

# **Multi-agent Reinforcement Learning on the EDGE through Integrating Spiking Neural Network and Federated Learning**

by

Mohammad Tayefe Ramezanlou

Thesis proposal submitted to  
The Faculty of Graduate Studies and Research  
in partial fulfillment of the requirements for the degree of  
Doctor of Philosophy in Electrical Engineering  
Ottawa-Carleton Institute of Electrical and Computer Engineering Department of  
Systems and Computer Engineering

Carleton University

Ottawa, Ontario

Fall, 2024

Copyright ©

2024, Mohammad Tayefe Ramezanlou

# Abstract

This proposal outlines an innovative approach to multiagent reinforcement learning by integrating Spiking Neural Networks (SNNs) with Federated Learning (FL), aimed at advancing the capabilities of edge computing. This integration promises notable improvements in learning efficiency and data privacy but presents unique challenges requiring further investigation. The research will focus on four pivotal areas: enhancing learning stability in SNNs, reducing communication overhead in FL, exploring the heterogeneity of spiking neuron models, and strengthening resilience against model poisoning attacks.

This research aims to develop adaptive mechanisms for SNNs to maintain learning stability amidst dynamic environmental rewards, thereby addressing the critical balance between synaptic plasticity and equilibrium. The proposal also aims to innovate in managing communication overhead by introducing adaptive communication strategies that ensure efficiency without compromising responsiveness, leveraging algorithms that reduce redundancy in model updates.

Furthermore, the research will investigate the implications of neuron model diversity within SNNs, examining how heterogeneity affects learning outcomes and system performance. This includes employing methodologies such as Neuroevolution of Augmenting Topologies (NEAT), transfer learning, and evolutionary optimization algorithms to optimize the global model. Lastly, the proposal emphasizes enhancing the system's defense against model poisoning attacks, proposing the use of robust statistical methods to detect and mitigate such risks.

By addressing these areas, this research aims to significantly contribute to the develop-

ment of more resilient, efficient, and adaptive multiagent systems, leveraging the synergies between SNNs and FL in dynamic and potentially adversarial environments. This work aims to improve the efficiency of bio-inspired neural network models in edge computing and ensure the integrity and privacy of data in decentralized networks.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Spiking Neural Networks: Models and Learning Algorithms . . . . .	4
1.1.1	Neuron model . . . . .	4
1.1.2	Learning Approaches in SNNs . . . . .	7
1.2	Literature Review . . . . .	10
1.2.1	Spiking Neural Network . . . . .	10
1.2.2	Federated Learning . . . . .	13
<b>2</b>	<b>Modular Learning in SNNs for Optimal Multi-Agent Decision-Making</b>	<b>19</b>
2.1	Introduction . . . . .	19
2.2	The ATD problem and SNN-based solution . . . . .	20
2.3	Learning using R-STDP . . . . .	20
2.4	Network structure and encoding method . . . . .	25
2.5	Results . . . . .	30
2.5.1	Simulation without noise . . . . .	34
2.5.2	Simulation with noise . . . . .	35
2.6	Conclusion . . . . .	37
<b>3</b>	<b>Integration of R-STDP and Federated Learning</b>	<b>38</b>
3.1	Consensus Flying Problem . . . . .	38
3.2	Proposed Method . . . . .	40

3.2.1	Network Structure . . . . .	40
3.2.2	Training algorithm . . . . .	44
3.2.3	Weight Stabilization using Reward-Modulated Competitive Synaptic Equilibrium (R-CSE) . . . . .	46
3.2.4	Federated Learning for Consensus Flying . . . . .	54
3.3	Results and Discussion . . . . .	58
3.3.1	Simulation without Federated Learning (FL) . . . . .	58
3.3.2	Simulation with Federated Learning (FL) and Reward-Modulated Competitive Synaptic Equilibrium (RCSE) . . . . .	68
3.4	Conclusion . . . . .	71
<b>4</b>	<b>Vision-based Multi-Agent Autonomous Docking Using a Deep Spiking Neural Network and Reward-modulated Spike-Timing-Dependent Plasticity</b>	<b>73</b>
4.1	Preliminaries . . . . .	73
4.1.1	Problem definition: Docking Missions with Swarm Robots . . . . .	73
4.1.2	Event-Based Camera Simulation . . . . .	76
4.1.3	Dynamic Model of the Payload and Agents . . . . .	78
4.1.4	Neuron model . . . . .	80
4.2	Proposed Neuromorphic Framework . . . . .	83
4.2.1	Deep Spiking Neural Network (SNN) Structure . . . . .	83
4.2.2	Training Deep Spiking Neural Network (SNN) with Reward-modulated Spike-Timing-Dependent Plasticity (R-STDP) . . . . .	96
4.3	Results . . . . .	102
<b>5</b>	<b>Discussion and Conclusion</b>	<b>105</b>
5.1	Future Work . . . . .	105

# List of Figures

1.1	Schematic diagram to illustrate distributed learning and Federated Learning (FL) [1]. . . . .	1
1.2	Simple model of a Spiking Neural Network (SNN). The spike pattern shows that the neurons spike whenever the voltage of the neuron reaches a threshold.	4
2.1	Active target defense game with three agents (LOS angles are shown for both agents). . . . .	21
2.2	Two Spiking Neural Network (SNN)s simultaneously play and control the invader and defender. Each agent uses the LOS angle and relative velocities for training. . . . .	21
2.3	An illustration of input current to Spiking Neural Network (SNN) and synaptic current to the output neurons after training for a single connection. . . . .	22
2.4	Relative velocities and LOS angles used in the reward function . . . . .	24
2.5	Network structure and encoding process for the input layer (Defender). Each neuron is associated with a membership function in Gaussian Receptive Fields (GRF). The Gaussian Receptive Fields (GRF) encodes an input State ( $S^t$ ) at each time step. There is both a training phase when ( $\xi = 0$ ) and an operating phase when ( $\xi = 1$ ). . . . .	26
2.6	Reward, eligibility trace, and weight change during the simulation for $W^{3-3}$ . The $C(t)R(t)$ changes the synaptic weight by considering activation strength and reward value. . . . .	27

2.7	Spiking Neural Network (SNN)’s performance during training. Hollow circles show the initial positions. . . . .	31
2.8	Changes in synaptic weights during training. Only Maximum synaptic weights for both agents are shown. . . . .	32
2.9	Spiking Neural Network (SNN)’s performance after training. The highlighted region shows the defender’s dominant region. The purple dot is the optimal capture point. The CO Type-E shows the reachable regions of the Invader and the Defender. . . . .	34
2.10	Spiking Neural Network (SNN)’s performance in noisy conditions. A white Gaussian noise with a variance of 0.01 is added to the measured inputs. . . .	36
3.1	The central server (the leader) and the surrounding follower agents (white drones). The follower agents learn to fly in a formation to maintain the commanded distance. The Local models trained individually by follower agents are sent to the leader. The leader aggregates the models and sends back the global model for another round of training of the follower agents. . . . .	39
3.2	Spiking Neural Network (SNN) structure with encoding and decoding layers. Each sub-layer consists of a fuzzy encoder and the F2S Converter, with the output layer receiving inputs from synaptic weights and a random action selector. During the training phase, the output layer receives input only from the random action selector, which then shifts to synaptic weight inputs after the training (test phase). . . . .	41
3.3	The fuzzy encoding principle for the input sub-layer. . . . .	43
3.4	Synaptic weight change for $\lambda = 5$ , $\Psi^S = 15.5$ , and $\mathcal{AR}_{max}^G = 1$ . . . . .	52
3.5	Reward-based learning rate and decay rate functions. In the blue region (active learning rate), the reward adjusts the weights, and in the red region (active decay rate), the RCSE method controls synaptic growth. . . . .	52

3.6 RCSE working principle in inhibiting the adjacent synaptic connections. The heatmap shows the synaptic weight matrix. Neurons have different firing strengths due to the difference in fuzzy membership values, which affects the increase or decrease rate and shapes the patterns in the synaptic weight matrix.	53
3.7 Measured distances used for evaluating swarm flight performance and collision detection.	61
3.8 Agents' trajectory during the test phase	61
3.9 Variation of distances within the swarm during the test phase	62
3.10 Adaptive response to commanded distance adjustments - reconfiguration during the test phase	62
3.11 Trajectory adaptations of following agents in response to reward change for the leader during the test phase.	64
3.12 Variations in distances after reward changes and Leader becomes Obstacle - test phase	64
3.13 Synaptic Weights before Reward change in Reward-Modulated Competitive Synaptic Equilibrium (RCSE) method.	66
3.14 Synaptic Weights after Reward change in Reward-Modulated Competitive Synaptic Equilibrium (RCSE) method.	66
3.15 Synaptic Weights before Reward change in Multiplicative Synaptic Normalization method.	66
3.16 Synaptic Weights after Reward change in Multiplicative Synaptic Normalization method.	66
3.17 Synaptic weights increase after reward change in Reward-Modulated Competitive Synaptic Equilibrium (RCSE) method.	67
3.18 Synaptic weights decrease after reward change in Reward-Modulated Competitive Synaptic Equilibrium (RCSE) method.	67
3.19 Distances during the test phase before reward change in the proposed event-triggered Federated Learning (FL) method.	69

3.20 Distances during the test phase after reward change in the proposed event-triggered Federated Learning (FL) method. . . . .	69
3.21 Frobenius norm of the Agent 1's weighs during the learning phase. The reward changes for the Leader after 600 s. . . . .	70
3.22 Communication times for agents and the Central Server (Leader). Red and blue dots show the times that agents and the Central Server have sent their model, respectively. . . . .	70
4.1 Swarm of robots performing docking mission. Each agent pushes the Payload towards the Central Hub to align them together. The central Hub and agents are equipped with a Dynamic Vision Sensor (DVS) that streams high-frequency data for the proximity mission. . . . .	74
4.2 Deep Spiking Neural Network (SNN) Architecture with Entropy-Based Adaptive Pooling and Structured Hidden Layers for Swarm Robotics. The purple connections are excitatory, and the yellow connections are inhibitory. . . . .	84
4.3 Structured First Hidden Layer Architecture with Active LED Marker (ALM), Payload, and Agent Perception Repositories for Visual Perception. . . . .	89
4.4 Structured Second Hidden Layer Architecture with Mission-Phase and Regulatory Neuron Repositories. . . . .	92
4.5 Reward function pipeline for the payload case. The pipeline illustrates the raw Dynamic Vision Sensor (DVS) events, the attention-based fuzzy map, the normalized hidden layer activity during the initial training phase ( $r_q^{(1)}$ ), and the modulation signal calculated as the difference between the attention map and the hidden layer activity ( $\alpha_q$ ). . . . .	98
4.6 Reward function pipeline for the agents case. This pipeline demonstrates the raw Dynamic Vision Sensor (DVS) events for multiple agents, the reduced entropy-based fuzzy map used for attention, the hidden layer's normalized activity, and the modulation signal computed for reward-based learning in the spiking neural network. . . . .	99

# List of Tables

1.1	Spike Response Function of the neurons in an Spiking Neural Network (SNN).	5
2.1	Parameter values for LIF neuron model [10] . . . . .	30
2.2	Parameter values for Reward-modulated Spike-Timing-Dependent Plasticity (R-STDP) . . . . .	31
3.1	Parameter values for Leaky Integrate-and-Fire (LIF) neuron model [10] . . .	59
3.2	Simulation Parameters . . . . .	60
3.3	Comparative Performance Analysis of the Proposed Aggregation Algorithm+Reward- Modulated Competitive Synaptic Equilibrium (RCSE) and Reward-Modulated Competitive Synaptic Equilibrium (RCSE) . . . . .	70
4.1	Physical parameters of the payload and agents used in the docking simulations.	103
4.2	Neuron layer configurations: model, excitability type, number of neurons $N$ , and simulation time step $\Delta t$ . . . . .	103

# Acronyms

**ALM** Active LED Marker

**CDM** Collaborative Docking Mission

**DVS** Dynamic Vision Sensor

**FFM** Formation Flying Mission

**FL** Federated Learning

**GRF** Gaussian Receptive Fields

**LIF** Leaky Integrate-and-Fire

**MPC** Mission Phase Control

**MPS** Mission Phase Switch

**R-STDP** Reward-modulated Spike-Timing-Dependent Plasticity

**RCSE** Reward-Modulated Competitive Synaptic Equilibrium

**SNN** Spiking Neural Network

# Chapter 1

## Introduction

Federated Learning (FL) is a groundbreaking approach to machine learning that allows models to be trained directly on edge devices without centralizing data. This method addresses significant concerns regarding data privacy and security by keeping sensitive information local to the device it originates from, thus preventing data leakage and enhancing user privacy. FL represents a shift towards decentralized, privacy-preserving machine learning models, making it a critical area of research for advancing machine learning technologies that are both accessible and secure.

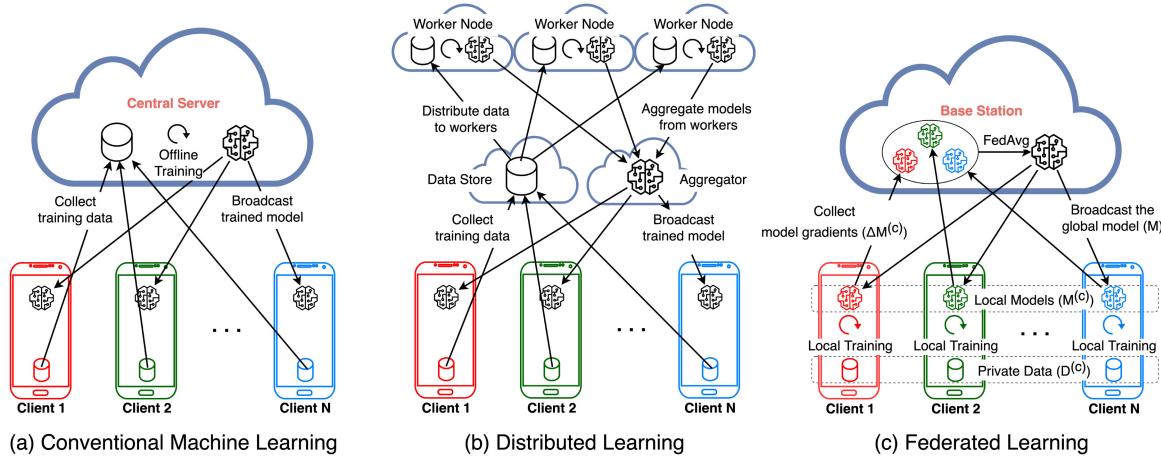


Figure 1.1: Schematic diagram to illustrate distributed learning and FL [1].

Figure 1.1 shows a diagram of distributed learning and FL. In conventional learning, the data is collected from the clients, and the model is trained offline. In distributed learning, the dataset is collected at a single data store, and then it is distributed across multiple worker nodes for training. In FL, the training is performed at the client, and model gradients are communicated to the central server, which aggregates the gradients, updates the global model, and broadcasts it back to all the clients. Each of these processes is iterative, and models for the clients are periodically updated.

Spiking Neural Network (SNN)s bring a new dimension to the capabilities of FL. Inspired by the biological processes of the human brain, SNNs operate on an event-driven basis, processing information only in response to stimuli. This method of operation significantly reduces the power consumption of these networks, making them particularly suited for deployment across distributed, battery-operated devices. The integration of SNNs with FL leverages these unique advantages, combining energy efficiency with the privacy-preserving features of FL to address the challenges faced by traditional artificial neural network models in federated settings [2, 3].

The energy-efficient nature of SNNs aligns perfectly with the objectives of FL, where models are trained across a network of distributed devices without centralized data collection. Unlike conventional artificial neural networks that require continuous data flow and computation, SNNs' event-driven operation allows for significant reductions in energy consumption, which is critical for battery-operated agents participating in FL. This characteristic of SNNs supports the development of low-power, distributed learning solutions, enabling more agents to participate in FL without compromising on power efficiency [4].

Furthermore, the integration of SNNs in FL scenarios can also help overcome limitations associated with bandwidth-constrained environments. The sparse nature of data representation and communication in SNNs means that less information needs to be exchanged between devices and the central server during the training process. This efficiency is crucial in FL environments, where network bandwidth and connectivity can significantly impact the feasibility and performance of distributed learning systems [5].

Another critical advantage of SNNs in the context of FL is their compatibility with neuromorphic hardware. Neuromorphic chips, designed to replicate the neural structures of the human brain, provide an ideal platform for deploying SNNs. This synergy between neuromorphic computing and SNNs paves the way for the development of highly efficient, scalable, and adaptive FL systems capable of leveraging the full potential of edge computing [6].

Despite these advantages, the integration of SNNs with FL presents several challenges. The primary hurdle is the complexity of training SNNs in complex problems. The dynamic and temporal nature of SNNs introduces new challenges in developing effective FL protocols that can accommodate the unique online learning rules of SNNs. Proper scheduling of communication rounds within the local SNN time steps is essential for successful collaborative training [7].

Moreover, managing the performance trade-offs associated with the frequency of communication rounds in FL is another significant challenge. Experiments have shown the impact of the number of time steps between local updates and the frequency of model aggregation on the training performance of SNNs. Finding an optimal balance to maximize model performance while minimizing communication overhead is crucial for the efficient deployment of FL systems powered by SNNs [8].

Additionally, communication constraints and the non-stationarity of data distribution pose significant challenges. The need for larger model update intervals to reduce communication costs and the changing nature of data over time (e.g., reward function in RL) require innovative solutions to maintain the accuracy and reliability of FL systems employing SNNs [9].

Despite these challenges, the potential benefits of integrating SNNs with FL justify continued research and development in this area. The combination of energy efficiency, privacy preservation, and compatibility with neuromorphic hardware, along with the distributed and collaborative nature of FL, represents a compelling approach to deploying machine learning models in real-world applications [10].

## 1.1 Spiking Neural Networks: Models and Learning Algorithms

### 1.1.1 Neuron model

In the study of computational neuroscience and the development of neural networks, particularly SNN, various neuron models have been proposed to simulate the electrical activity of neurons. These models range from simple to complex, aiming to capture the essential features of neuronal dynamics. The diagram in Figure 1.2 illustrates a basic network consisting of spiking neurons. In this network, the pre-synaptic neuron (pre-neuron) in the input layer spikes in response to the input ( $I$ ) and transmits these spikes through synaptic weights to the post-synaptic neuron (post-neuron). Whenever the pre-neuron spikes, the post-neuron receives input current with a value of  $W$ .

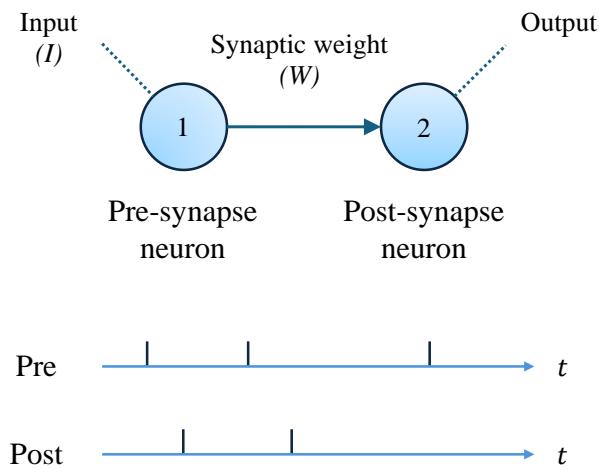


Figure 1.2: Simple model of a SNN. The spike pattern shows that the neurons spike whenever the voltage of the neuron reaches a threshold.

Table 1.1 shows the differential equations used in the network. In SNNs, the activation function, like ReLu and tanh in regular neural networks, is replaced by a differential equation that mimics the biological neuron activity.

---

Spike Response Function	
<b>Pre-Neuron</b>	$\dot{V}_1 = \frac{1}{\tau_m} [E_L - V_1 + R_m I]$
<b>Post-Neuron</b>	$\dot{V}_2 = \frac{1}{\tau_m} [E_L - V_2 + R_m (W\delta(t - t_{\text{pre}}))]$

---

Table 1.1: Spike Response Function of the neurons in an SNN.

Several spike response functions have been proposed to emulate neuron activity. In this chapter, the Leaky-Integrate and Fire model is represented because it is simple and accurate enough in emulating real neurons. This section overviews the neuron model and its characteristics.

### Leaky-Integrate and Fire model

The Leaky Integrate-and-Fire (LIF) model is a biological model that can be represented as a circuit with a resistor and capacitor and represents a first-order dynamic system [11],

$$R_m C_m \frac{dV_m(t)}{dt} = E_l - V_m(t) + R_m I(t) \quad (1.1)$$

where  $V_m(t)$  is the neuron's membrane potential shown as  $V_1$  and  $V_2$  in Table 1.1,  $R_m$  is the membrane resistance,  $C_m$  is the membrane capacitance,  $E_l$  is the resting potential, and  $I(t)$  is the input current. The neuron spikes when its potential reaches the threshold potential ( $V_{th}$ ). The potential of the neuron immediately reaches the reset potential ( $V_{res}$ ) after it spikes.

