each frame is not too large. This way, the neurons can receive a stable input state signal, and the performance in weights will not fluctuate too much. The robustness of the algorithm is then tested in several different scenarios. In the right-hand scenario, the average deviation value is the smallest and the weights fluctuation is also the smallest, so the performance is the best. In the square scenario, although the average deviation value is also the smallest, the weights fluctuate more, which may be related to the switching between inside and outside scenarios. It can be improved by debugging the learning rate and other related parameters later. In contrast, the eight-shape scenario is due to the intersection, which leads to the uncertainty of the vehicle's judgment of the direction, thus causing the deviation amplitude to be larger. This is also a side note that the algorithm currently lacks the reward prediction capability required for more complex decision making tasks. Improvements to the algorithm are needed for application in more complex scenarios. In addition, the FCN network model needs to be retrained to recognize new scenes when the scenes change significantly. In terms of parameters, the number of events during each step for maximum poisson frequency also needs to be adjusted at any time according to the size of the input state. In the future, the R-STDP algorithm could be combined with a reward prediction model, perhaps solving more complex sequential decision tasks. This thesis lays the foundation for the future development of SNNs.

Appendix A

Simulation Parameters

DQN	Network architecture	512 - 200 - 200 - 3
	Connections	feed-forward, fully connected
	Batch size	32
	Update frequency	4
	Soft update	$\tau = 0.001$
	Learning rate	$\alpha = 0.0001$
	Buffer size	5000
ϵ-greedy policy	Pre-training steps	1000
	Annealing steps	49000
	Random probability start	1.0
	Random probability end	0.1
MDP	Discount factor	$\gamma = 0.99$
	Reset distance	0.25 m
	Max. episode steps	1000
	Time step length	$0.5s$
Robot	Motor speed straight	$v_s = 1.0m/s$
	Motor speed turn	$v_t = 0.25m/s$

Table A.1: DQN parameters

Steering wheel model	Max. speed	$v_{max} = 1.5m/s$
	Min. speed	$v_{min} = 1.0m/s$
	Turn constant	$c_{turn} = 0.5m/s$
	Max.spikes during a simulation step	$n_{max} = 15$
Poisson neurons	Max. firing rate	$300Hz$
	Number of LiDAR events for above rate	$n = 130$
SNN simulation	Simulation time	$50ms$
	Time resolution	$0.1ms$
LIF neurons	NEST model	iaf psc alpha
	Resting membrane potential	$E_L = -70.0mV$
	Capacity of the membrane	$C_m = 250.0pF$
	Membrane time constant	$\tau_m = 10.0ms$
	Time constant of postsynaptic excitatory currents	$\tau_{syn,ex} = -2.0ms$
	Time constant of postsynaptic inhibitory currents	$\tau_{syn,in} = -2.0ms$

	Duration of refractory period	$t_{ref} = 2.0ms$
	Reset membrane potential	$V_{reset} = -70.0mV$
	Spike threshold	$V_{th} = -55.0mV$
	Constant input current	$I_e = 0.0pA$
`R-STDP synapse`	NEST model	stdp dopamine synapse
	Amplitude of weight change for facilitation	$A_+ = 1.0$
	Amplitude of weight change for depression	$A_- = 1.0$
	STDP time constant for facilitation	$\tau_+ = 20.0ms$
	Time constant of eligibility trace	$\tau_c = 1000.0ms$
	Time constant of dopaminergic trace	$\tau_n = 200.0ms$
	Min. synaptic weight	0.0
	Max. synaptic weight	3000.0
	Initial synaptic weight	200.0
	Reward constant	$c_r = 0.01$
`Other training setting`	Reset distance	$0.25m$
	ROS publication rate	$20.0Hz$
	Lenth of training procedure	100000
	Max. training step	10000

Table A.2: R-STDP parameters

Bibliography

[BKC17] Badrinarayanan, V., Kendall, A., and Cipolla, R. "Segnet: A deep convolutional encoder-decoder architecture for image segmentation". In: *IEEE transactions on pattern analysis and machine intelligence* 39.12 (2017), pp. 2481–2495.