The spike rate is a parameter that determines how fast the neuron spikes [12].

$$r[\text{Hz}] = \frac{1}{t_{isi} [\text{s}]} \quad (1.2)$$

where  $t_{isi}$  is the inter-spike interval that can be calculated using the neuron model, when the potential of a neuron reaches the threshold potential, it fires. Therefore, based on the analytical solution of (1.1), the inter-spike interval time can be written as,

$$t_{isi} = \tau_m \ln \left( \frac{E_l + R_m I - V_{res}}{E_l + R_m I - V_{th}} \right) \quad (1.3)$$

where  $\tau_m$  is the membrane time constant.

According to (1.3), the following condition should be satisfied to have a finite value for  $t_{isi}$ ,

$$E_l + R_m I - V_{th} > 0 \quad (1.4)$$

or

$$I > \frac{V_{th} - E_l}{R_m} \quad (1.5)$$

which means that the input current higher than the above value generates spikes.

After calculating the minimum input for neurons, we must find the maximum input based on the inter-spike interval. Equation (1.3) can be written as,

$$t_{isi} = \tau_m \ln \left( 1 + \frac{V_{th} - V_{res}}{E_l + R_m I - V_{th}} \right) \quad (1.6)$$

Equation (1.6) can be approximated using the Maclaurin series for the natural logarithm function ( $\ln(1 + z) \approx z$ ) as follows,

$$t_{isi} = \frac{\tau_m (V_{th} - V_{res})}{E_l + R_m I - V_{th}} \quad (1.7)$$

Solving for  $I$ , an input current as a function of the inter-spike interval can be obtained,

$$I = \frac{\tau_m (V_{th} - V_{res})}{t_{isi} R_m} + \frac{V_{th} - E_l}{R_m} \quad (1.8)$$

The maximum value for the input current makes the neuron fire at each sample time ( $\Delta t$ ). Therefore, the maximum input current is,

$$I^{max} = \frac{\tau_m (V_{th} - V_{res})}{\Delta t R_m} + \frac{V_{th} - E_l}{R_m} \quad (1.9)$$

In this section, we obtained the minimum and maximum values for input current using (1.5) and (1.9). These equations are used in the learning and encoding processes of the SNN.

### 1.1.2 Learning Approaches in SNNs

#### Hebbian Learning

Hebbian Learning is a fundamental neural learning principle summarized by the axiom “neurons that fire together, wire together,” describing how simultaneous activation of neurons leads to strengthened connections between them [13]. Hebbian learning, particularly within the context of SNN, primarily revolves around the modulation of synaptic strengths based on the firing rates of pre- and postsynaptic neurons. The principles of locality and joint activity are fundamental, emphasizing that synaptic changes occur only when both neurons are active simultaneously.

##### Methods of Synaptic Modification:

- **Local Rules:** Synaptic changes are influenced directly by the activities of the connecting neurons without external influences.
- **Bounded Growth:** To avoid uncontrolled increases in synaptic strength, models typically incorporate mechanisms such as hard and soft bounds. Hard bounds prevent any further increase once a maximum weight is achieved, while soft bounds slow down the rate of increase as the maximum is approached [14].
- **Synaptic Decay:** Realistic models also consider mechanisms for reducing synaptic strengths, typically through a decay term that weakens connections in the absence of activity [15].

##### Advanced Hebbian Models:

- **Covariance Rule:** This model refines the synaptic modification to depend on the deviation of firing rates from their means, enhancing the dynamic response of synapses to changes in neural activity [16].

- **Oja's Rule:** A self-stabilizing rule that ensures synaptic weights do not grow indefinitely by normalizing the weight vector, thereby maintaining the overall stability of the network [17].
- **BCM Rule:** The Bienenstock-Cooper-Munro rule introduces an adaptive threshold for synaptic modification, which evolves based on the historical activity of the neuron, allowing for more refined potentiation and depression based on relative activity levels. [18].
- **RCHP:** Rarely Correlating Hebbian Plasticity focuses on synaptic changes driven by rare, significant coincidences in neuronal activity, aiming to strengthen connections that are crucial for neural function while avoiding over-strengthening due to common activity patterns [19].

**Incorporation into Reinforcement Learning:** Modern adaptations in SNNs integrate the concept of rewards, adding a third dimension to synaptic adjustments. This integration uses scaling or gating mechanisms in response to global reward signals, further refining the learning capabilities of neural networks based on external feedback.

## Neo-Hebbian Learning and Modulation Mechanisms

In neo-Hebbian reinforcement learning, significant advancements come from a global reward signal modulating synaptic plasticity alongside an eligibility trace that decays over time [20]. This trace increases with recent, successful neurotransmission and decreases as time passes, linking closely with Temporal-Difference (TD) learning mechanisms. Such dynamics allow for Long-Term Potentiation (LTP) or Depression (LTD) at synapses based on the timing of synaptic activity and the nature of the reward signal. Specifically, recent successful activities followed by positive rewards enhance LTP, while activities preceding negative rewards lead to LTD.

### Distal rewards and credit assignment

This model introduces a sophisticated approach to synaptic modification through the interaction of Reward-modulated Spike-Timing-Dependent Plasticity (R-STDP) with a modulatory signal reflective of reward, embodying the essence of TD learning within the realm of spiking neurons.

Central to this framework is the eligibility trace mechanism, elegantly adapted from its conventional application in TD learning to facilitate synaptic credit assignment over varying temporal extents. This adaptation allows for the dynamic modulation of synaptic strengths based on the timing and sequence of pre- and post-synaptic spikes, in conjunction with the temporal dynamics of received rewards. The eligibility trace is mathematically represented as follows [21]:

$$\frac{dC_{ji}}{dt} = -\frac{C_{ji}}{\tau_C} + \text{STDP}(t_{\text{post}} - t_{\text{pre}})\delta(t - t^{(f)}) \quad (1.10)$$

Here,  $C_{ji}(t)$  denotes the eligibility trace for the synapse between pre-synaptic neuron  $j$  and post-synaptic neuron  $i$ , evolving with a decay governed by  $\tau_C$ . The Dirac delta function,  $\delta(t - t^{(f)})$ , signifies the occurrence of a spike, serving as a pivotal factor in the temporal credit assignment process.

The STDP function can be represented as follows,

$$\text{STDP}(\tau) = \mathcal{A} \exp\left(-\frac{\tau}{\tau_s}\right) \text{ for } \tau \geq 0, (\tau = t_{\text{post}} - t_{\text{pre}}) \quad (1.11)$$

where  $\mathcal{A}$  stands as the amplitude and  $\tau_s$  acts as the time constant.

Synaptic weight updates are then guided by the interaction between the eligibility trace and the reward signal,  $R(t)$ , as captured in the following equation:

$$\frac{dw_{ji}}{dt} = C_{ji}(t) \cdot R(t) \quad (1.12)$$

where  $dw_{ji}/dt$  symbolizes the rate of change in synaptic weight, contingent upon the compounded influence of the eligibility trace and the reward signal.

## 1.2 Literature Review

### 1.2.1 Spiking Neural Network

SNNs have garnered significant attention for their ability to emulate complex neural dynamics observed in biological systems. This literature review focuses on the learning mechanisms, including STDP, and their implications in modeling biological learning behaviors.

Besides the LIF model, there are different types of SNN models. The Hodgkin-Huxley model is a foundational framework in neuroscience that meticulously describes the ionic processes critical for the initiation and propagation of action potentials in neurons [22]. The model uses differential equations to capture the dynamics of membrane potential ( $V$ ) influenced by various ion-specific currents and gating variables, which regulate ion channel states. These gating variables, which transition between 0 and 1, reflect the proportion of ion channels in different states, crucial for mimicking the action potentials' temporal complexity. In contrast, the FitzHugh-Nagumo model simplifies the complex Hodgkin-Huxley framework into a two-variable system, focusing on capturing the essential characteristics of neuronal excitability with fewer computational demands.

The Izhikevich model merges the biological realism of the Hodgkin-Huxley model with the computational simplicity of integrate-and-fire models, offering a balanced approach for simulating neuronal behavior [23]. This model is particularly noted for its ability to produce diverse firing patterns with computational efficiency, making it suitable for simulating large networks of neurons while still capturing key aspects of neuronal activity.

The learning in SNNs is based on the timing between neurons firing. The STPD process is one of the biological processes observed in real neurons. The investigation of competitive Hebbian learning through STDP reveals how the timing of spikes modulates synaptic efficacy, underscoring the significance of temporal dynamics in synaptic modification within neural circuits [24]. This highlights the critical role of spike timing in learning processes, which is essential for understanding neural adaptation.

In a significant study, the distal reward problem was tackled by linking STDP with

dopamine signaling, illustrating a novel approach to reward-based learning in the brain [21]. This study proposed a model where dopamine modulates STDP mechanisms, effectively capturing the essence of reward-based learning processes.

Another research posited that STDP emerges from fundamental learning rules governed by intracellular calcium dynamics, suggesting a deeper biophysical basis for synaptic strength regulation [25]. This challenges the traditional understanding of STDP and points towards a more comprehensive framework for neural plasticity.

An extensive review of the development and implications of STDP in neural circuits provided insights into its critical role in precise spike timing for synaptic modification, enriching our understanding of the neural basis for learning and memory [26].

A study on multiagent reinforcement learning within the Iterated Prisoner's Dilemma framework highlighted SNNs' comparable or enhanced performance against traditional models [27]. This emphasizes the critical role of spike timing in complex decision-making processes, showcasing SNNs' potential in simulating interactive and competitive environments.

The introduction of a model for parallel path planning using spiking neural activity, inspired by hippocampal navigation strategies, showcased the efficiency of SNNs in solving spatial navigation problems through biologically plausible mechanisms [28].

In a pioneering study, the implementation of probabilistic inference within networks of LIF neurons demonstrated how Bayesian networks can be transformed into a computable framework for SNNs [29]. This methodology underscores SNNs' capability to perform complex cognitive functions through probabilistic reasoning.

The performance comparison between SNNs and multilayer perceptrons in a computer-based racing game highlighted SNNs' superior adaptability and decision-making capabilities in dynamic environments, attributed to the efficient processing of real-time data through spike-based mechanisms [30].

Hierarchical Bayesian inference within SNNs introduces a learning algorithm based on synaptic plasticity, showcasing the ability of SNNs to execute complex tasks like pattern recognition through hierarchical structures [31]. This emphasizes the adaptability and com-

putational efficiency of SNNs.

BP-STDP, which approximates backpropagation using STDP, bridges the gap between SNN mechanisms and traditional neural network computational efficiency [32]. This algorithm advances learning algorithms for SNNs, making them more accessible for machine learning tasks.

Indirect and direct training methods for SNNs in end-to-end control of a lane-keeping vehicle highlight the effectiveness of both approaches in utilizing SNNs for real-world control tasks, providing insights into training SNNs for complex systems like autonomous vehicles [33].

N3-CPL, a neuroplasticity-based learning method for neuromorphic networks, focuses on cell proliferation in SNNs, enhancing learning capabilities through mechanisms inspired by biological neural network growth and adaptation [34].

Parameter optimization in an SNN model for UAV obstacle avoidance tackles the challenge of tuning SNN parameters for specific tasks using optimization techniques, emphasizing the importance of optimization in SNN application for real-time processing and decision-making [35].

A methodology for multi-task autonomous learning in mobile robots using SNNs was proposed, employing Modified Integrate-and-Fire and LIF neuron models, alongside a task switch mechanism inspired by lateral inhibition and a novel learning rule based on R-STDP. This approach demonstrated SNNs' capabilities in efficiently learning and adapting across various tasks such as obstacle avoidance and target tracking [36].

An autonomous learning framework that combines SNNs with a pre-trained binary Convolutional Neural Network for image-based reinforcement learning was developed. This hybrid model efficiently processes high-dimensional sensory data and learns from sparse rewards, highlighting the synergistic potential of SNNs and CNNs in complex visual environments [37].

A transfer learning algorithm for SNNs aimed at deep learning tasks was introduced, utilizing Centered Kernel Alignment for domain distance measurement and focusing on domain-

invariant representation. This methodology enables effective knowledge transfer, showcasing the scalability and adaptability of SNNs [38].

Mathematical constraints influencing Hebbian learning and STDP in SNNs were explored, revealing relationships between synaptic weight promotion/demotion probabilities and normalized weights, offering insights into optimizing SNN training [39].

“Spikepropamine,” incorporating differentiable plasticity within SNNs, was presented. This approach enables adaptive learning and improved task performance, signifying advancements in neuromorphic computing and robotics applications of SNNs [40].

NeuronGPU, a library for simulating large-scale networks of spiking neurons with GPU acceleration, was developed. This tool demonstrated scalability and efficiency in simulating complex neuronal dynamics, marking a significant contribution to computational neuroscience tools [41].

### 1.2.2 Federated Learning

FL aggregation methods form a cornerstone of how distributed models learn collectively while maintaining the privacy of local data. The seminal aggregation method in FL is FedAvg, or Federated Averaging. As introduced by McMahan et al. [42], FedAvg operates by averaging the weights of models updated locally by clients. This simple yet effective approach allows a global model to benefit from diverse local updates without sharing the data itself.

Addressing heterogeneity in client data distributions, the FedProx method [43], introduced by Li et al., modifies the local optimization process. FedProx introduces a proximal term which moderates the deviation of local updates from the global model, aiming to stabilize training across clients with non-IID data.

The Scaffold method [44] deals with statistical heterogeneity and client drift, issues prevalent in scenarios where client data is not Independent and Identically Distributed (IID). Karimireddy et al. propose using control variates that correct the local updates toward the global objective, which helps in reducing variance in updates and aligning clients closer to the global model.

Introduced by Wang et al., the FedNova method [45] employs a novel normalization mechanism in the aggregation process that adjusts based on the number of local updates performed by each client. This design accommodates the diversity in client update contributions, particularly beneficial in non-IID data scenarios.

The MOON method, or Model-contrastive Federated Learning, minimizes the contrastive loss between local and global models to maintain consistency across updates [46]. By focusing on minimizing differences in model parameters using a similarity metric, MOON seeks to preserve the quality of the global model despite the diversity of local datasets.

Each of these methods represents an evolution in FL, introducing unique adjustments to the aggregation process to enhance convergence, reduce communication overhead, and improve robustness against adversarial clients and non-IID data conditions.

Building on the foundational principles of FL methods detailed previously, we now turn our focus towards specialized neural network architectures that leverage these FL strategies under unique operational constraints. A particularly interesting area is the utilization of SNNs, which offer a biologically inspired alternative to traditional Artificial Neural Networks (ANNs). SNNs are well-suited for on-device learning in edge computing scenarios due to their efficient energy consumption and capability for processing temporal information. However, the deployment of SNNs in such environments is not without challenges, primarily due to the limited data available on individual devices, which could significantly constrain the learning capabilities.

The limitations imposed by on-device training of SNNs, due to the constrained data availability on each device, is studied in [2]. Based on the results, the limitation can be mitigated through cooperative training utilizing FL. An online FL-based learning rule, termed FL-SNN, for networked on-device SNNs is introduced. Through this scheme, local feedback signals are utilized within each SNN instead of backpropagation, and global feedback is facilitated through communication via a base station, showcasing the potential to significantly enhance training over isolated approaches by enabling a flexible compromise between communication load and accuracy through selective synaptic weight exchange.

A novel approach of selective model aggregation in FL within the context of Vehicular EDGE Computing (VEC) is explored in [47]. This approach focuses on the unique challenges posed by the diversity in image quality and computational capacity of vehicular clients for image classification. The approach, which selectively aggregates ‘fine’ local DNN models based on an evaluation of local image quality and computation capability without compromising client privacy, employs two-dimensional contract theory for model selection due to the central server’s unawareness of these local parameters. This methodology is shown to outperform conventional federated averaging in terms of accuracy and efficiency on the MNIST and BelgiumTSC datasets, also offering higher utility at the central server.

The paper by [48] discusses a novel approach for fast global model aggregation in FL via cloud computation, aiming to address the primary bottleneck of limited communication bandwidth in the aggregation of locally computed updates across devices with strict latency and privacy requirements, like drones and smart vehicles. This approach, employing a sparse and low-rank optimization problem solution through a Difference-of-Convex-functions algorithm for joint device selection and beamforming design, demonstrates significant algorithmic advantages and performance improvements in numerical results.

According to [49], FL is suggested as a solution to the challenges faced in training machine learning models on IoT data due to constraints in network bandwidth, storage, and privacy. The focus is on how existing work has primarily centered on learning algorithms’ convergence time, leaving issues like incentive mechanisms largely unexplored. A deep reinforcement learning-based incentive mechanism is proposed to motivate edge nodes towards model training contribution, with its efficiency validated through numerical experiments.

In [4], the exploration of synergies between wireless communications and artificial intelligence, particularly the challenges of implementing ML models on battery-powered devices connected via bandwidth-constrained channels, is discussed. The paper highlights how FL for distributed training of SNNs and the integration of neuromorphic sensing with SNNs can help overcome these challenges.

The research presented in [1] introduces a FL method designed for training decentral-

ized and privacy-preserving SNNs, focusing on the energy efficiency advantages of SNNs in FL scenarios across energy-constrained devices. The method's effectiveness is experimentally validated through improved accuracy and energy efficiency in comparison to ANNs on CIFAR10 and CIFAR100 benchmarks.

A comprehensive study on deploying distributed learning over wireless edge networks, highlighting the challenges and emerging paradigms like FL and distributed inference, is provided in [50]. It presents an overview of communication techniques for efficient deployment and future research opportunities within wireless networks.

The effect of user selection and resource allocation on the convergence time and accuracy of FL over wireless networks was investigated in [51]. A probabilistic user selection scheme is proposed alongside the use of ANNs to estimate unselected users' local FL models, showing a reduction in convergence time and improvement in model accuracy through simulations.

Federated deep learning approaches for cybersecurity in IoT applications are reviewed and experimentally analyzed in [52]. The article highlights the superiority of FL in preserving IoT device data privacy and enhancing attack detection accuracy across various deep learning models.

The challenge of training FL algorithms over wireless networks is studied in [53], where the impact of wireless factors on training quality and the optimization of learning, wireless resource allocation, and user selection to minimize FL loss function are discussed. Simulation results demonstrate the effectiveness of the proposed framework in improving identification accuracy.

In [54], an investigation is conducted into the allocation of energy-efficient transmission and computation resources for FL over wireless communication networks. The model considered involves users training local FL models with their data and sending these to a base station (BS), which then aggregates these models and broadcasts the aggregated model back to the users. The optimization problem formulated aims to minimize total energy consumption under a latency constraint, with an iterative algorithm proposed that derives closed-form solutions for various parameters at every step. Initial feasible solutions are

generated through a bisection-based algorithm aimed at minimizing completion time. It is shown through numerical results that energy consumption can be reduced by up to 59.5% compared to conventional FL methods.

In [9], a lead federated neuromorphic learning (LFNL) technique is proposed, designed as a decentralized, energy-efficient, brain-inspired computing method utilizing SNNs. This technique allows edge devices to collaboratively train a global model, preserving privacy and significantly reducing both data traffic and computational latency, with only a slight accuracy loss reported. The efficacy of LFNL in reducing energy consumption while maintaining competitive recognition accuracy, despite uneven dataset distribution, is demonstrated through experimental results.

The aggregation strategies in FL through a comprehensive mathematical convergence analysis, leading to the derivation of novel aggregation algorithms that adapt model architecture by differentiating client contributions based on loss values, is presented in [55]. This approach, which extends beyond theoretical assumptions to practical performance evaluations, is compared with the conventional FedAvg method across both IID and non-IID settings without additional hypotheses.

An edge-based backhaul (BH) selection technique aimed at improving traffic delivery by leveraging multiobjective feedback, utilizing advantage-actor-critic deep reinforcement learning methods, is introduced in [56]. To enhance DRL training performance in large-scale IoT system deployments, FL is applied, enabling collaboration among multiple edge devices. The effectiveness of this federated DRL approach in solving the BH selection problem is verified through extensive simulations.

In [57], a hierarchical trajectory planning method named HALOES, which employs deep reinforcement learning within a FL scheme, is proposed for efficient, automatic parking in narrow spaces. This method combines high-level deep reinforcement learning with low-level optimization, improving planning time and model generalization capabilities while protecting data privacy during model parameter aggregation. Simulation results affirm HALOES's efficiency in various narrow-space parking scenarios.

The hierarchical FL paradigm of HiFL is discussed in [58], which aims to mitigate communication overhead in FL systems by integrating synchronous client-edge and asynchronous edge-cloud model aggregations. An enhanced design, HiFlash, incorporates adaptive staleness control and a heterogeneity-aware client-edge association strategy through deep reinforcement learning, demonstrating improved system efficiency, communication reduction, and maintained model accuracy in experiments.

A survey on poisoning attacks and defense strategies in FL from a privacy-preserving perspective, classifying and analyzing poisoning attacks and defense mechanisms, is provided in [59]. The paper highlights the urgency of defending against poisoning attacks and suggests potential research directions for both attack and defense strategies, emphasizing the need for systematic reviews in this area.

# Chapter 2

## Modular Learning in SNNs for Optimal Multi-Agent Decision-Making

### 2.1 Introduction

This chapter looks at two main things: how well SNNs handle noise and how they can handle complex scenarios like optimal decision-making. This detailed study shows us the potential of SNNs in managing complex behaviors even when there are outside disturbances and their capability to separate the reward function for the neural network's different parts.

The neural structure in SNNs helps us implement complex learning systems. One good example of a complex situation is a differential game like the Active Target Defense (ATD) problem [60].

In the ATD, a defender tries to protect a target, while a superior invader with higher velocity than the defender and a moving target tries to reach the target and escape the defender at the same time. In the context of defense strategy, aircraft movement analysis is crucial in determining the most effective strategies. In the context of defense strategy, aircraft movement analysis is crucial in determining the most effective strategies. Law enforcement benefits from these games by minimizing escape possibilities. The ATD problems are a fundamental component of game theory, providing insights into strategic decision-making

across disciplines like economics and biology. Additionally, these games find applications in Cybersecurity for modeling attacker-defender interactions [61].

There are two distinct outcomes to the ATD game, characterized by two termination sets [62]. The first outcome happens when the invader reaches the target while the defender is far from it. The second outcome happens when the defender reaches the target while the invader's distance from the target is larger than the defender's distance.

## 2.2 The ATD problem and SNN-based solution

The ATD problem has various solving methods, such as Apollonius Circle and Cartesian Ovals (CO). This chapter opts for the CO method over the Apollonius Circle because it considers the capture radius and effectiveness against superior invaders [62]. The optimal capture point is considered the minimum distance between the target's position and reachable region if the target is inside the defender's dominant region. The defender's dominant region is a region where the defender can reach the target without letting the invader capture the target.

In this chapter, the problem is solved using reinforcement learning. It is considered that each agent knows the relative velocity of other agents. Figure 2.1 shows LOS angles used as the input for the SNNs [63].

The R-STDP algorithm is used to train two separate SNNs simultaneously that control the invader and defender. The target in this chapter moves in the environment, and the SNNs receive the LOS angles (e.g., the defender receives the LOS angle to both the invader and the target) and calculate the steering angle for the agent (Figure 2.2). In this figure, the  $\phi_I^T$  represents the LOS angle to the target relative to the invader.

## 2.3 Learning using R-STDP

R-STDP is a biological learning algorithm that is believed to underlie certain learning mechanisms in the brain [64]. The R-STDP algorithm is based on the idea that if a pre-synaptic

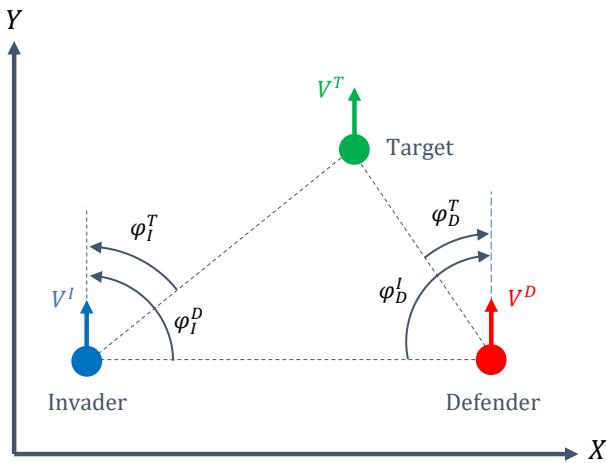


Figure 2.1: Active target defense game with three agents (LOS angles are shown for both agents).

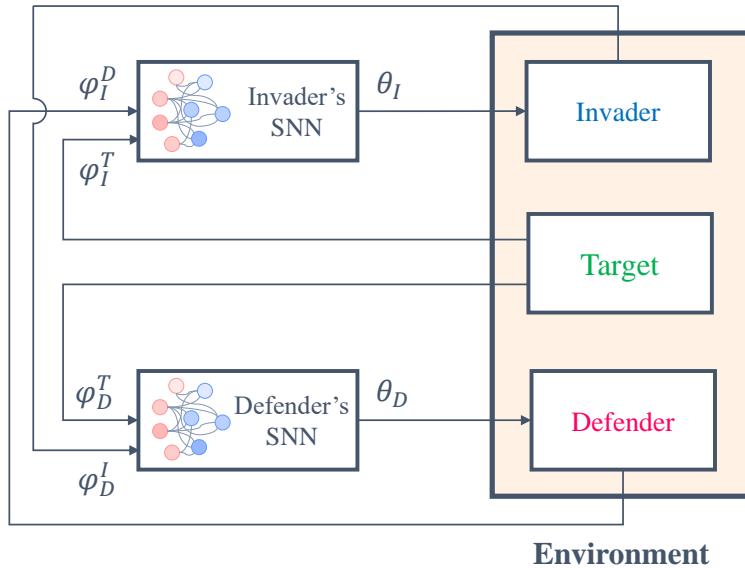


Figure 2.2: Two SNNs simultaneously play and control the invader and defender. Each agent uses the LOS angle and relative velocities for training.

neuron fires just before a post-synaptic neuron and the network receives a reward, the strength of the synapse between the two neurons should be increased.

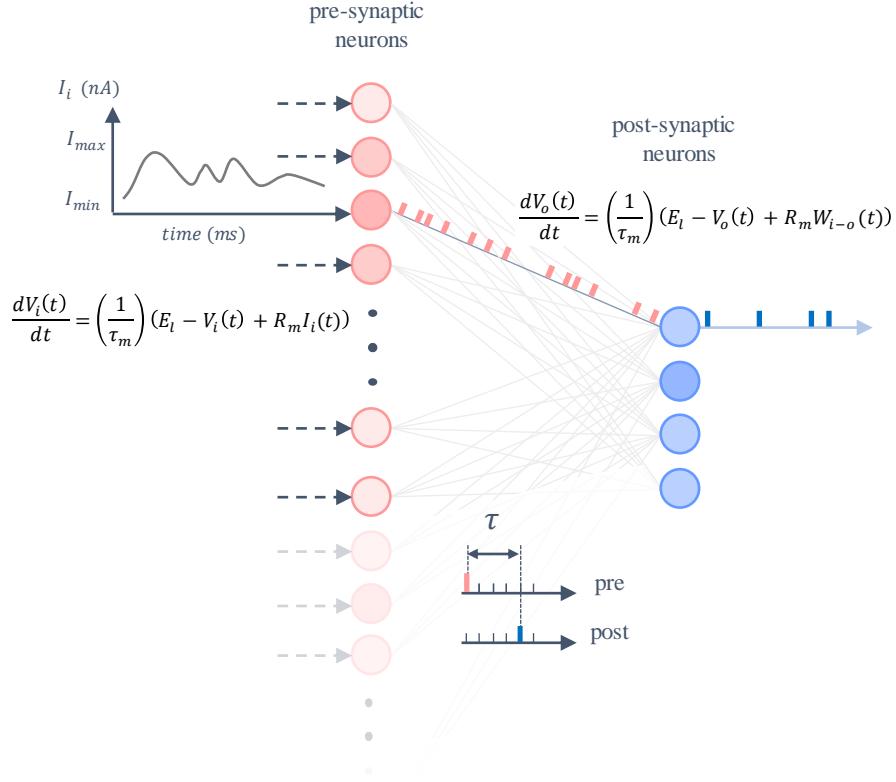


Figure 2.3: An illustration of input current to SNN and synaptic current to the output neurons after training for a single connection.

In the SNN, the pre-synaptic neurons are input neurons, and post-synaptic neurons are output neurons. The synaptic plasticity is referred to as an eligibility trace ( $C$ ), which is calculated based on the following equation [21],

$$\dot{C}^{i-j} = -C^{i-j}/\tau_C + STDP^{i-j}(\tau)\delta(t - t_{pre/post}) \quad (2.1)$$

where the  $C^{i-j}$  is the eligibility trace for the synaptic connection between neurons  $i$  and  $j$ ,  $\tau$ , is the spike timing difference between the input and output spike times,  $\tau_C$  is the time constant for the synaptic plasticity,  $\delta$  is the Dirac function,  $t_{pre/post}$  is the firing time of the pre or post-synaptic neuron, and  $STDP(\tau)$  is a function of the firing time of the input and

output neurons as follow,

$$STDP^{i-j}(\tau) = \begin{cases} A_+ \exp\left(\frac{-\tau}{\tau_s}\right) & \text{if } \tau > 0 \\ A_- \exp\left(\frac{\tau}{\tau_s}\right) & \text{if } \tau < 0 \end{cases} \quad (2.2)$$

where  $A_+$  and  $A_-$  are the amplitude of the exponential function, and  $\tau_s$  is the time constant that determines the decaying rate of the R-STDP function. If  $\tau_s$  goes to infinity, the exponential function becomes 1, and the effect of time in R-STDP will be lost.

The synaptic weights are changed according to the following equation,

$$\dot{W}^{i-j}(t) = C^{i-j}(t)R(t) \quad (2.3)$$

where  $W^{i-j}$  is the synaptic weight between neurons  $i$  and  $j$ , which is the amount of input ( $I(t)$ ) that the post-synaptic neuron receives when the pre-synaptic neuron spikes, and  $R(t)$  is the reward.

This chapter uses the Multiplicative Synaptic Normalization (MSN) method to keep runaway excitation under control. This method keeps pre-existing memories in the network by conserving the proportional difference between smaller and larger synaptic weights. According to (1.9), we can calculate the maximum input for each post-synaptic neuron (output layer).

The MSN normalizes the synaptic weights based on the cumulative input synaptic weights and the maximum input as follows [64],

$$\vec{W}^n(t) = \vec{W}^n(t-1) \left( \frac{I^{max}}{\sum_{\ell=1}^N W^\ell(t)} \right) \quad (2.4)$$

where  $\vec{W}^n(t)$  is the vector consisting of synaptic weights that send input current to the  $n^{th}$  output neuron, and  $N$ , is the total number of input synapses for each output neuron. Therefore, when R-STDP increases a single synaptic weight, the MSN proportionally decreases the other synaptic weights for the  $n^{th}$  neuron.

The reward for the invader and defender is defined based on the projection of the velocities

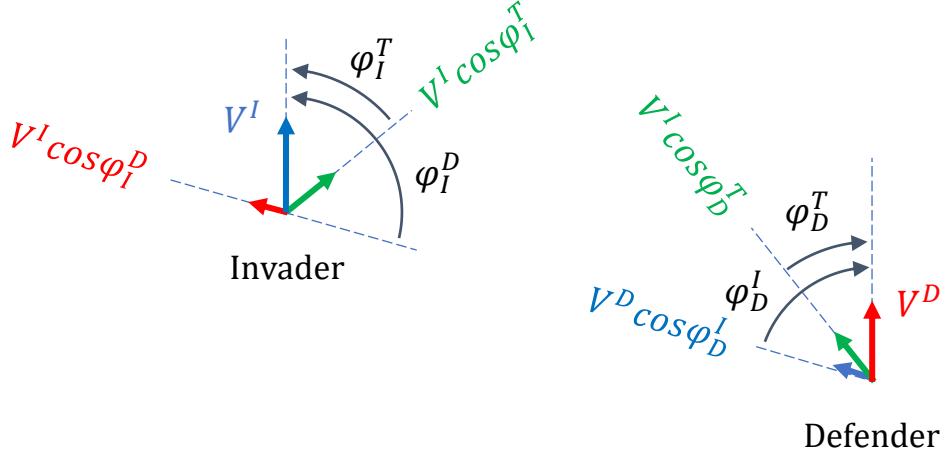


Figure 2.4: Relative velocities and LOS angles used in the reward function

along the LOS direction. Figure 2.4 shows the projected velocities for the invader and defender, where both agents measure the relative velocities from each other. It means that the invader receives a negative reward when it moves toward the defender and receives a positive reward when it moves toward the target. The defender receives a positive reward when it moves toward the target and invader. Since the velocities are constant, the reward value depends only on the headings that cause the change in relative velocities. Therefore, the LOS toward the other agents determines the reward value.

The reward for the invader considering the target and defender consists of two parts,

$$R_I^T(t) = \eta_I^T (V^I + V^T) \cos(\phi_I^T) \quad (2.5)$$

and

$$R_I^D(t) = \eta_I^D (V^D - V^I) \cos(\phi_I^D) \quad (2.6)$$

The reward for the defender also can be calculated as follows,

$$R_D^T(t) = \eta_D^T (V^D + V^T) \cos(\phi_D^T) \quad (2.7)$$

and

$$R_D^I(t) = \eta_I^T (V^D - V^I) \cos(\phi_D^I) \quad (2.8)$$

In (2.5)-(2.8),  $\eta_I^T$ ,  $\eta_I^D$ ,  $\eta_D^T$ , and  $\eta_D^I$  are constant coefficients. Adjusting these values changes the agent's attention to other agents. For example, in the invader case, increasing the  $\eta_I^T$  and decreasing the  $\eta_I^D$  increases the effect of the target on the output and reduces the defender's effect. The invader then places more importance on getting to the target than evading the defender.

The change in synaptic weights for the invader regarding the target and defender consists of two parts as follows (the same process is true for the defender),

$$\mathbf{W}_I^T(t) = \mathbf{W}_I^T(t-1) + \mathbf{C}_I^T(t)R_I^T(t) \quad (2.9)$$

and

$$\mathbf{W}_I^D(t) = \mathbf{W}_I^D(t-1) + \mathbf{C}_I^D(t)R_I^D(t) \quad (2.10)$$

where  $\mathbf{W}_I^T$  is a  $k \times l$  matrix that represents the synaptic weights corresponding to the target ( $k$  is the number of output neurons and  $l$  is the number of input neurons for the target). The  $\mathbf{W}_I^D$  is a  $k \times m$  matrix that represents the synaptic weights regarding the defender ( $m$  is the number of input neurons for the defender). The eligibility trace matrices  $\mathbf{C}_I^T$  and  $\mathbf{C}_I^D$  are of dimension  $k \times l$  and  $k \times m$ , respectively.

## 2.4 Network structure and encoding method

Figure 2.5 shows the defender's network structure and encoding process. The invader has the same network with different inputs. The network receives the LOS angles and converts the inputs to Fuzzy Membership Values (FMV) using Gaussian Receptive Fields (GRF) [65]. Since the fuzzy membership values are used just for encoding data into the SNN, the type of the membership function does not affect the computation complexity.

There are  $q$  input neurons, and a membership function is assigned to each neuron. There-

fore, there are  $q$  membership functions. The acquired fuzzy membership values are real numbers between 0 and 1 and should be converted to the input currents. This can be done using a linear function and the minimum and maximum inputs.

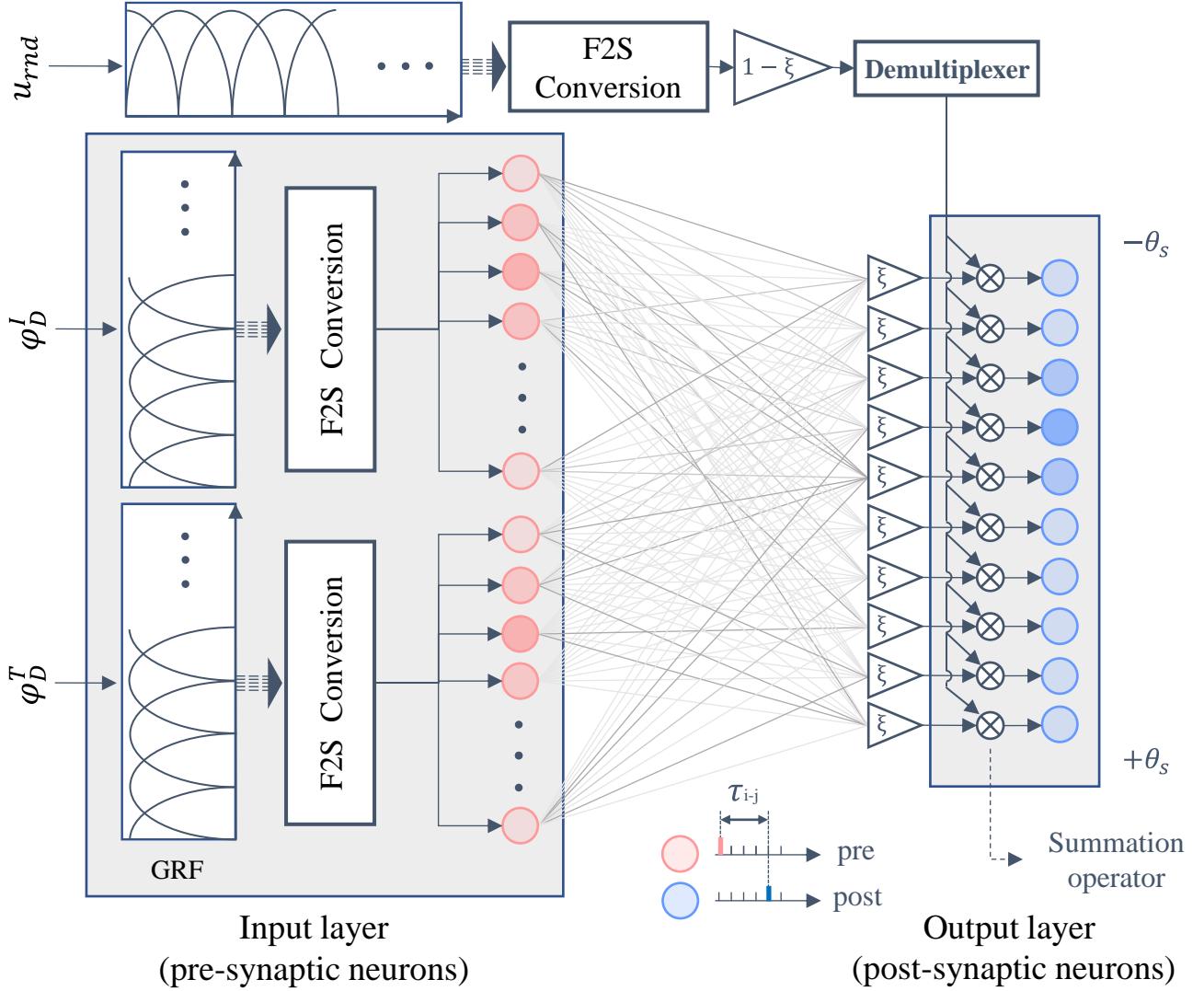


Figure 2.5: Network structure and encoding process for the input layer (Defender). Each neuron is associated with a membership function in GRF. The GRF encodes an input State ( $S^t$ ) at each time step. There is both a training phase when ( $\xi = 0$ ) and an operating phase when ( $\xi = 1$ ).

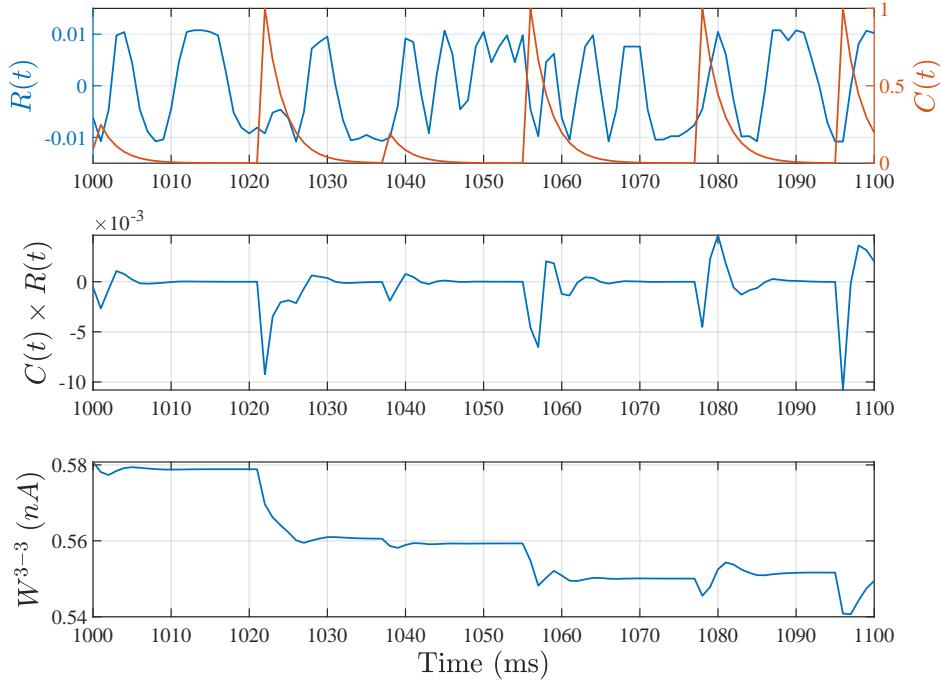


Figure 2.6: Reward, eligibility trace, and weight change during the simulation for  $W^{3-3}$ . The  $C(t)R(t)$  changes the synaptic weight by considering activation strength and reward value.