[Bin+19a] Bing, Z., Baumann, I., Jiang, Z., Huang, K., Cai, C., and Knoll, A. "Supervised learning in SNN via reward-modulated spike-timing-dependent plasticity for a target reaching vehicle". In: *Frontiers in neurorobotics* 13 (2019), p. 18.

[Bin+19b] Bing, Z., Jiang, Z., Cheng, L., Cai, C., Huang, K., and Knoll, A. "End to end learning of a multi-layered snn based on r-stdp for a target tracking snake-like robot". In: (2019), pp. 9645–9651.

[Bot12] Bottou, L. *Neural Networks: Tricks of the Trade-Stochastic Gradient Descent Tricks*. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, 2012, pp. 421–436. ISBN: 978-3-642-35288-1, 978-3-642-35289-8.

[BM10] Buchanan, K. and Mellor, J. "The activity requirements for spike timing-dependent plasticity in the hippocampus". In: *Frontiers in synaptic neuroscience* (2010), p. 11.

[Cal+17] Caltagirone, L., Scheidegger, S., Svensson, L., and Wahde, M. "Fast LIDAR-based road detection using fully convolutional neural networks". In: *2017 ieee intelligent vehicles symposium (iv)*. IEEE. 2017, pp. 1019–1024.

[Chr92] Christopher JCH Watkins, P. D. "Q-learning". In: *Machine learning* 8.3 (1992), pp. 279–292.

[Coh14] Cohen, M. X. "A neural microcircuit for cognitive conflict detection and signaling". In: *Trends in neurosciences* 37.9 (2014), pp. 480–490.

[Dav12] David Silver Richard S. Sutton, M. M. "Temporal-difference search in computer Go". In: *Mach Learn* 87 (2012), pp. 183–219.

[DSB11] Dethe, S. N., Shevatkar, V. S., and Bijwe, R. "Google driverless car". In: *International Journal of Scientific Research in Science, Engfineering and Technology* 21.2 (2011), pp. 2394–4099.

[End17] Endsley, M. R. "Autonomous Driving Systems: A Preliminary Naturalistic Study of the Tesla Model S". In: *Journal of Cognitive Engineering and Decision Making* 11.3 (2017), pp. 225–238. DOI: 10.1177/1555343417695197. eprint: https://doi.org/10.1177/1555343417695197. URL: https://doi.org/10.1177/1555343417695197.

[Fay+20] Fayyad, J., Jaradat, M. A., Gruyer, D., and Najjaran, H. "Deep Learning Sensor Fusion for Autonomous Vehicle Perception and Localization: A Review". In: *Sensors* 20.15 (2020). ISSN: 1424-8220. DOI: 10.3390/s20154220. URL: https://www.mdpi.com/1424-8220/20/15/4220.

[Flo07] Florian, R. V. "Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity". In: *Neural computation* 19.6 (2007), pp. 1468–1502.

[Gar+08] Garland, M., Le Grand, S., Nickolls, J., Anderson, J., Hardwick, J., Morton, S., Phillips, E., Zhang, Y., and Volkov, V. “Parallel computing experiences with CUDA”. In: *IEEE micro* 28.4 (2008), pp. 13–27.

[Ger+12] Gerstein, G. L., Williams, E. R., Diesmann, M., Grün, S., and Trengove, C. “Detecting synfire chains in parallel spike data”. In: *Journal of neuroscience methods* 206.1 (2012), pp. 54–64.

[Ger+14] Gerstner, W., Kistler, W. M., Naud, R., and Paninski, L. *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.

[GA09] Ghosh-Dastidar, S. and Adeli, H. “Third generation neural networks: Spiking neural networks”. In: *Advances in Computational Intelligence*. Springer, 2009, pp. 167–178.