The acquired fuzzy membership values are converted to the spiking inputs for the neurons using fuzzy-to-spiking (F2S) conversion. If  $FMV = 0$  ( $t_{isi} = \infty$ ), then the input to the desired neuron in the input layer is  $I^{min}$ , and if  $FMV = 1$  ( $t_{isi} = \Delta t$ ,  $\Delta t$  is the sampling time), then the input is  $I^{max}$ . Therefore, the input current for the input neurons can be calculated using the following equation,

$$I_\sigma = (I^{max} - I^{min}) FMV_\sigma + I^{min}$$

or

$$I_\sigma = \frac{\tau_m (V_{th} - V_{res})}{\Delta t R_m} FMV_\sigma + \frac{V_{th} - E_l}{R_m} \quad (2.11)$$

where  $\sigma$  is the index of each neuron in the input layer and its corresponding fuzzy membership value in GRF.

**Algorithm 1** Weight training algorithm

---

```

1: for  $t = 0 : \Delta t : t_{final}$  do
2:   input  $\phi_I^T$  and  $\phi_I^D$ 
3:    $\zeta \leftarrow$  number of input neurons
4:    $\kappa \leftarrow$  number of output neurons
5:   for  $i = 0:2\pi/\zeta:2\pi$  do
6:      $Z_T(i, 1) = \exp(-0.5(\phi_I^T - i)/\sigma)$ 
7:      $Z_D(i, 1) = \exp(-0.5(\phi_I^D - i)/\sigma)$ 
8:   end for
9:    $I^T = (I^{max} - I^{min}) Z_T + I^{min}$ 
10:   $I^D = (I^{max} - I^{min}) Z_D + I^{min}$ 
11:  Generate a uniform random number for exploration  $u_{rnd} \in [-\theta_s, \theta_s]$ 
12:   $j = 1$ 
13:  for  $i = -\theta_s:2\theta_s/\kappa:\theta_s$  do
14:     $Z_{rnd}(j, 1) = \exp(-0.5(u_{rnd} - i)/\sigma)$ 
15:     $j \leftarrow j + 1$ 
16:  end for
17:   $I_{rnd} = (I^{max} - I^{min}) Z_{rnd} + I^{min}$ 
18:  for  $j = 1$  to  $\kappa$  do
19:     $I_{syn}(j, 1) = \sum_{i=1}^{\zeta} W^{i-j} \delta(t - t_i)$        $\triangleright t_i$  is the firing time of the  $i^{th}$  input neuron
20:  end for
21:   $I_{in} = \begin{bmatrix} I^T \\ I^D \end{bmatrix}$  Input current
22:   $I_{out} = \xi I_{syn} + (1 - \xi) I_{rnd}$  ( $\xi = 0$  in training phase)
23:   $\mathbf{V}_m(t + \Delta t) = (1 - \frac{\Delta t}{\tau_m}) \mathbf{V}_m(t) + \frac{\Delta t}{\tau_m} \left( \mathbf{E}_l + R_m \begin{bmatrix} I_{in} \\ I_{out} \end{bmatrix} \right)$   $\{\mathbf{V}_m \text{ and } \mathbf{E}_l \text{ are } [\zeta + \kappa] \times 1\}$ 
24:  Find fired neurons in input and output layer
25:  Calculate the  $\boldsymbol{\tau}$  matrix that shows the difference in firing time between the fired
   input neurons and fired output neurons (Figure 2.5)
26:  Calculate  $\mathbf{R} - \mathbf{STDP}$  for all connection using (2.2)
27:  Calculate  $\mathbf{C}$  matrix using (2.1) for all the connections
28:  Calculate  $\mathbf{W}$  matrix using (2.3) for all the connections considering reward from (2.5)
   to (2.8)
29:  for  $i = 1$  to  $\kappa$  do
30:    if sum of the input weights to  $i^{th}$  neuron  $\geq I^{max}$  then
31:      Normalize input weights of  $i^{th}$  neuron using (2.4)
32:    end if
33:  end for
34:  Set voltage of the fired neurons to reset voltage ( $V_{res}$ )
35: end for

```

---

The output of the SNN is the steering angle ( $\theta_s$ ) for the agent. The output is calculated using the weighted average method. Each neuron in the output layer represents a specific steering angle. The number of spikes for each output neuron in  $\tau_s$  millisecond represents how much it contributes to the output. The contribution level is considered 1 for an output neuron that fires at each step time, while it is considered 0 for the output neuron that has not fired. Levels of contributions are then multiplied by the angle that each output neuron represents. Finally, the summation of all the calculated terms is divided by the summation of all levels of contributions.

The output consists of two terms. One term comes from the synaptic weights, and the other term is random noise for exploration. The output of the SNN can be shown as follows,

$$I_{out} = \xi I_{syn} + (1 - \xi) I_{rnd} \quad (2.12)$$

where  $\xi$  is a constant that is 0 during training and becomes 1 after training is completed,  $I_{rnd}$  is a random steering angle that is selected at each time step, and  $I_{syn}$  is the synaptic output (steering angle based on synaptic weights). Therefore, there are two phases: a training phase and an operating phase.

During training, the input State ( $S^t$ ) is encoded into the network, and a random steering angle ( $u_{rnd}$ ) is encoded as a random action using the F2S process into the output layer. These two encoding currents for the input and output layer make input and output neurons fire independently. The R-STDP adapts the weights for the fired neurons. Since  $\xi$  is 0 during the training, the  $I_{syn}$  does not affect the SNN's output (Equation 2.12).

In the training process, the output and input neurons are excited separately. The input neurons are fired based on the agent's current state, whereas the agent's steering angle is randomly assigned based on the random input to the output neurons, as shown in Figure 2.6. The agent then takes a step based on the random steering angle and a reward is assigned. The training algorithm then evaluates the reward for that given random steering angle. If the reward is positive, then the weight associated with the input neurons to output neurons that fired for that state is strengthened, and if the reward is negative, then the weight for

the input to output neurons in (2.2) is weakened. Future research will include an inhibitory effect where the weights can become negative.

After training, the  $\xi$  changes to 1 and eliminates the effect of random output, and the SNN's output is calculated based on the synaptic currents. Algorithm 1 shows the training process.

## 2.5 Results

A numerical simulation is conducted to evaluate the SNN's performance in solving the ATD problem. The simulation is done in MATLAB 2022a, with a 1 *ms* sample time. The simulation parameters for neurons are presented in Table 2.1.

Table 2.1: Parameter values for LIF neuron model [10]

Parameter	Value	Description
$R_m$	40 M $\Omega$	Membrane Resistance
$\tau_m$	30 ms	Membrane time constant
$E_l$	-70 mV	Resting potential
$V_{res}$	-70 mV	Reset potential
$V_0$	-70 mV	Initial membrane potential
$V_{th}$	-50 mV	Threshold membrane potential

Table 2.2: Parameter values for R-STDP

Parameter	Value	Description
$\tau_s$	3 ms	Time constant
$A_{\pm}$	1	Amplitude of the R-STDP function
$\eta_D^T$	0.90	Reward coefficient
$\eta_D^I$	1.10	Reward coefficient
$\eta_I^T$	1.20	Reward coefficient
$\eta_I^D$	0.80	Reward coefficient

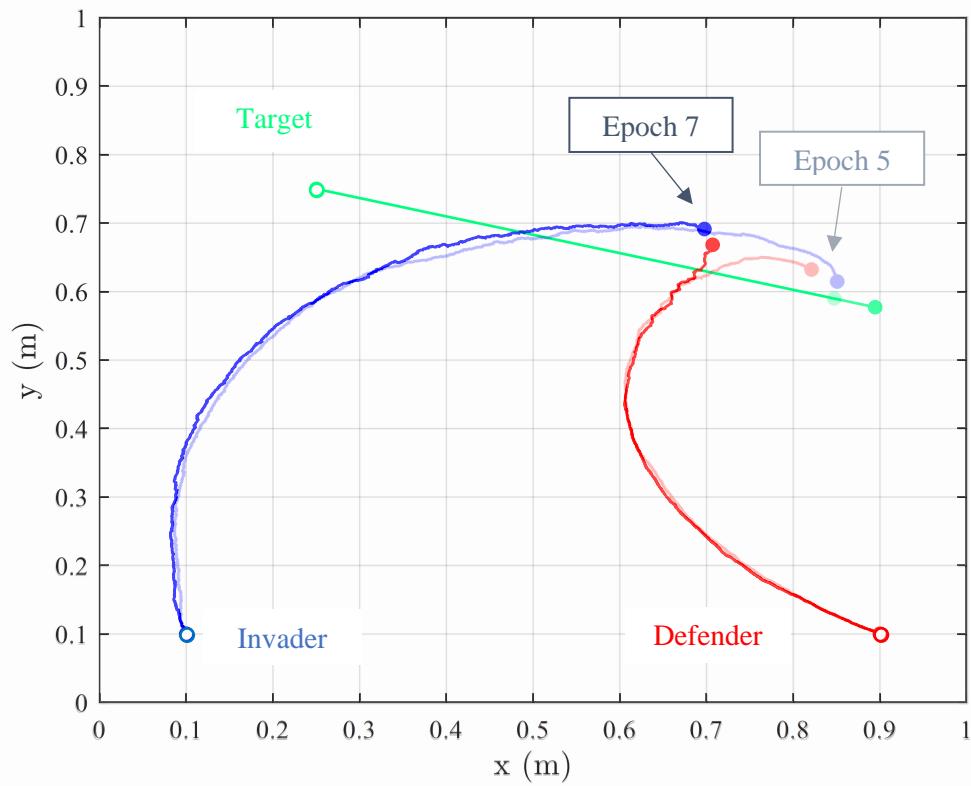


Figure 2.7: SNN's performance during training. Hollow circles show the initial positions.

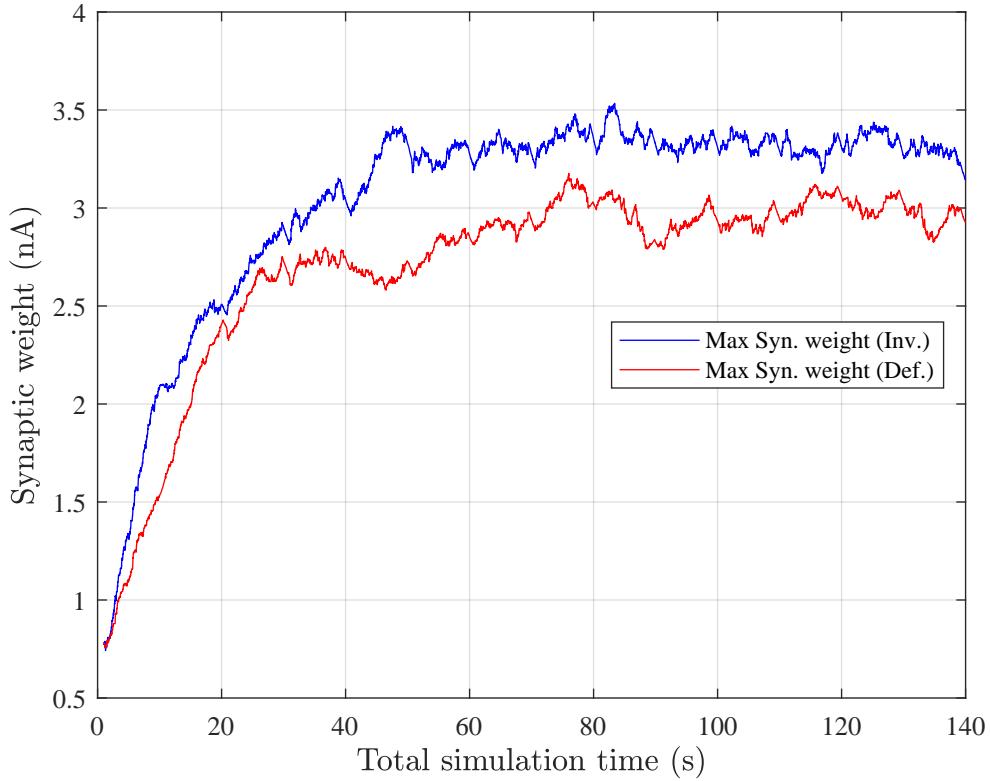


Figure 2.8: Changes in synaptic weights during training. Only Maximum synaptic weights for both agents are shown.

After several simulations, the number of input neurons for invader and defender networks was set to 20. Both agents have 10 neurons in their output layer. The encoding resolution can be enhanced by increasing the number of neurons, although this results in a higher number of synaptic connections. An optimization algorithm can be employed to determine the optimal neuron number in the SNN. Half of the input neurons for the invader are for the  $\phi_I^T$ , and the other half is for the  $\phi_I^D$ . Half of the defender's input neurons are for the  $\phi_D^T$ , and the other half is for the  $\phi_D^I$ . Since each network contains 20 input neurons, the input layer has 20 Gaussian membership functions.

The output of the activation of each membership function is the input to a neuron associated with that specific membership function. There are 10 input neurons for the 10 membership functions related to each input. Furthermore, no more than 2 membership

functions fire for any given input. Therefore, at most, only two neurons are excited and generate an impulse sequence for a given input.

The target's velocity is  $0.15 \text{ m/s}$ . The invader's velocity is  $0.3 \text{ m/s}$ . The  $\gamma$  is  $0.75$ , so the defender's velocity is  $0.225 \text{ m/s}$ . The defender's capture radius ( $\rho$ ) is set to  $0.025 \text{ m}$ . The maximum simulation time for each epoch is 10 seconds. The  $-\pi/4 < \theta_s < \pi/4$  for invader and defender, and the  $\sigma$  is set to  $1.25$ . The parameter values for R-STDP, shown in Table 2.2, are set through several simulations.

The reward coefficients are set manually. The  $\tau_s$  in Table 2.2 defines the decaying rate of  $C$  in (2.1) that determines the R-STDP sensitivity to prior firings. According to (2.2), higher  $\tau_s$  means that the R-STDP takes into account the activity of the two neurons that have fired in the relatively larger time window. Different studies have considered different values for this parameter.

As mentioned in Section 3.1, the optimal capture point is the closest point from the reachable region to the target position. Figure 2.7 shows the agents during the training process. In epoch 5, the defender is not able to capture the invader, and the invader reaches the target. In epoch 7, the defender learns how to block and capture the invader. The defender has won the game. However, the invader should learn to reach the minimum distance from the target.

### 2.5.1 Simulation without noise

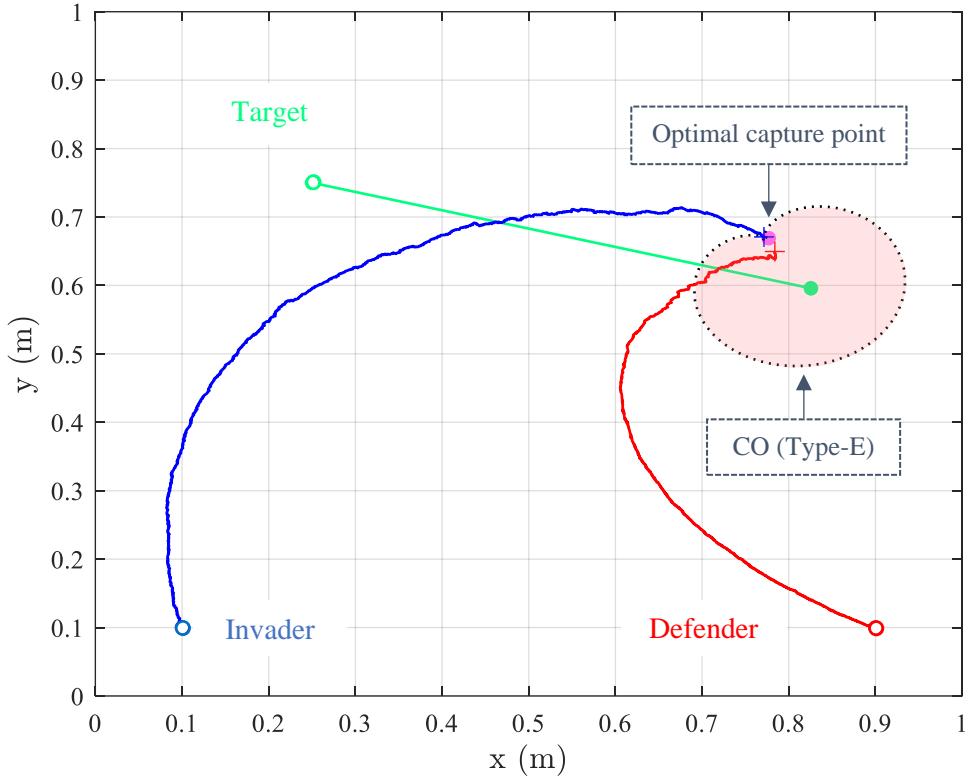


Figure 2.9: SNN’s performance after training. The highlighted region shows the defender’s dominant region. The purple dot is the optimal capture point. The CO Type-E shows the reachable regions of the Invader and the Defender.

Figure 2.9 shows the performance after training. Each epoch has a maximum time of 10 seconds. After 14 epochs, the defender learned to capture the target, while the invader learned to reduce its distance from the target. According to the CO, the target is inside the defender’s dominant region. Therefore, although the invader’s velocity is higher than the defender’s velocity, it cannot reach the target. In this situation, the optimal policy for the invader is to minimize its distance from the target.

According to Figure 2.9, the invader’s SNN can find the optimal [62] capture point for the invader, while the defender’s SNN can protect the moving target against a superior invader. It should be noted that this solution is obtained without a global reference frame. This

is important because, in real-world swarm applications, defining a global reference frame is difficult while the learning process is highly dependent on the precise definition of the coordinate system.

Figure 2.8 shows the changes in synaptic weights. Only the synaptic weights with maximum values are shown in this figure because the network has 200 synaptic connections. The minimum value is limited to zero because negative synaptic weights inhibit the post-synaptic neurons. This chapter does not consider the inhibition process. According to figure 2.8, after almost 100 seconds of simulation time, the MSN process causes the synaptic weights to converge.

### 2.5.2 Simulation with noise

In Figure 2.10, we observe the performance of two methods, namely the SNN and the Cartesian Oval (CO) method, in the presence of noise. The noise in this experiment is introduced as white Gaussian noise, characterized by a mean of zero and a variance of 0.01. Both SNN and CO receive position data that has been corrupted by this noise.

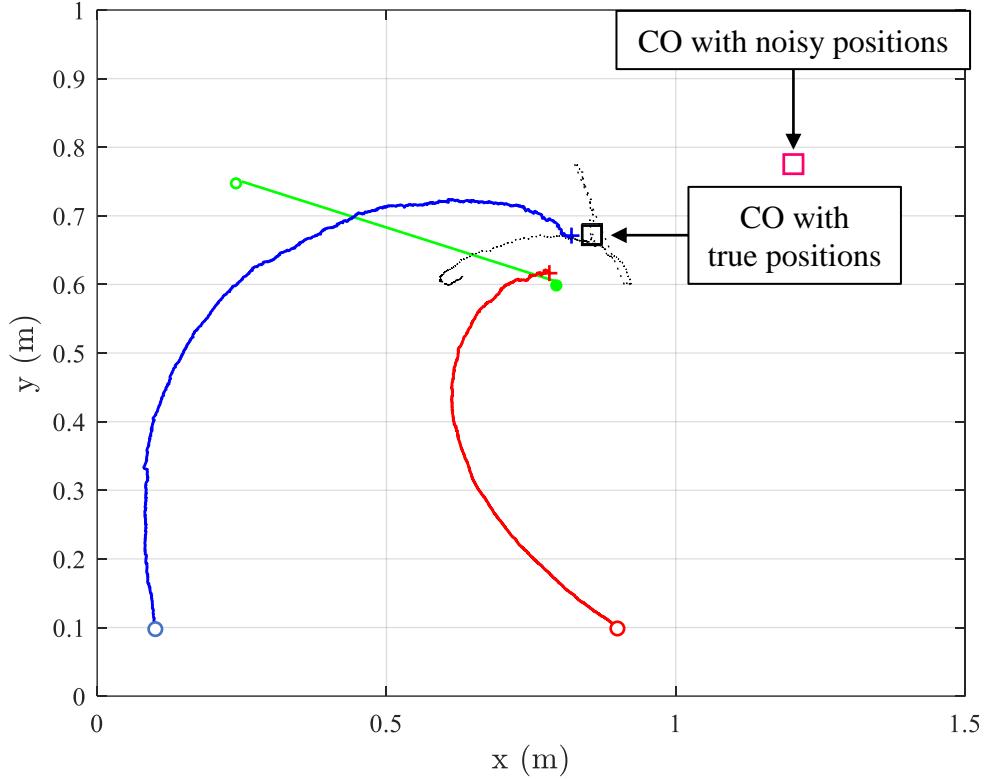


Figure 2.10: SNN’s performance in noisy conditions. A white Gaussian noise with a variance of 0.01 is added to the measured inputs.

The results of the simulation reveal that the CO method is highly affected by the presence of noise, making it unable to calculate the optimal capture point accurately. Due to its sensitivity to measurement noise, the CO method exhibits a significant deviation from the desired capture point. On the other hand, the SNN method demonstrates a higher level of robustness against noise. Despite the presence of measurement noise, the SNN method manages to achieve the optimal capture point with an error of only 0.036  $m$ . This outcome highlights the superior performance of the SNN method in noisy conditions compared to the CO method.

## 2.6 Conclusion

This chapter focused on addressing the ATD problem within a dynamic environment involving two agents, where the target is in motion. The approach involved training two SNN simultaneously to engage in a competitive game. During the game, the target transitions from the invader's dominant region to the defender's dominant region. This shift in the target's location within the defender's dominant region satisfied the necessary conditions for determining the reachable regions for both the invader and defender.

To evaluate the effectiveness of the SNN's solution, a comparison was made with Cartesian Oval designed for centralized problems. The results demonstrated that the SNN method was capable of identifying the optimal solution for decentralized problems, even under the presence of noise. This result holds significant practical implications, particularly in scenarios where establishing a global coordinate system for all agents proves to be challenging. The obtained solution provided by the SNN approach offers a valuable alternative in such cases, showcasing its potential in real-world applications.

# Chapter 3

## Integration of R-STDP and Federated Learning

### 3.1 Consensus Flying Problem

This chapter studies the cooperation between follower drones to follow the leader drone by integrating R-STDP and FL. The cooperation problem is formation flying or “Consensus Flying”. The consensus flying problem deals with ensuring drones can work together in real-time to agree on their flight paths and positions. When many drones are close together, like in swarms, avoiding crashes is vital. Advanced algorithms and communication methods are needed so drones can exchange information and handle changing situations and unexpected obstacles.

As shown in Figure 3.1, a swarm of agents (follower drones) flies around a leader. The leader is controlled from a remote base station, and the swarm agents should learn to fly safely with the leader. The leader sends its position to all agents, and each agent only sees two neighboring agents. The swarm aims to learn how to keep a commanded distance from each other and the leader. The commanded distance is provided from the leader. Each agent uses the onboard sensors to find the distance and line of sight to neighboring agents.

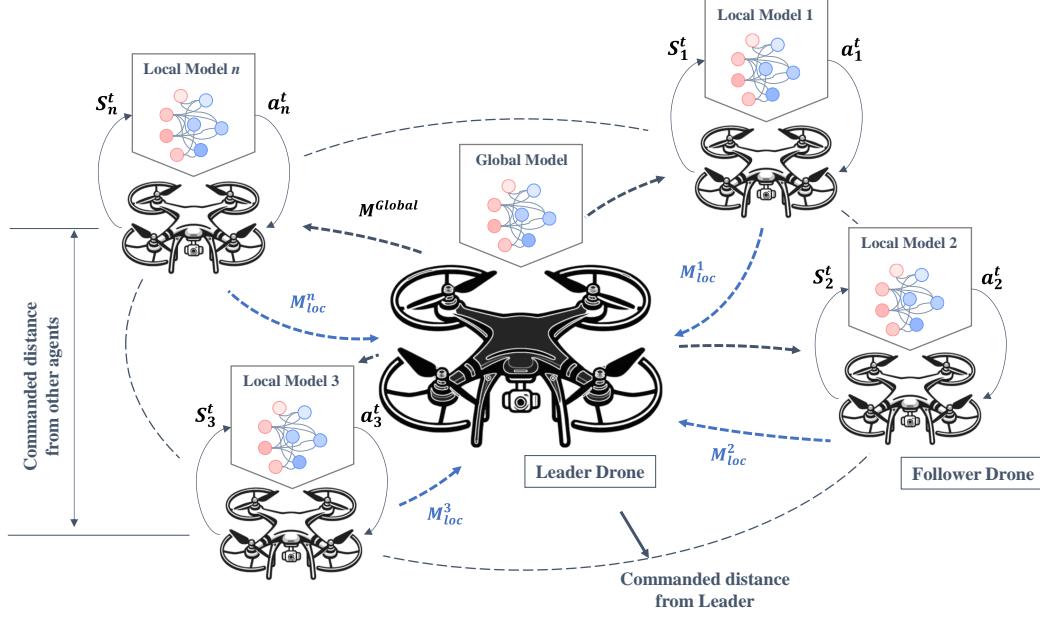


Figure 3.1: The central server (the leader) and the surrounding follower agents (white drones). The follower agents learn to fly in a formation to maintain the commanded distance. The Local models trained individually by follower agents are sent to the leader. The leader aggregates the models and sends back the global model for another round of training of the follower agents.

The follower agents are equipped with an SNN, and their learning algorithm incorporates R-STDP and FL. Each follower agent trains a local network ( $M_{loc}^n$ ) using R-STDP and sends its model to the leader as the central server. The leader aggregates models and sends back the global model ( $M^{Global}$ ).

This chapter employs the SNN model to train a group of swarm agents that follow a leader. Each agent has its own SNN, which is trained independently using the R-STDP algorithm. Each agent receives position data from the agents nearby. The goal is for each agent to keep a commanded distance from the leader agent and the other agents in the group. The encoding and decoding processes for the input and output layers of the SNN are fuzzy encoding, and a novel method is introduced to stabilize the network dynamics considering the reward function. This work presents several key contributions:

- The chapter presents a comprehensive method for stabilizing and enhancing the learning process in SNN. This method focuses on controlling the unbounded growth of synaptic weights in SNNs, utilizing a strategy that dynamically adapts to changes in reward conditions and coefficients. It introduces a decay rate and learning rate adjustment based on the status of synaptic weights and enhances the responsiveness of the SNN weights to reward change.
- In terms of advancements in FL with R-STDP, the chapter addresses the FL challenges in the R-STDP framework. It introduces an event-triggered mechanism for model publishing and receiving within the network, improving network traffic. Additionally, the chapter implements a novel weighted aggregation method on the server. This method calculates weights based on the time of arrival of the models, effectively tackling the asynchronous issues in FL.

## 3.2 Proposed Method

### 3.2.1 Network Structure

This chapter assumes that each agent detects only two neighboring agents in addition to the leader. The information obtained from other agents includes the Line-of-Sight (LOS) angle and the distance. Each agent's neural network consists of three sub-layers in the input layer, as shown in Figure 3.2. Two sub-layers correspond to the two neighboring follower agents ( $F_1$  and  $F_2$ ), and the third is dedicated to the leader ( $L$ ). Inputs for these sub-layers are encoded using the Gaussian Receptive Fields (GRF) that use fuzzy membership functions. The network uses the difference between current and commanded distances within the swarm ( $r_{cmd}$ ) and between followers and the leader ( $R_{cmd}$ ) to stimulate input neurons.

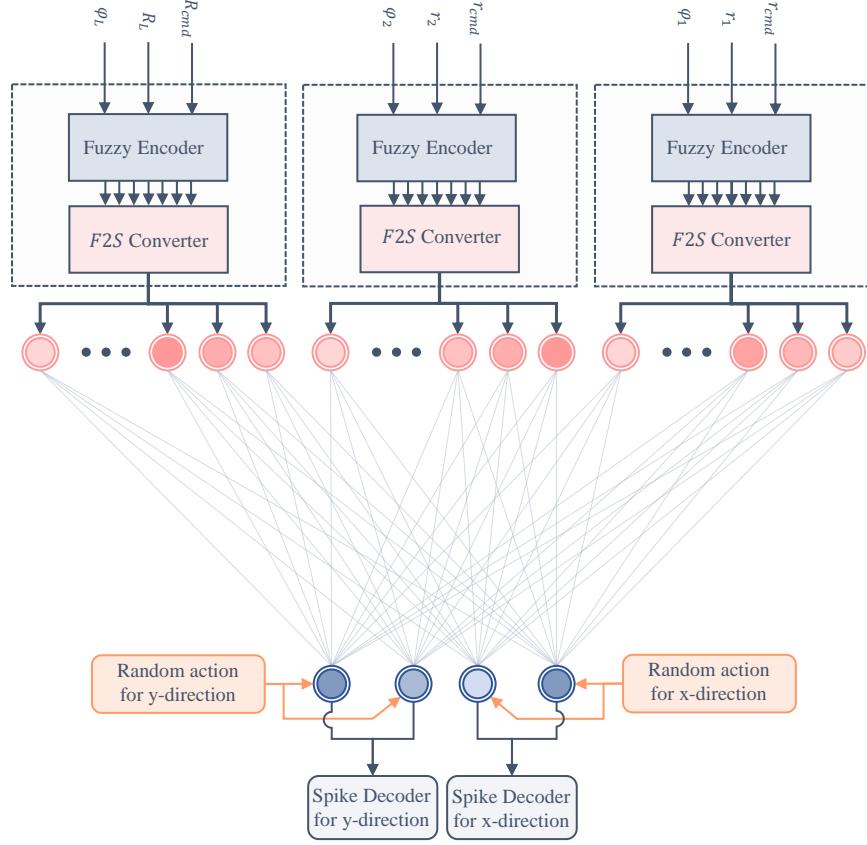


Figure 3.2: SNN structure with encoding and decoding layers. Each sub-layer consists of a fuzzy encoder and the F2S Converter, with the output layer receiving inputs from synaptic weights and a random action selector. During the training phase, the output layer receives input only from the random action selector, which then shifts to synaptic weight inputs after the training (test phase).

Every input sub-layer is split into two parts. The first part deals with distances greater than the commanded distance, while the second focuses on the space between the agent and the commanded distance. Within each part, the LOS angle is encoded with fuzzy membership functions. The difference between the current and commanded distance is represented as the error. We transform this difference into an amplitude value using the  $tanh$  function so that it is bounded between 0 and 1. An error of zero leads to an amplitude of zero, and as the error increases towards infinity, the amplitude approaches one. Consequently, the

encoding function for the input layer is expressed as follows,

$$\mu_I(\phi_i, r_i) = |\tanh(r - r_i)| \cdot \exp\left(-\frac{(\phi_i - \zeta)^2}{2\sigma^2}\right) \quad (3.1)$$

where  $\zeta$  and  $\sigma$  are the Gaussian membership functions' center and standard deviation. The  $r_i$  is the distance from the corresponding agent,  $\phi_i$  is the LOS angle, and  $\mu_I$  is the vector of the membership degrees. Here,  $r$  is a placeholder that can either represent  $r_{cmd}$  or  $R_{cmd}$ , depending on the context. The firing strengths from fuzzy encoders are then converted to the spiking input based on the neuron model as follows [66],

$$I_{sub-layer} = (I^{max} - I^{min}) \mu_I(\phi_i, r_i) + I^{min} \quad (3.2)$$

or

$$I_{sub-layer} = \frac{\tau_m (V_{th} - V_{res})}{\Delta t R_m} \mu_I(\phi_i, r_i) + \frac{V_{th} - E_l}{R_m} \quad (3.3)$$

The encoding process is shown in Figure 3.3. The Fuzzy-to-Spiking (F2S) block uses (3.3) to calculate the inputs for the associated sub-layer.

The output layer has two sub-layers, and each sub-layer has two neurons. The first sub-layer determines the  $\Delta x$ , and the second one determines  $\Delta y$ . The first neuron of the sub-layers is for negative values, and the second one is for positive values. Each neuron is associated with the output sign, and the magnitude of the  $\Delta x$  and  $\Delta y$  is encoded into the output sub-layers based on the minimum and maximum synaptic weights. Equation (3.3) is used to encode the magnitude of the random action into the output sub-layers. The only difference is that a function called  $\mu_O$  is used to normalize the maximum step between 0 and 1 as follows,

$$\mu_{Ox} = \frac{\Delta x}{\Delta X_{max}} \quad (3.4)$$

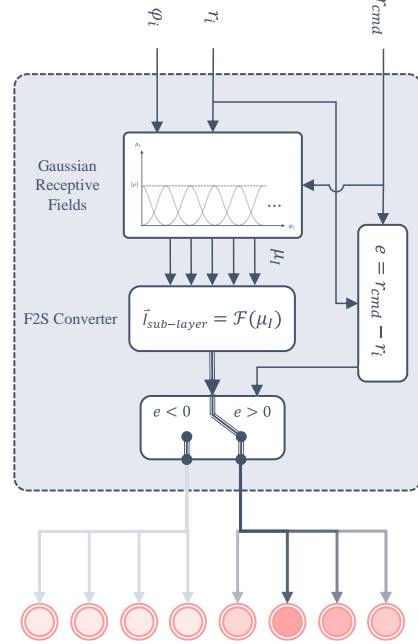


Figure 3.3: The fuzzy encoding principle for the input sub-layer.

$$\mu_{Oy} = \frac{\Delta y}{\Delta Y_{max}} \quad (3.5)$$

where  $\Delta x$  and  $\Delta y$  are selected actions, and  $\Delta X_{max}$  and  $\Delta Y_{max}$  are maximum steps (displacements) in  $X$  and  $Y$  directions. Two random actions, one for  $\Delta x$  and one for  $\Delta y$ , are generated for the training process.

The decoding of the spiking output is determined by the difference in the firing rates of the output neurons within each sub-layer. Let us denote  $f(t)$  as the activity of the output neurons that control the movement in the x and y-directions:

$$f(i) = \begin{cases} 1 & \text{if the neuron spikes at time } i, \\ 0 & \text{otherwise,} \end{cases}$$

The equation for decoding this activity can be expressed as:

$$\Delta x_{decoded} = \left[ \sum_{i=t-\Delta T}^t (f^{x+}(i) - f^{x-}(i)) \right] \Delta X_{max} \quad (3.6)$$

where  $f^{x+}(i)$  and  $f^{x-}(i)$  are the activities of the two output neurons associated with the x-direction. A similar process is applied for decoding in the y-direction:

$$\Delta y_{decoded} = \left[ \sum_{i=t-\Delta T}^t (f^{y+}(i) - f^{y-}(i)) \right] \Delta Y_{max} \quad (3.7)$$

where  $\Delta T$  is the time window the network updates weights.

One of the challenges in robotic applications is ensuring smooth transitions in actions to prevent abrupt and potentially harmful changes. Therefore, the recursive random number generation method is used to produce correlated random numbers. This method ensures that during training, the current displacements of the robot are influenced by its previous displacements, leading to smoother transitions. The recursive random number generation can be formulated as,

$$\Pi_t = \gamma \cdot \Pi_{t-1} + (1 - \gamma) \cdot \Upsilon_t \quad (3.8)$$

In this equation,  $\Pi_t$  represents the random action at time  $t$ ,  $\gamma$  is a correlation coefficient that controls the influence of the previous action, and  $\Upsilon_t$  is a random number drawn from a standard distribution (e.g., Gaussian) at time  $t$ . This recursive formulation ensures that the action at any given time  $t$  is a weighted blend of the previous action and a new random input, functioning as a first-order filter to produce colored noise.

### 3.2.2 Training algorithm

The R-STDP algorithm without considering the eligibility trace ( $C$ ) is used for training. The reward  $\mathcal{R}(t)$  at time  $t$  is defined as,

$$\mathcal{R}_{Fi}^{Fj}(t) = \mathcal{C}_{Fi}^{Fj} \left[ r_{Fi}^{Fj}(t-1) - r_{Fi}^{Fj}(t) \right] \tanh(r_{Fi}^{Fj}(t) - r_{cmd}) \quad (3.9)$$

$$\mathcal{R}_{Fi}^L(t) = \mathcal{C}_{Fi}^L [r_{Fi}^L(t-1) - r_{Fi}^L(t)] \tanh(r_{Fi}^L(t) - R_{cmd}) \quad (3.10)$$

where,  $\mathcal{R}_{Fi}^{Fj}$ ,  $r_{Fi}^{Fj}$ , and  $r_{cmd}$  denote the reward, distance, and commanded distance between two  $i$  and  $j$  follower agents, respectively. Similarly,  $\mathcal{R}_{Fi}^L$ ,  $r_{Fi}^L$ , and  $R_{cmd}$  represent the reward, distance, and the commanded distance between the follower agent  $i$  and the Leader ( $L$ ), respectively. The term  $\mathcal{C}_{Fi}^{Fj}$  and  $\mathcal{C}_{Fi}^L$  are reward coefficients and the  $\tanh(r_{Fi}^{Fj}(t) - r_{cmd})$  and  $\tanh(r_{Fi}^L(t) - R_{cmd})$  functions determine the reward's sign according to the agents' relative distance and the commanded distance. The expressions  $r_{Fi}^{Fj}(t-1) - r_{Fi}^{Fj}(t)$  and  $r_{Fi}^L(t-1) - r_{Fi}^L(t)$  specify the magnitude of the instantaneous reward.

If an agent finds itself farther away from the commanded distance than a neighboring agent or the leader, it will be rewarded positively for decreasing its distance. Conversely, moving closer results in a negative reward if the agent is within the commanded distance from a neighboring agent or the leader. This system is designed to encourage the maintenance of a commanded distance: being too far away from the commanded distance invites a penalty. At the same time, positive reinforcement is given for closing the gap between the current distance and the commanded distance.

One of the challenges in R-STDP is the unbounded growth or decay of synaptic weights, which can impede stable and effective learning in neural networks. The following section introduces a novel method focused on learning rate and weight stabilization to address this challenge and enhance the algorithm's applicability. This proposed method, designed to regulate synaptic weight changes, ensures a balanced and controlled learning process. It innovatively incorporates an adaptive decay rate technique designed to maintain stability in synaptic weight adjustments, thereby significantly improving the performance and reliability of R-STDP in SNNs.

### 3.2.3 Weight Stabilization using Reward-Modulated Competitive Synaptic Equilibrium (R-CSE)

Controlling the excessive increase of synaptic weights in SNNs is important to maintain network resilience and function. If not controlled, this growth can lead to saturation, affecting the network's ability to learn and adapt. When the network receives fuzzy sets of firing strengths as input, the synaptic weights grow in a pattern influenced by the Gaussian function's shape used for fuzzy encoding. Imposing a limit on synaptic weights disrupts this growth pattern over time, and eventually, all the synaptic weights reach the maximum. Weight normalization, while preventing excessive growth in one part of the network, can inhibit overall growth; when a synaptic connection reaches its maximum, its activation subsequently diminishes other weights.

Traditional methods like L1 regularization and weight decay employ a constant decay rate, which can slow the network's responsiveness to changes in rewards. Alternatively, a more advanced approach, the Bienenstock, Cooper, and Munro (BCM) method dynamically adjusts both a threshold and a decay rate in response to input variations. However, this method does not provide a control mechanism for the fuzzy inputs. In this chapter, we introduce a method called Reward-Modulated Competitive Synaptic Equilibrium (RCSE) to manage the unbounded growth of synaptic weights while maintaining the gradual change in the synaptic weights formed due to differences in firing strength from fuzzy membership functions. Our method also dynamically adjusts the network when the reward changes by adjusting the maximum synaptic weight based on the reward.

The enhanced version of the R-STDP method considering the control mechanism from RCSE algorithm is expressed as follows,

$$\dot{\mathbf{W}}(t) = \boldsymbol{\alpha} \odot \text{STDP}(\tau) \odot \mathcal{R}(t) - \boldsymbol{\Theta} \odot \text{sgn}(\mathbf{W}) \quad (3.11)$$

where  $\odot$  is the Hadamard product,  $\boldsymbol{\alpha}$  is the learning rate matrix, and  $\boldsymbol{\Theta}$  is the decay rate matrix. The primary distinction between the RCSE method and the approach detailed in

Section 2.4 lies in the decay rate, which allows the learning system to remain adaptable after the learning phase, and in the learning rate, which is represented as a matrix rather than a scalar value affecting all synaptic weights uniformly. These modifications enhance the learning algorithm’s flexibility in responding to reward changes and provide greater control over synaptic weight adjustments.

Let us define  $\mathcal{S}$  as the set of input and output neurons that fired at time  $t$  in one of the network sections. If we consider  $W_{max}^{\mathcal{S}}(t)$  as the maximum weight among the firing neurons in set  $\mathcal{S}$ , then we can characterize the learning rate using a Sigmoid function. The learning rate value ( $\alpha^{\mathcal{S}}(t)$ ) gradually transitions from 1 to 0 as the learning process advances, as explained below:

$$\alpha^{\mathcal{S}}(t) = \frac{1}{1 + \exp\left[\frac{1}{\epsilon}(|W_{max}^{\mathcal{S}}(t)| - \Psi^{\mathcal{S}})\right]}, \quad \left(\Psi^{\mathcal{S}} = \frac{\mathcal{R}_{max}^{\mathcal{S}}}{\mathcal{R}_{max}^G} I^{max}\right) \quad (3.12)$$

where  $\mathcal{R}_{max}^{\mathcal{S}}$  is the maximum reward in the network section (e.g.,  $\max(\mathcal{R}_{Fi}^{Fj})$ ),  $\mathcal{R}_{max}^G = \max(\mathcal{R}_{Fi}^{Fj}, \mathcal{R}_{Fi}^L)$ , and  $\epsilon$  is a small positive number that controls the curvature of the function around  $W_{max}^{\mathcal{S}}(t) = \Psi^{\mathcal{S}}$ . This model determines the learning rate by the highest synaptic weight among the active input and output neurons. This mechanism is similar to the “winner-takes-all” approach. When a synaptic connection reaches its weight limit, it prevents further changes in the adjacent synaptic weights.