[Iak+15] Iakymchuk, T., Rosado-Muñoz, A., Guerrero-Martınez, J. F., Bataller-Mompeán, M., and Francés-Vıllora, J. V. “Simplified spiking neural network architecture and STDP learning algorithm applied to image classification”. In: *EURASIP Journal on Image and Video Processing* 2015.1 (2015), pp. 1–11.

[Ier06] Ierusalimschy, R. *Programming in Lua, Second Edition*. PUC-RioBrazil: Rio de Janeiro, 2006, p. 308. ISBN: 85-903798-2-5.

[Izh04] Izhikevich, E. M. “Which model to use for cortical spiking neurons?” In: *IEEE transactions on neural networks* 15.5 (2004), pp. 1063–1070.

[Izh07] Izhikevich, E. M. “Solving the distal reward problem through linkage of STDP and dopamine signaling”. In: *Cerebral cortex* 17.10 (2007), pp. 2443–2452.

[Jua+22] Juarez-Lora, A., Ponce-Ponce, V. H., Sossa, H., and Rubio-Espino, E. “R-STDP Spiking Neural Network Architecture for Motion Control on a Changing Friction Joint Robotic Arm”. In: *Frontiers in Neurorobotics* 16 (2022).

[Kai+16] Kaiser, J., Tieck, J. C. V., Hubschneider, C., Wolf, P., Weber, M., Hoff, M., Friedrich, A., Wojtasik, K., Roennau, A., Kohlhaas, R., et al. “Towards a framework for end-to-end control of a simulated vehicle with spiking neural networks”. In: *2016 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*. IEEE. 2016, pp. 127–134.

[Khe+18] Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S. J., and Masquelier, T. “STDP-based spiking deep convolutional neural networks for object recognition”. In: *Neural Networks* 99 (2018), pp. 56–67.

[KB14] Kingma, D. P. and Ba, J. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[Lev11] Levinson Jesse Askeland Jake, B. J. “Towards fully autonomous driving: Systems and algorithms”. In: *IEEE Intelligent Vehicles Symposium (IV)* (June 2011), pp. 163–168.

[Lia20] Liang, Z. “Thoughts on the Development Status and Application Direction of the Driverless Industry”. In: *International Journal of Scientific Engineering and Science* 4.3 (2020), pp. 15–16.

[LSD15] Long, J., Shelhamer, E., and Darrell, T. “Fully convolutional networks for semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.

[Lu+21] Lu, H., Liu, J., Luo, Y., Hua, Y., Qiu, S., and Huang, Y. "An autonomous learning mobile robot using biological reward modulate STDP". In: *Neurocomputing* 458 (2021), pp. 308–318.

[Luc14] Lucas, N. "Comparative Analysis Between Gazebo and V-REP Robotic Simulators". In: (Dec. 2014). DOI: 10.13140/RG.2.2.18282.36808.

[Maa11] Maass, W. "Liquid state machines: motivation, theory, and applications". In: *Computability in context: computation and logic in the real world* (2011), pp. 275–296.

[Moz+18] Mozafari, M., Kheradpisheh, S. R., Masquelier, T., Nowzari-Dalini, A., and Ganjtabesh, M. "First-spike-based visual categorization using reward-modulated STDP". In: *IEEE transactions on neural networks and learning systems* 29.12 (2018), pp. 6178–6190.

[NR98] Nelson, M. and Rinzel, J. "The hodgkin—huxley model". In: *The book of genesis*. Springer, 1998, pp. 29–49.

[Pan+20] Pandey, A., Panwar, V. S., Hasan, M. E., and Parhi, D. R. "V-REP-based navigation of automated wheeled robot between obstacles using PSO-tuned feedforward neural network". In: *Journal of Computational Design and Engineering* 7.4 (Apr. 2020), pp. 427–434. ISSN: 2288-5048. DOI: 10.1093/jcde/qwaa035. eprint: https://academic.oup.com/jcde/article-pdf/7/4/427/33646692/qwaa035.pdf. URL: https://doi.org/10.1093/jcde/qwaa035.