The network contains a variety of reward functions, each with its own maximum and minimum values. The highest reward value in a specific area of the network sets the limit for the synaptic weight in that area. The synaptic weight limit is linked to the ratio of the local maximum reward ( $\mathcal{R}_{max}^{\mathcal{S}}$ ) to the global maximum reward ( $\mathcal{R}_{max}^G$ ). As a result, the network section with the highest local maximum reward ( $\mathcal{R}_{max}^{\mathcal{S}} = \mathcal{R}_{max}^G$ ) attains the maximum allowable synaptic weights because  $\frac{\mathcal{R}_{max}^{\mathcal{S}}}{\mathcal{R}_{max}^G} = 1$ , while sections with lower local maximum rewards reach only a proportional fraction of the maximum weight. The adjustment of the learning rate transforms into a competitive algorithm that modifies the growth rate of individual synaptic weights by considering network parameters, like reward and maximum synaptic weight.

A significant challenge in learning algorithms is their capacity to adapt to changes in rewards. Commonly, once the learning rate reduces to zero, weight adjustments stop. To address this, a variable decay rate is introduced to prevent weights in each network section from indefinitely remaining at their peak values. In our method, the decay rate is represented as a matrix, and it is calculated using the SoftPlus function, enabling it to adjust according to the current stage of learning. This method ensures that weight modifications continue to respond effectively to changes in the learning environment.

This chapter defines the decay rate as a function of the maximum synaptic weight among neurons in the set  $\mathcal{S}$ . This approach is designed to address a critical aspect: when the maximum synaptic weight in  $\mathcal{S}$  reaches its peak ( $|W_{max}^{\mathcal{S}}(t)| = \Psi^{\mathcal{S}}$ ), it is essential that the learning rate remains above zero. This condition is necessary to allow weight change and prevent the learning rate from stagnating at zero ((3.11)). Simultaneously, the learning rate must not exceed the maximum acceptable rate of weight change, which is  $\mathcal{A} \times \mathcal{R}_{max}^G$ . When the reward coefficients change after training, it can cause  $|W_{max}^{\mathcal{S}}(t)|$  to exceed  $\Psi^{\mathcal{S}}$  for set  $\mathcal{S}$ . With these considerations, we propose that the decay rate should be set to  $\mathcal{A}/\lambda \times \mathcal{R}_{max}^G$  when  $|W_{max}^{\mathcal{S}}(t)| = \Psi^{\mathcal{S}}$  and increase to  $\lambda \mathcal{A} \times \mathcal{R}_{max}^G$  when  $|W_{max}^{\mathcal{S}}(t)| = 2\Psi^{\mathcal{S}}$ , where  $\lambda$  is a coefficient that controls the rate of decay when  $|W_{max}^{\mathcal{S}}(t)| > \Psi^{\mathcal{S}}$ .

By applying the mentioned condition and solving for the SoftPlus function, the decay rate function can be obtained as follows,

$$\Theta^{\mathcal{S}} = \left( \frac{\eta}{\beta} \right) \log \left( 1 + \exp \left[ \beta \left( |W_{max}^{\mathcal{S}}(t)| - \Psi^{\mathcal{S}} \right) \right] \right) \quad (3.13)$$

where  $\eta = \frac{\mathcal{A}\mathcal{R}_{max}^G \ln(2^{\lambda^2}-1)}{\lambda\Psi^{\mathcal{S}} \log(2)}$  is a scaling parameter that can adjust the output scale of the function, and  $\beta = \frac{\ln(2^{\lambda^2}-1)}{\Psi^{\mathcal{S}}}$  controls the curvature of the function. A higher  $\beta$  makes the SoftPlus function approach a step function, making it closer to the binary behavior. Conversely, a smaller  $\beta$  makes the function smoother and more gradual. Equation (3.13) can be represented as,

$$\Theta^S = \left( \frac{\mathcal{A} \mathcal{R}_{max}^G}{\lambda \log(2)} \right) \log \left( 1 + \exp \left[ \left( \frac{\ln(2^{\lambda^2} - 1)}{\Psi^S} \right) (|W_{max}^S(t)| - \Psi^S) \right] \right) \quad (3.14)$$

The choice of setting the decay rate to  $\lambda \mathcal{A} \times \mathcal{R}_{max}^G$  when  $|W_{max}^S(t)| = 2\Psi^S$  is based on the feature of reward coefficients. Specifically, when the reward coefficients in (3.9) and (3.10) increase, leading to new condition where  $\mathcal{R}_{max}^S$  or  $\mathcal{R}_{max}^G$  change, the  $|W_{max}^S(t)|$  is allowed to increase. Conversely, a decrease in the reward coefficient, resulting in  $|W_{max}^S(t)| > \Psi^S$ , necessitates a higher decay rate to reduce the  $|W_{max}^S(t)|$  back to  $\Psi^S$ .

When  $|W_{max}^S(t)| < \Psi^S$ , the reward adjusts the synaptic weights, and there is no weight decay to disturb the learning process. When  $|W_{max}^S(t)| > \Psi^S$ , the decay rate changes the synaptic weights and brings the maximum weight to the reward zone, where  $|W_{max}^S(t)| < \Psi^S$  and the networks responds to reward change.

### Numerical example for the RCSE algorithm

Consider a set of input neurons firing based on their fuzzy membership degrees. For an input received, a set of adjacent input neurons fires, along with an output neuron. Let the set  $\mathcal{S}$  be defined based on the fired neurons as follows:

$$\mathcal{S} = \{15, 16, 17, 18, 73\}$$

In this set, neurons 15, 16, 17, and 18 are fired input neurons, and neuron 73 is the fired output neuron. First, the algorithm finds the maximum synaptic weight connecting these input neurons to the output neuron, denoted as  $W_{max}^S(t)$ . Let's assume that  $I^{max} = 15.5$  and  $\epsilon$  is a small positive number (e.g., 0.0001).

The decay rate is formulated using a SoftPlus function, which is parameterized by  $\lambda$  and  $\mathcal{A}$ , where  $\lambda = 5$  and  $\mathcal{A} = 1$  in this numerical example. The  $\eta$  and  $\beta$  influence the decay rate adjustments under various synaptic conditions and reward structures.

#### Case 1: $W_{max}^S(t) < \Psi^S$

Let's assume that  $\mathcal{R}_{max}^S = 0.5$  and  $\mathcal{R}_{max}^G = 1$  ( $\Psi^S = 0.5/1 \times 15.5 = 7.75$ ), so the maximum change rate for this set  $\mathcal{S}$  is  $\mathcal{A} \mathcal{R}_{max}^S = 0.5$  while the maximum change rate in the network is

$\mathcal{AR}_{max}^G = 1$ . For  $W_{max}^S(t) = 7$ , the decay rate  $\Theta^S$  can be calculated as follows,

$$\beta = \frac{\ln(2^{\lambda^2} - 1)}{\Psi^S} = \frac{\ln(2^{5^2} - 1)}{7.75} = 2.236$$

$$\eta = \frac{\mathcal{AR}_{max}^G \ln(2^{\lambda^2} - 1)}{\lambda \Psi^S \log(2)} = \frac{1 \cdot 1 \cdot \ln(2^{5^2} - 1)}{5 \cdot 7.75 \cdot \log(2)} = 1.485$$

$$\Theta^S = \left( \frac{1.485}{2.236} \right) \log(1 + \exp[2.236 \cdot (7 - 7.75)]) = 0.05$$

$$\alpha^S(t) = \frac{1}{1 + \exp\left[\frac{1}{0.0001}(7 - 7.75)\right]} = 1$$

According to (3.11), the decay rate is low enough in comparison with the learning rate to allow for synaptic growth ( $\mathcal{AR}_{max}^S$ ), promoting an increase in synaptic strength.

### Case 2: $W_{max}^S(t) > \Psi^S$

Let's assume that  $W_{max}^S(t) = 10$ , the decay and learning rates can be calculated as follows,

$$\Theta^S = \left( \frac{1.485}{2.236} \right) \log(1 + \exp[2.236 \cdot (10 - 7.75)]) = 1.453$$

$$\alpha^S(t) = \frac{1}{1 + \exp\left[\frac{1}{0.0001}(10 - 7.75)\right]} \approx 0$$

This results in a higher decay rate for the set  $S$  (while  $\alpha^S = 0$ ), actively working to reduce synaptic strength towards the threshold  $\Psi^S$ , due to  $W_{max}^S(t)$  exceeding  $\Psi^S$ .

### Case 3: When $\Psi^S$ increases from 7.75 to 15.5 (Reward Change)

Assuming a reward change causes  $\Psi^S$  to increase to 15.5 ( $\mathcal{R}_{max}^S = \mathcal{R}_{max}^G = 1$ ) and considering  $W_{max}^S(t) = 7.8$ , the decay rate  $\Theta^S$  is recalculated as follows,

$$\beta = \frac{\ln(2^{\lambda^2} - 1)}{\Psi^S} = \frac{\ln(2^{5^2} - 1)}{15.5} = 1.118$$

$$\eta = \frac{\mathcal{AR}_{max}^G \ln(2^{\lambda^2} - 1)}{\lambda \Psi^S \log(2)} = \frac{1 \cdot 1 \cdot \ln(2^{5^2} - 1)}{5 \cdot 15.5 \cdot \log(2)} = 0.743$$

$$\Theta^S = \left( \frac{0.743}{1.118} \right) \log(1 + \exp[1.118 \cdot (7.8 - 15.5)]) \approx 0$$

$$\alpha^S(t) = \frac{1}{1 + \exp\left[\frac{1}{0.01}(7.8 - 15.5)\right]} = 1$$

The decay rate allows for synaptic growth of set  $\mathcal{S}$  as the maximum weight is below the new threshold.

#### **Case 4: When $\Psi^S$ decreases from 15.5 to 7.75 (Reward Change)**

If  $\Psi^S$  decreases to 7.75 due to reward changes and assuming  $W_{max}^S(t) = 15.6$ , the decay rate  $\Theta^S$  increases as follows,

$$\beta = \frac{\ln(2^{\lambda^2} - 1)}{\Psi^S} = \frac{\ln(2^{5^2} - 1)}{7.75} = 2.236$$

$$\eta = \frac{\mathcal{AR}_{max}^G \ln(2^{\lambda^2} - 1)}{\lambda \Psi^S \log(2)} = \frac{1 \cdot 1 \cdot \ln(2^{5^2} - 1)}{5 \cdot 7.75 \cdot \log(2)} = 1.485$$

$$\Theta^S = \left( \frac{1.485}{2.236} \right) \log(1 + \exp[2.236 \cdot (15.6 - 7.75)]) = 5.063$$

$$\alpha^S(t) = \frac{1}{1 + \exp\left[\frac{1}{0.01}(15.6 - 7.75)\right]} = 0$$

In this scenario, the decay rate is very high, and the learning rate is 0. Therefore, the algorithm aggressively pulls the synaptic weight back toward the lowered threshold.

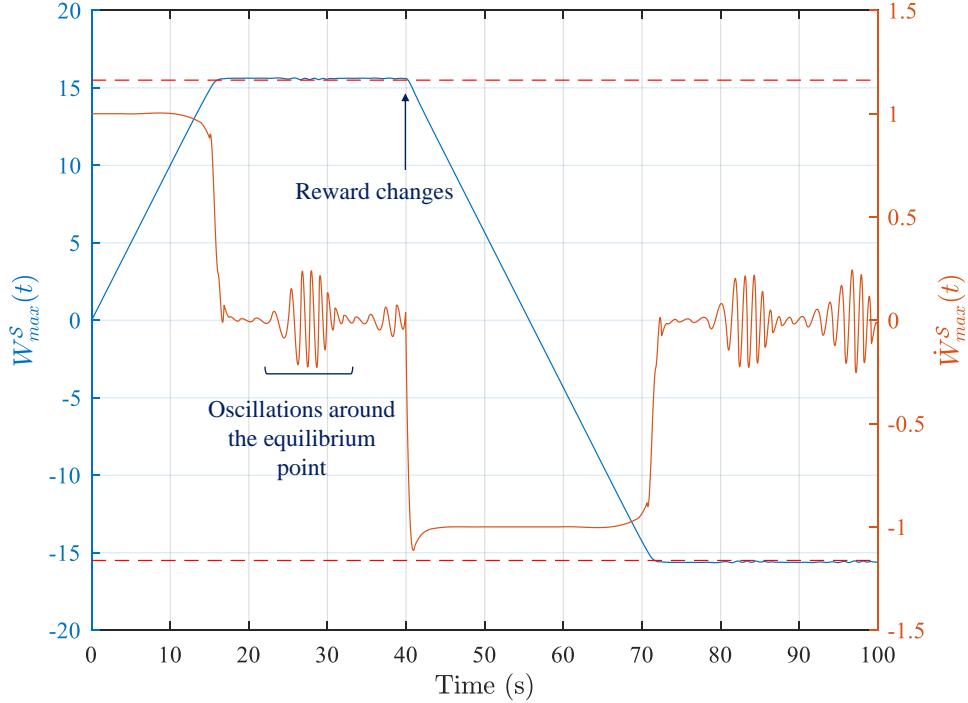


Figure 3.4: Synaptic weight change for  $\lambda = 5$ ,  $\Psi^S = 15.5$ , and  $\mathcal{AR}_{max}^G = 1$ .

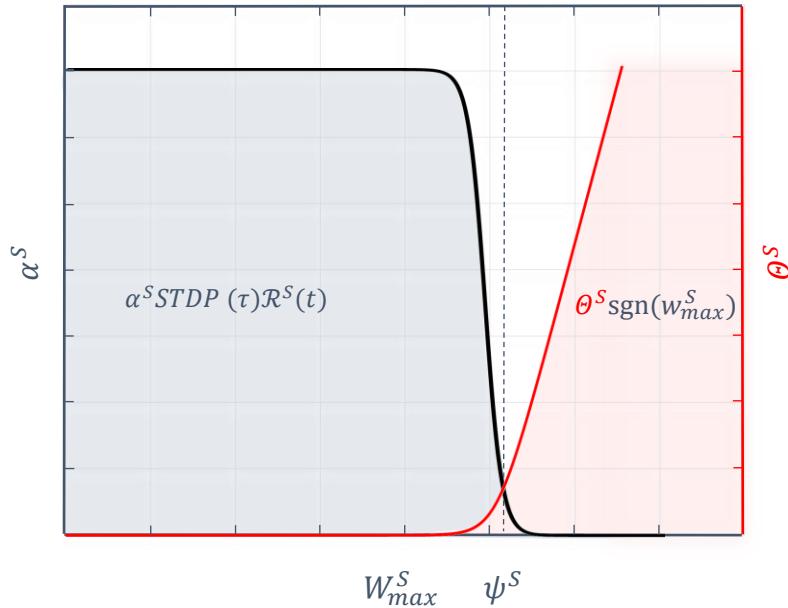


Figure 3.5: Reward-based learning rate and decay rate functions. In the blue region (active learning rate), the reward adjusts the weights, and in the red region (active decay rate), the RCSE method controls synaptic growth.

Figure 3.4 demonstrates the performance of the RCSE when it regulates the synaptic weights to prevent unbounded growth. The red dotted line represents the value of  $\Psi^S$ , which is derived from the maximum reward value of the corresponding network section. As shown in Figure 3.4, the synaptic weight oscillates around  $\Psi^S$ , and when it drops below  $\Psi^S$ , the learning rate is set to 1 by (3.12). This allows any changes in the reward function to be applied to the synapse.

According to Figure 3.5, when  $W_{max}^S \leq \Psi^S$ , synaptic weights in set  $S$  increase. If the reward changes,  $\Psi^S$  also changes. Depending on the current value of  $W_{max}^S$ , the RCSE either increases or decreases the synaptic weights within set  $S$ .

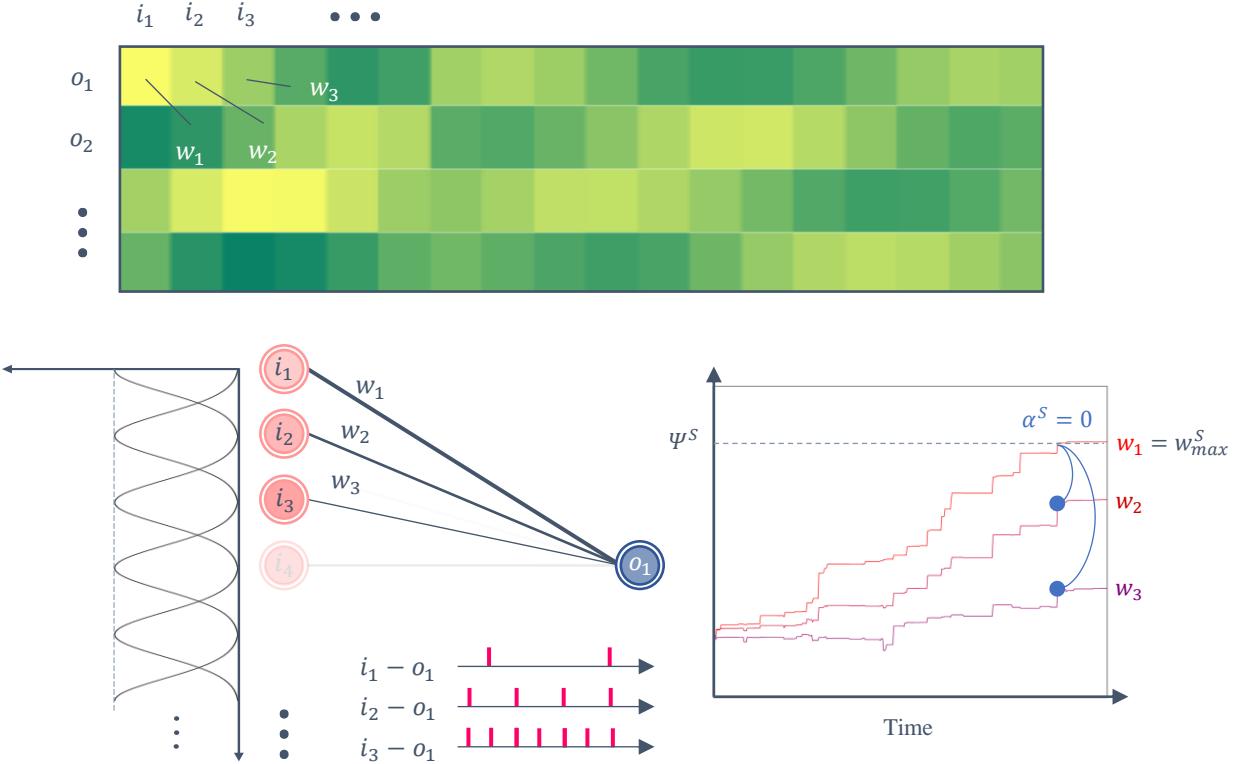


Figure 3.6: RCSE working principle in inhibiting the adjacent synaptic connections. The heatmap shows the synaptic weight matrix. Neurons have different firing strengths due to the difference in fuzzy membership values, which affects the increase or decrease rate and shapes the patterns in the synaptic weight matrix.

Figure 3.6 shows how the maximum synaptic weight of the active synapses in set  $S$  stops

the adjacent synaptic connections' growth by setting the learning rate of the set to 0.

### 3.2.4 Federated Learning for Consensus Flying

In FL, a key challenge is centralizing various models on one server. This process must effectively combine these models to create a unified global model without compromising the specific adjustments made to each model. A critical strategy involves choosing models that contain substantial information. Another significant aspect is determining the frequency of model aggregation. Shorter intervals between aggregations can enhance learning efficiency but may strain network resources, particularly as the number of participating agents and devices grows. Conversely, longer intervals might slow down the learning process due to delayed updates of the global model. This section proposes an aggregation method for SNN. Our focus is on reducing network usage and energy consumption.

This approach allows clients to upload their local model updates at different times rather than synchronously. Such a method is particularly beneficial in reducing the negative impacts of device heterogeneity, which can include varying computational capacities and network connectivity among devices [67]. In traditional FL setups, delays caused by poor network signals or unexpected client crashes can significantly prolong the time the server takes to receive updates from all clients. By adopting asynchronous aggregation, the server processes and aggregates models as they are received without synchronizing with all clients. This strategy accelerates the training process, making FL more efficient and adaptable to diverse client conditions.

Our proposed FL model aggregation algorithm aims to establish an efficient and event-triggered system for global and local model publishing. This system relies on the similarity between consecutive global and local models and publishes updates only when significant changes are detected, thus avoiding redundant updates and improving overall efficiency. Unlike the uniform model updates in FedAvg [68, 69], our approach allows individual agents to evaluate and send their local models based on a similarity threshold with the global model, thereby enabling a potentially more effective update process. Our aggregation strategy

emphasizes similarity metrics for model updates, which is not commonly emphasized in methods like FedNova [70], adding a layer of context sensitivity to our approach.

In our approach, considering the difference in agents' neural network parameters and maximum and minimum synaptic weights, the weights are normalized to align them on a uniform scale ranging from -1 to 1. This normalization process makes the neural model values comparable across the network. Based on the maximum and minimum synaptic weights outlined in Equations (1.5) and (1.9), and taking into account the highest excitation ( $I^{max}$ ) and inhibition ( $-I^{max}$ ), the normalization of synaptic weights is performed as follows:

$$\bar{\mathbf{W}}_k(t) = \frac{1}{I_k^{max}} [\mathbf{W}_k(t)] \quad (3.15)$$

where  $\mathbf{W}_k(t)$  represents the matrix of synaptic weights,  $\bar{\mathbf{W}}_k(t)$  denotes the normalized synaptic weight matrix for agent  $k \in \{1, 2, 3, \dots\}$ , and  $I_k^{max}$  is the maximum synaptic weight for agent  $k$ .

The global model on the server (Leader) is then computed using a weighted average,

$$\bar{\mathbf{W}}_G(t) = \frac{\sum_{k=1}^N \omega_k \cdot \bar{\mathbf{W}}_k(t)}{\sum_{k=1}^N \omega_k} \quad (3.16)$$

where  $\bar{\mathbf{W}}_G(t)$  is the global normalized model on the central server,  $N$  is the number of agents, and  $\omega_k$  is the aggregation weight for each SNN model, defined as,

$$\omega_k = \frac{1}{\sqrt{mn}} \|\bar{\mathbf{W}}_k(t)\|_F \exp \left( -\frac{t - T_k}{\tau_{cs}} \right) \quad (t \geq T_k) \quad (3.17)$$

where the term  $\|\cdot\|_F$  is the Frobenius norm, and  $m$  and  $n$  are the dimensions of the matrix  $\bar{\mathbf{W}}_k(t)$ , used for normalizing the Frobenius norm.  $T_k$  indicates the time at which agent  $k$  last transmitted its local model to the central server, and  $\tau_{cs}$  is a time constant that reduces the weight to zero if there is no recent update from the agent.

Both agents and the central server employ an event-triggered mechanism for transmitting local and global models. Throughout the training phase, each agent calculates the Euclidean distance between the most recent global model from the central server and its

current synaptic weights matrix, as follows,

$$\mathcal{D}_a(\bar{\mathbf{W}}_k(t), \bar{\mathbf{W}}_G(T_{cs})) = \frac{1}{2\sqrt{mn}} \sqrt{\sum_{i=1}^m \sum_{j=1}^n (a_{ij} - b_{ij})^2} \quad (3.18)$$

where  $\mathcal{D}_a$  is the Euclidean distance on the agent side,  $T_{cs}$  is the time when the central server published the global model, and  $a_{ij}$  and  $b_{ij}$  are elements of the latest global model and the current local model, respectively. If this distance exceeds a certain threshold, set between 0 and 1, the agent transmits its model to the central server.

If the  $\mathcal{D}_a$  on the agent  $k$  reaches the threshold and it does not receive any update from the server, the agent sends its model to the server, and then it calculates the  $\mathcal{D}_a$  between current synaptic weights  $\bar{\mathbf{W}}_k(t)$  and the last model that it sent to the server at time  $T_k$  ( $\bar{\mathbf{W}}_k(T_k)$ ) until it receives a new model update from the central server.

The central server follows a similar procedure as the agents, evaluating the distance  $\mathcal{D}_G$  between the current and recently published model at time  $T_{cs}$ ,

$$\mathcal{D}_G(\bar{\mathbf{W}}_G(t), \bar{\mathbf{W}}_G(T_{cs})) = \frac{1}{2\sqrt{mn}} \sqrt{\sum_{i=1}^m \sum_{j=1}^n (a_{ij} - b_{ij})^2} \quad (t \geq T_{cs}) \quad (3.19)$$

Incorporating the proposed FL method with the RCSE algorithm, the modified R-STDP equation can be represented as follows,

$$\begin{aligned} \dot{\mathbf{W}}_k(t) = & (1 - \delta(t - T_{cs})) [\boldsymbol{\alpha} \odot \mathbf{STD}\mathbf{P}(\tau) \odot \mathcal{R}(t) - \boldsymbol{\Theta} \odot \text{sgn}(\mathbf{W}_k(t))] \\ & + \delta(t - T_{cs}) I_k^{max} (\bar{\mathbf{W}}_G(t) - \bar{\mathbf{W}}_k(t)) \end{aligned} \quad (3.20)$$

where  $\delta$  is the Dirac delta function. Algorithm 2 shows the step-by-step implementation process of the proposed method. The  $\bar{\mathbf{W}}_G(t) - \bar{\mathbf{W}}_k(t)$  part is the difference between the agent's current model and the global model at aggregation time. Therefore, the change in synaptic weights  $\dot{\mathbf{W}}_k(t)$  is the difference between models when  $t = T_{cs}$ .

---

**Algorithm 2** High-Level Algorithm for the Proposed FL Algorithm

---

**Require:** Initialization of Central Server and Agents**Ensure:** Updated Global Model on the Central Server and Local Models on Agents

```

1: Initialize the agents and Central Server with default parameters for model publication
   threshold, Euclidean distance, and model publish status
2: Initialize the Global Model on the Central Server
3: if  $t$  is greater than 0 then
4:   Normalize synaptic weights of local models using (3.15)
5:   Aggregate models from all agents at the Central Server using (3.16)
6:   Calculate the  $\mathcal{D}_G$  between the current and previous global models on the Central
   Server using (3.19)
7:   if  $\mathcal{D}_G >$  the Central Server's threshold then
8:     Publish the global model
9:     set  $T_{cs} = t$ 
10:    end if
11:    for each Agent in the network do
12:      if Central Server publishes a new global model then
13:        Update the local model of the Agent with the global model using (3.20)
14:      else
15:        Agents evaluate their local models against the latest global model ( $\mathcal{D}_a$ ) using
           (3.18)
16:        if  $\mathcal{D}_a >$  the Agent's threshold then
17:          Send the model to the Central Server
18:          set  $T_k = t$ 
19:        end if
20:      end if
21:    end for
22:  end if

```

---

The proposed algorithm allows agents to communicate less often and save energy. It only sends essential updates to the Central Server, which helps when many agents have different SNN models and communication interfaces. This method reduces unnecessary data transmission, making the whole system more efficient.

### 3.3 Results and Discussion

In this section, we conducted a numerical simulation to validate the performance of the proposed method. The simulation involves a group of five agents flying around a leader who is moving in a circular path. Initially, a scenario without implementing FL was conducted to evaluate the performance of the SNN in achieving coordinated flight. During this phase, the effect of the change in reward was simulated to examine the RCSE method. In the second part of the simulation, the proposed FL aggregation algorithm is used, where the leader agent acts as a central server. Finally, the algorithm was tested both before and after changing the rewards.

#### 3.3.1 Simulation without FL

In this simulation, we modeled five agents, each equipped with its own SNN model, capable of reaching a maximum speed of  $1 \text{ m/s}$ . The architecture of each agent's neural network included 72 input neurons. The number of input neurons was derived from a series of numerical simulations. Since each agent was designed to detect three distinct objects within its environment, the input layer was organized into sub-layers, with 24 neurons dedicated to each object (24 membership functions and neurons for each object). The network's output layer comprised 4 neurons, divided equally to represent negative and positive  $\Delta x$  and  $\Delta y$  movements. The SNN model in the simulation is a fully connected network, and the parameters of the LIF neuron are also presented in Table 3.1.

Table 3.1: Parameter values for LIF neuron model [10]

Parameter	Value	Description
$R_m$	40 MΩ	Membrane Resistance
$\tau_m$	30 ms	Membrane time constant
$E_l$	-70 mV	Resting potential
$V_{res}$	-70 mV	Reset potential
$V_0$	-70 mV	Initial membrane potential
$V_{th}$	-50 mV	Threshold membrane potential

The R-STDP mechanism updated synaptic weights at 10 ms intervals. During these intervals, the learning algorithm adjusted the agent's states based on received data from other agents and the leader while simultaneously generating random outputs as part of an exploration strategy (section 3.2.3).

Table 3.2: Simulation Parameters

Parameter	Value	Description
$\Delta T$	10 ms	Weight and state update sample time
$\tau_s$	2 ms	Time constant for R-STDP
$\mathcal{A}$	1	Amplitude in R-STDP function
$\lambda$	5	Decay rate coefficient
$\Delta x$ and $\Delta y$	0.01 m	Max step per $\Delta T$
$\sigma$	0.5	Gaussian function's std. deviation
$\Delta t$	1 ms	Minimum inter-spike interval
$I^{min}$	0.5	Lower bound of synaptic weight
$I^{max}$	15.5	Upper bound of synaptic weight
$\gamma$	0.95	Correlation Coefficient

Table 3.2 shows the simulation parameters. The simulation was done in a 10 m by 10 m area, and the leader followed a circular path centered at (5,5) with a 2.5 m radius and a 0.1 m/s speed.

In order to monitor the swarm performance, the minimum and maximum distances of each agent from other agents and the minimum and maximum distances of the swarm from the leader were measured. Figure 3.7 shows the definition of the distances.

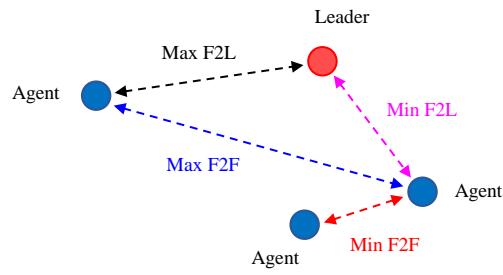


Figure 3.7: Measured distances used for evaluating swarm flight performance and collision detection.

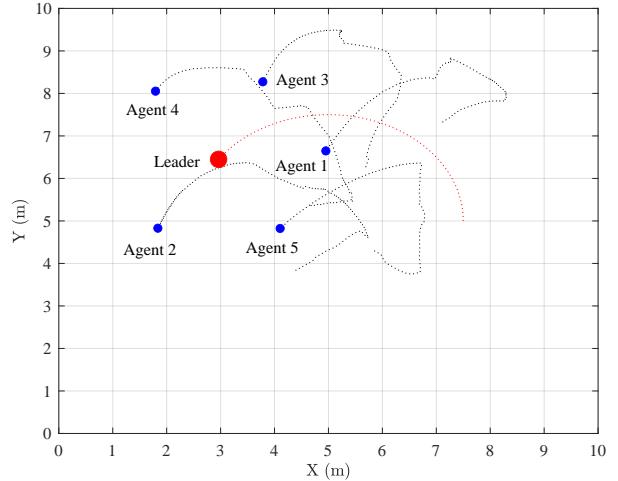


Figure 3.8: Agents' trajectory during the test phase

The simulation included two phases. During the initial phase, the objective was for the agents to learn to maintain the commanded distance from each other and the leader. This phase took 600 seconds for training, and the reward coefficient among followers ( $\mathcal{C}_{Fi}^{Fj}$ ) was set at 0.02, while the coefficient between followers and the leader ( $\mathcal{C}_{Fi}^L$ ) was set at 0.07. These parameters were derived from a series of numerical simulations. A higher value of  $\mathcal{C}_{Fi}^L$  signifies an increased emphasis on the leader in the learning process, which means that the distance to the leader is more important than the commanded distance between agents.

Figure 3.9 shows the simulation results for the RCSE method. According to the results, the agents rapidly aligned around the leader within 6.89 seconds, and the maximum distance was reduced from 7.632 meters to the target distance of 2 meters. The swarm completed the formation around the leader in approximately 8.94 seconds, avoiding collisions.

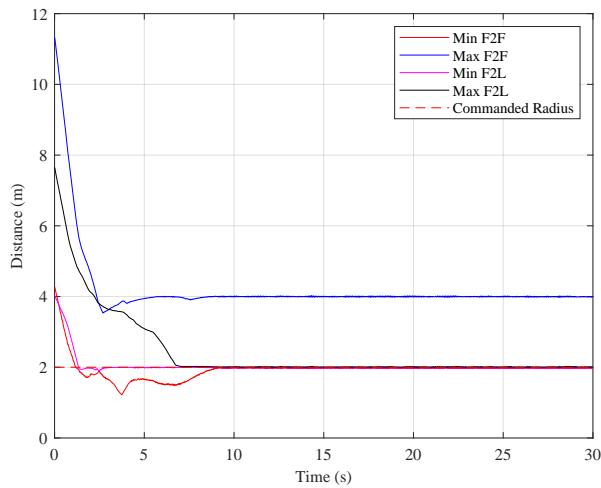


Figure 3.9: Variation of distances within the swarm during the test phase

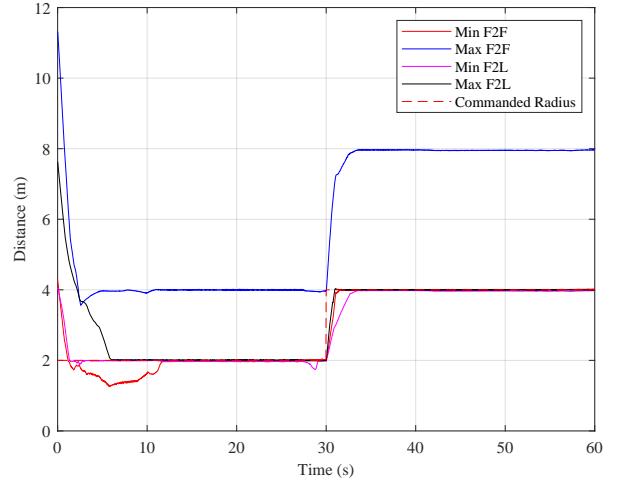


Figure 3.10: Adaptive response to commanded distance adjustments - reconfiguration during the test phase

As mentioned in section 3, the input encoding uses the error between current and commanded distance. Therefore, one of the advantages of the encoding and learning method in this chapter is that the learned policies are independent of the commanded distance. The commanded distance can be changed after training since the SNN uses the distance error. Figure 3.10 shows the agents' response to changes in commanded distance after training. According to this figure, when the commanded distance is changed at 30 seconds, the swarm immediately responds to this change in 2.98 seconds without disrupting the formation or any collision.

After 600 seconds, the leader is changed into an obstacle, and its reward coefficient  $\mathcal{C}_F^L$  is changed to 0.0175. The reward sign function,  $\tanh$  in (3.10), is also changed to -1, so the reward function for the leader is changed as follows,

$$\mathcal{R}_{Fi}^L(t) = -\mathcal{C}_{Fi}^L [r_{Fi}^L(t-1) - r_{Fi}^L(t)] \quad (3.21)$$

When the leader is transformed into an obstacle, the encoding equation for the input layer

must be changed. This is because the obstacle has no commanded distance, and the agents must maintain a commanded distance only from each other. Therefore, the commanded distance from the obstacle encoder in the input layer must be removed. Therefore, (3.1) can then be rewritten as follows:

$$\mu_I(\phi_i) = \exp\left(-\frac{(\phi_i - \zeta)^2}{2\sigma^2}\right) \quad (3.22)$$

The simulation proceeded for an additional 1200 seconds, during which the synaptic weights were adjusted in accordance with the new reward function given by (3.21). In this case, because the leader is now an obstacle, so then  $\alpha^S(t)$  in Figure 3.5 switches to 1 and weight adaptation starts again.

The results of the reward change are shown in Figures 3.11 and 3.12, which indicates that the agents quickly reduced their initial distance to the commanded distance of 2 m. Simultaneously, the minimum distance from the obstacle, which was the leader, increased over time, indicating that the agents adapted their behavior to maintain a greater distance from the obstacle. Figure 3.11 shows the trajectory of each agent after the reward change.

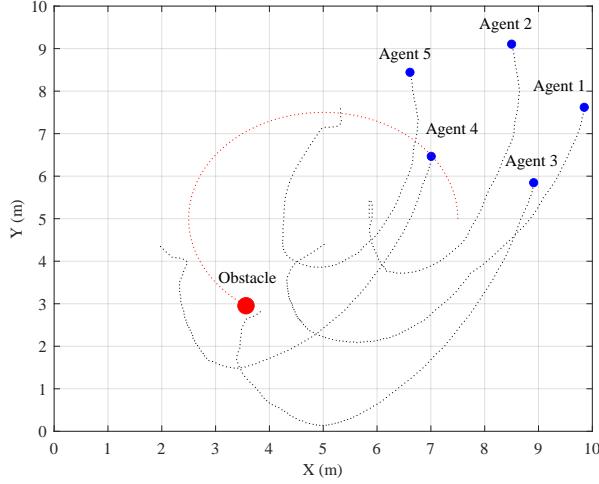


Figure 3.11: Trajectory adaptations of following agents in response to reward change for the leader during the test phase.

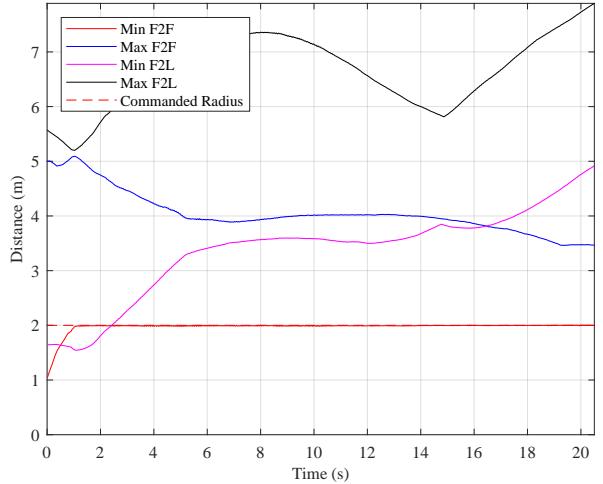


Figure 3.12: Variations in distances after reward changes and Leader becomes Obstacle - test phase.

Each agent in the swarm detects two neighboring agents and the leader, acquiring the Line-of-Sight angle and distance. Their neural network consists of three sub-layers in the input layer: two for neighboring followers ( $F_1$  and  $F_2$ ) and one for the leader ( $L$ ), with inputs encoded using Gaussian Receptive Fields and fuzzy membership functions. The network uses differences between current and commanded distances within the swarm ( $r_{cmd}$ ) and between followers and the leader ( $R_{cmd}$ ) to stimulate input neurons, which are split into parts for distances greater than and within the commanded distance, encoding the LOS angle with fuzzy membership functions. Errors in distances are transformed into amplitude values using the  $tanh$  function, bounded between 0 and 1. Each agent's neural network includes 72 input neurons, with 24 neurons per detected object, derived from numerical simulations. The output layer has 4 neurons representing positive and negative  $\Delta x$  and  $\Delta y$  movements.

In order to better understand the effect of reward change on the SNN, the synaptic weights matrix before and after reward change has been illustrated in Figures 3.13 and 3.14, the vertical axis shows the output neurons. The first output neuron is for negative

displacement in the x-direction, while the second output neuron is dedicated to positive displacement in the x-direction. Similarly, the third output neuron corresponds to negative displacement in the y-direction and the fourth output neuron to positive displacement in the y-direction. The horizontal axis shows the input neurons. The neuron IDs from 1 to 24 are for the first sub-layer dedicated to the neighboring follower. The network has two sub-layers for the neighboring follower agents, but only one is shown since they are similar in the case of synaptic weight values. The neuron numbers from 25 to 48 are for the sub-layer dedicated to the leader. The RCSE method aims to maintain the synaptic weight matrix gradient while adapting to changes in the reward signal.

Considering the numerical values presented in Table 3.2 along with the reward coefficients  $C_{F_i}^{Fj} = 0.02$  and  $C_L^{Fj} = 0.07$ , and  $r_{F_i}^{Fj} = 1 \text{ m/s}$  and  $r_{F_i}^L = 0.1 \text{ m/s}$ , the maximum rewards at each weight update interval ( $\Delta T$ ) for  $\mathcal{R}_{F_i}^{Fj}$  and  $\mathcal{R}_{F_i}^L$  are calculated using (3.9) and (3.10) as  $4 \times 10^{-4}$  and  $7.7 \times 10^{-4}$ , respectively. Consequently,  $\mathcal{R}_{max}^G = \max(\mathcal{R}_{F_i}^{Fj}, \mathcal{R}_{F_i}^L) = 7.7 \times 10^{-4}$ . The  $\Psi^S$  for the follower section in the network is  $\left[ \frac{4 \times 10^{-4}}{7 \times 10^{-4}} \right] 15.5 = 8.0519$ , and for the leader section, it is  $\left[ \frac{7 \times 10^{-4}}{7 \times 10^{-4}} \right] 15.5 = 15.5$ . The  $\eta$  and  $\beta$  for the follower section within the network are  $\frac{\mathcal{A}\mathcal{R}_{max}^G \ln(2^{\lambda^2} - 1)}{\lambda \Psi^S \log(2)} = 0.0011$  and  $\beta = \frac{\ln(2^{\lambda^2} - 1)}{\Psi^S} = 2.152$ , respectively. For the leader section, these values are  $\frac{\mathcal{A}\mathcal{R}_{max}^G \ln(2^{\lambda^2} - 1)}{\lambda \Psi^S \log(2)} = 5.719 \times 10^{-4}$  and  $\beta = \frac{\ln(2^{\lambda^2} - 1)}{\Psi^S} = 1.118$ , respectively.