[PMD10] Potjans, W., Morrison, A., and Diesmann, M. "Enabling functional neural circuit simulations with distributed computing of neuromodulated plasticity". In: *Frontiers in computational neuroscience* 4 (2010), p. 141.

[Qui+09] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A. Y., et al. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software* 3.3.2 (2009), p. 5.

[QPG22] Quintana, F. M., Perez-Pena, F., and Galindo, P. L. "Bio-plausible digital implementation of a reward modulated STDP synapse". In: *Neural Computing and Applications* (2022), pp. 1–12.

[Rao+17] Rao, P. A., Kumar, B. N., Cadabam, S., and Praveena, T. "Distributed deep reinforcement learning using tensorflow". In: *2017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication (CTCEEC)*. IEEE. 2017, pp. 171–174.

[Ric98] Richard S. Sutton, A. G. B. *Reinforcement Learning: An Introduction-Monte Carlo Methods*. Cambridge: MIT press, 1998, pp. 111–131. ISBN: 9780262257053.

[Ric18] Richard S. Sutton, A. G. B. *Reinforcement Learning, second edition: An Introduction*. Cambridge: MIT press, Nov. 2018. ISBN: 0262352702, 9780262352703.

[RFB15] Ronneberger, O., Fischer, P., and Brox, T. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[Shi+17] Shi, W., Alawieh, M. B., Li, X., and Yu, H. "Algorithm and hardware implementation for visual perception system in autonomous vehicle: A survey". In: *Integration* 59 (2017), pp. 148–156.

[Sia+18] Siam, M., Gamal, M., Abdel-Razek, M., Yogamani, S., Jagersand, M., and Zhang, H. "A comparative study of real-time semantic segmentation for autonomous driving". In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2018, pp. 587–597.

[Sun+20] Sun, J., Cao, Y., Chen, Q. A., and Mao, Z. M. “Towards robust {LiDAR-based} perception in autonomous driving: General black-box adversarial sensor attack and countermeasures”. In: (2020), pp. 877–894. URL: https://www.usenix.org/conference/usenixsecurity20/presentation/sun.

[SK20] Sung, M. and Kim, Y. “Training spiking neural networks with an adaptive leaky integrate-and-fire neuron”. In: *2020 IEEE international conference on consumer electronics-Asia (ICCE-Asia)*. IEEE. 2020, pp. 1–2.

[TT11] Teichman, A. and Thrun, S. “Practical object recognition in autonomous driving and beyond”. In: *Advanced Robotics and its Social Impacts* (Oct. 2011), pp. 35–38. DOI: 10.1109/ARSO.2011.6301978.

[Tie+19] Tieck, J. C. V., Becker, P., Kaiser, J., Peric, I., Akl, M., Reichard, D., Roennau, A., and Dillmann, R. “Learning target reaching motions with a robotic arm using brain-inspired dopamine modulated STDP”. In: (2019), pp. 54–61.

[Vol13] Volodymyr Mnih Koray Kavukcuoglu, D. S. “Playing atari with deep reinforcement learning”. In: *arXiv* 1312.5602 (2013).

[Vol15] Volodymyr Mnih Koray Kavukcuoglu, D. S. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), pp. 529–533.

[YXL16] Yan, C., Xu, W., and Liu, J. “Can you trust autonomous vehicles: Contactless attacks against sensors of self-driving vehicle”. In: *Def Con* 24.8 (2016), p. 109.

[YCY03] Yang, D., Cai, B., and Yuan, Y. “An improved map-matching algorithm used in vehicle navigation system”. In: 2 (2003), 1246–1250 vol.2. DOI: 10.1109/ITSC.2003.1252683.

[Yan+19] Yang, K., Hu, X., Bergasa, L. M., Romera, E., and Wang, K. “Pass: Panoramic annular semantic segmentation”. In: *IEEE Transactions on Intelligent Transportation Systems* 21.10 (2019), pp. 4171–4185.