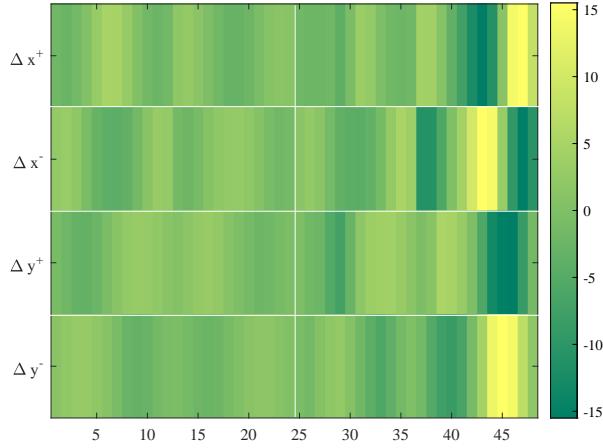


Figure 3.13: Synaptic Weights before Reward change in RCSE method.

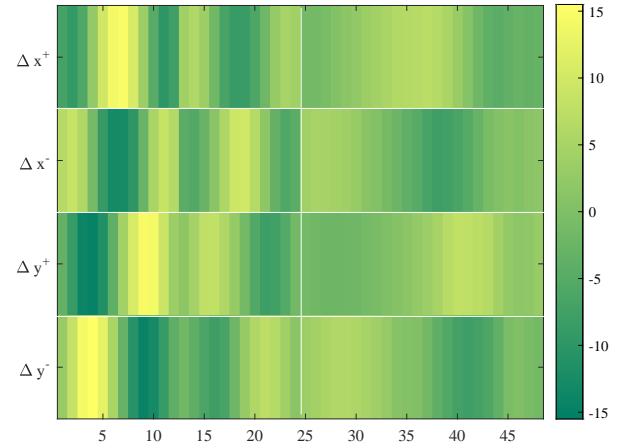


Figure 3.14: Synaptic Weights after Reward change in RCSE method.

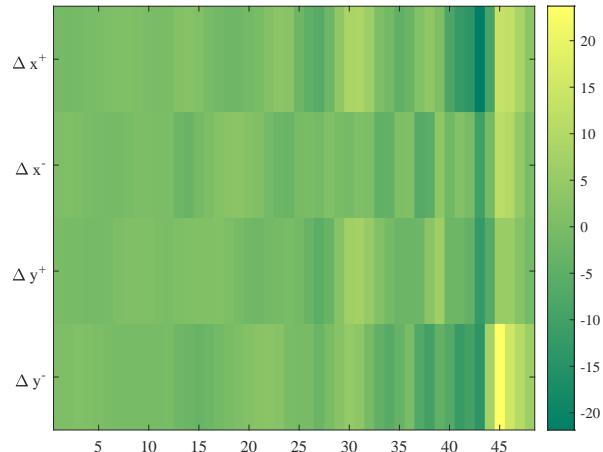


Figure 3.15: Synaptic Weights before Reward change in Multiplicative Synaptic Normalization method.

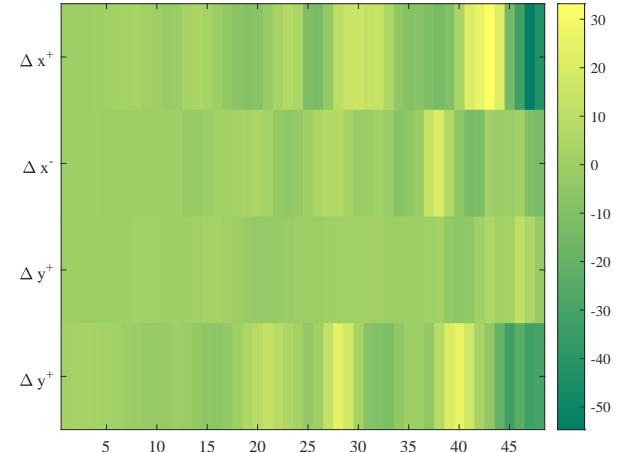


Figure 3.16: Synaptic Weights after Reward change in Multiplicative Synaptic Normalization method.

The visual patterns observed in the synaptic weights matrix in Figure 3.13 and 3.14, specifically, the gradual increases and decreases in values across weights, directly result from applying Gaussian membership functions for encoding. As illustrated in the heatmap vi-

sualization, regions of higher values denote areas closer to the function's center, where the degree of membership peaks. Conversely, areas of lower values reflect points moving away from the center, where the membership degree decreases according to the Gaussian distribution's tails. Figure 3.5 further demonstrates this behavior. The differences in synaptic strengths arise from variations in spike rates, reflecting differences in membership degree.

Since the reward coefficients for followers and leaders are different, their maximum allowed synaptic weights are also different. The proposed method given by (3.11) for controlling the unbounded growth of synaptic weights has successfully stabilized the network.

Figures 3.15 and 3.16 show the performance of the Multiplicative Synaptic Normalization (MSN) method. According to the results, this method cannot control the weight growth because of the coexistence of the excitatory and inhibitory (positive and negative synaptic weights). This method adjusts the weights when the summation of the input synaptic weights to an output neuron exceeds the maximum synaptic weight. In this case, the condition is never satisfied since positive and negative synaptic weights cancel out each other. According to Figure 3.16, when the reward changes for the Leader, the MSN cannot adjust the weights because it does not have the reward-modulated decay rate mechanism.

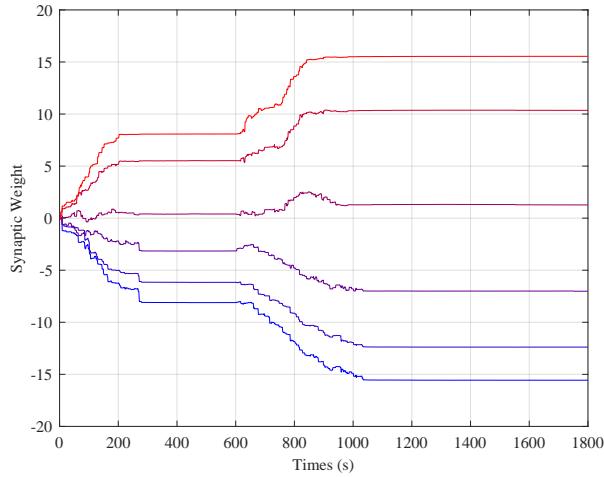


Figure 3.17: Synaptic weights increase after reward change in RCSE method.

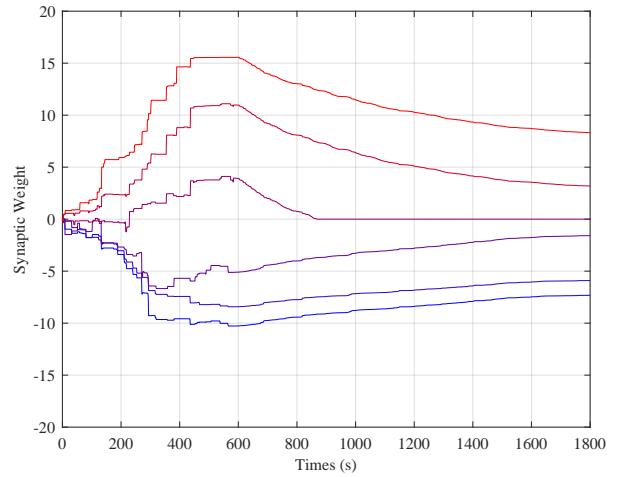


Figure 3.18: Synaptic weights decrease after reward change in RCSE method.

Figures 3.17 and 3.18 show the synaptic weights after the reward change. In this case, since the reward coefficients are changed, the  $\eta$  and  $\beta$  values in (3.13) are changed for the represented sub-layers, and the proposed method has helped the R-STDP algorithm to adjust the weights based on the new situation in the environment.

### 3.3.2 Simulation with FL and RCSE

In this section, the proposed aggregation algorithm is tested. In this case, the agents only send their models when the Euclidean distance between the current and previously published model or the latest global model reaches a threshold. In the first phase, the simulation was done in 600 seconds, and the agents learned to follow the leader. The threshold for publishing the agents' and server models was 0.0005 and 0.00051, respectively. The reason for choosing the server's threshold higher than the agents' is that as soon as the first agent sends its model to the server, the Euclidean distance between the current and previously published model by the server reaches 0.0005, and the server distributes the model immediately. Therefore, the serve's threshold is set higher than the agents' threshold, so it waits for the other agents to send their models.

Figure 3.19 shows the distances between agents and the leader before the reward change, indicating that the agents converge to the solution faster than in the scenario where federated learning has not been used, without any error. Figure 3.20 shows the simulation results for the reward change scenario, indicating that the proposed event-triggered FL method has improved the learning performance, enabling the swarm to converge to the solution in 6 seconds. In this case, the neural network learned the policies in less time while preserving the performance.

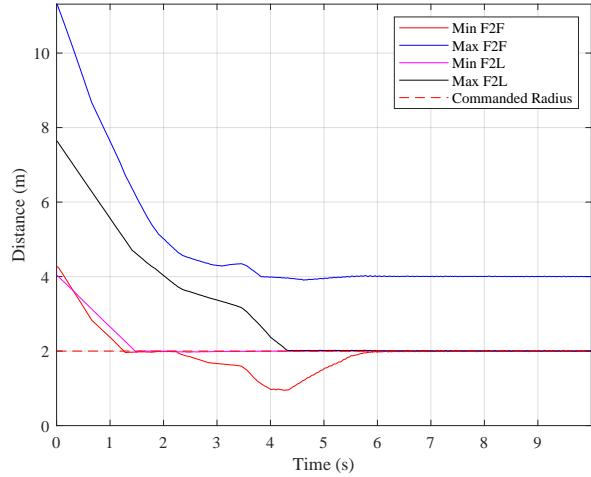


Figure 3.19: Distances during the test phase before reward change in the proposed event-triggered FL method.

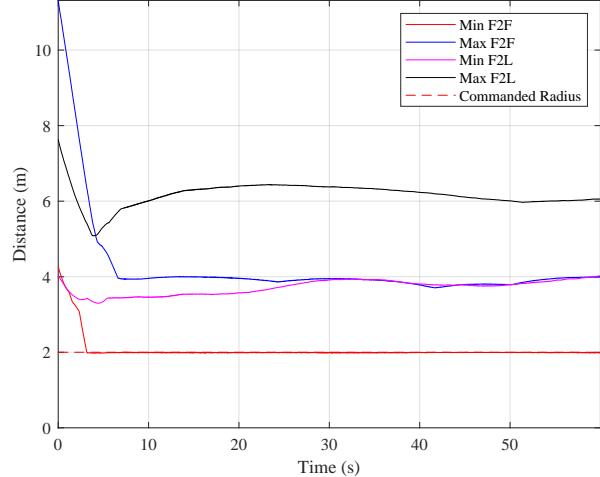


Figure 3.20: Distances during the test phase after reward change in the proposed event-triggered FL method.

The norm of the synaptic weights can represent the changes in the synaptic weights due to the change in the environment during the training process, which can be used to adjust the learning process in SNNs. In the proposed event-triggered FL method, agents communicate with the Central Server (leader) during training. According to Figure 3.22, the aggregation step time is small at the beginning of the training and increases as the SNN models converge to the final solution. The rate of change of the norm of the synaptic weights determines the communication sample time of the aggregation process, which results in small aggregation intervals when the change rate is high and larger intervals when it reduces.

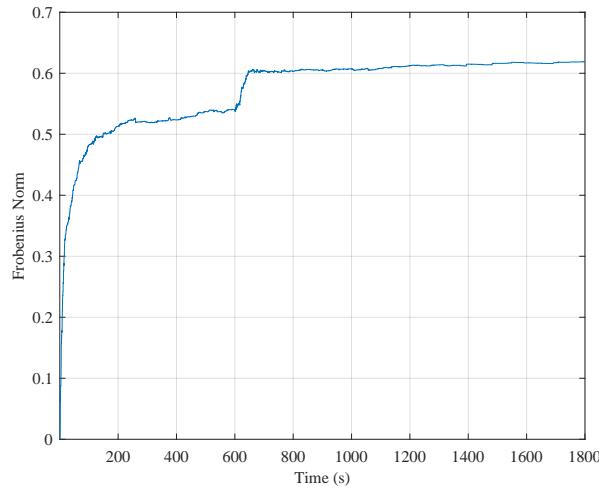


Figure 3.21: Frobenius norm of the Agent 1's weighs during the learning phase. The reward changes for the Leader after 600 s.

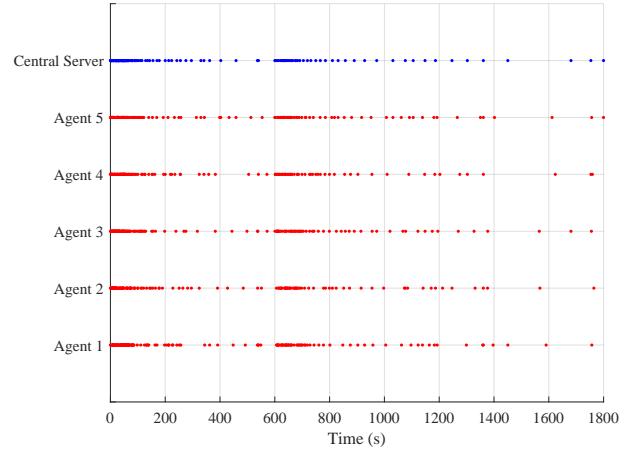


Figure 3.22: Communication times for agents and the Central Server (Leader). Red and blue dots show the times that agents and the Central Server have sent their model, respectively.

Therefore, the aggregation frequency is very high initially, and it reduces after the change in synaptic weights goes to zero because the learning rate matrix in (3.12) goes to zero. Also, after the reward changes and the leader becomes an obstacle after 600 seconds, the Euclidean distance between the converged and current models increases and reaches the threshold. As soon as the first agent sends its model to the server, the aggregation process starts again, and the agents adjust the associated synaptic weight.

Table 3.3: Comparative Performance Analysis of the Proposed Aggregation Algorithm+RCSE and RCSE

	<b>FL+RCSE</b>	<b>RCSE</b>
<b>Learning (Convergence) Time - Training Phase (s)</b>	261.92	554.71
<b>Max Distance Convergence Error - Test Phase (%)</b>	0.71	1.95
<b>Convergence Time for Distance - Test Phase (s)</b>	5.81	8.94

Table 3.3 compares the results and focuses on three critical metrics: learning time, maximum error after convergence, and convergence time. The proposed FL algorithm demon-

strates significant improvements in terms of efficiency and accuracy, as evidenced by its considerably shorter learning and convergence times and a notable reduction in error after convergence.

## 3.4 Conclusion

In this chapter, we presented a comprehensive approach addressing the challenges of uncontrolled growth in synaptic weights and the limited responsiveness of R-STDP to real-time changes within SNNs. Our proposed solution integrates the RCSE method with a dynamic aggregation interval in FL and significantly reduces learning time while improving performance.

The R-CSE method introduces a novel mechanism to manage the unbounded growth of synaptic weights by dynamically adjusting the decay rate through the SoftPlus function. This adjustment is sensitive to the learning stages and rewards changes, ensuring that synaptic weight adjustments remain responsive over time. By addressing the challenge of synaptic weight saturation, the RCSE method facilitates a balanced approach to weight adjustment, preventing network saturation and promoting continuous learning adaptability.

We introduce a novel approach that uses FL in SNN and employs the Frobenius norm to adjust weighted aggregation in FL. Additionally, we include weight decay proportional to the time elapsed since an agent's last model publication. This improves the efficiency and responsiveness of the learning process.

Our proposed method adjusts its aggregation time based on the Euclidean norm. This metric measures the distance between the weight matrices of the agents and the server, determining reduced intervals for model publication. Our results show that the proposed aggregation method significantly accelerates agents' learning while reducing the distance convergence error (Table 3.3).

Moreover, the dynamic aggregation interval effectively reduces communication overhead between the agents and the central server, particularly after model converges and the algorithm switches from the training mode to the operational mode. This reduction is critical

when communication bandwidth is limited or costly. This approach is particularly advantageous in 5G networks, where efficient bandwidth use can enhance the overall throughput and reduce latency in real-time applications. Moreover, the adaptive use of communication resources aligns with the scalable and flexible infrastructure of 5G, optimizing network performance even during peak demand periods.

# Chapter 4

## Vision-based Multi-Agent Autonomous Docking Using a Deep Spiking Neural Network and Reward-modulated Spike-Timing-Dependent Plasticity

### 4.1 Preliminaries

#### 4.1.1 Problem definition: Docking Missions with Swarm Robots

Swarm robotics, inspired by the collective behaviors observed in natural systems such as ant colonies, bee swarms, and flocks of birds, represents a promising paradigm for creating intelligent robotic systems capable of performing complex, decentralized tasks. These systems are typically characterized by a large number of relatively simple robots that collaborate and coordinate their actions without reliance on any form of centralized control. This decentralized approach offers inherent advantages, including robustness to individual robot failures, flexibility in adapting to changing environments, and scalability to large numbers of agents [71].

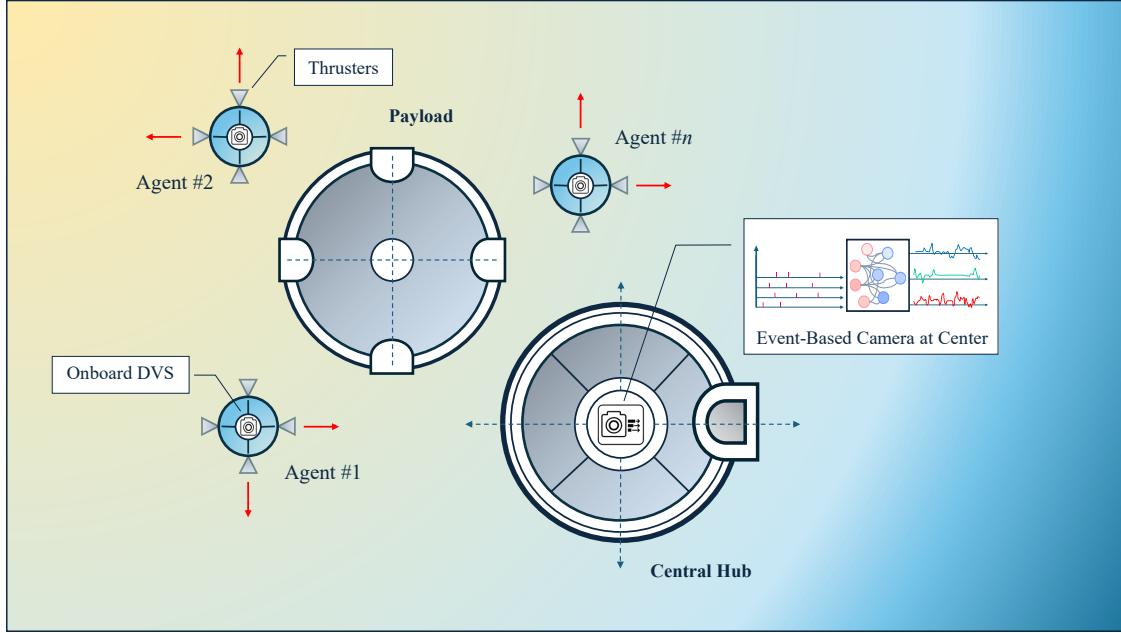


Figure 4.1: Swarm of robots performing docking mission. Each agent pushes the Payload towards the Central Hub to align them together. The central Hub and agents are equipped with a Dynamic Vision Sensor (DVS) that streams high-frequency data for the proximity mission.

The desired collective behavior of the swarm emerges from the local interactions among the individual robots and between the robots and their environment. This paradigm contrasts with traditional robotic systems, which often rely on a central controller to dictate the actions of individual robots. The term “docking missions,” within the context of swarm robotics, encompasses a range of cooperative tasks where multiple robots need to achieve a state of connection or close proximity. This includes tasks such as rendezvous, where the objective is for multiple robots to converge at a common location or maintain a defined spatial relationship; assembly, where robots collaboratively work to form a desired structure or connect with other robots or objects to create a larger, functional entity.

In the specific application of this study, the swarm agents are tasked with pushing a payload towards a central hub to achieve a precise docking alignment. Each agent is equipped with thrusters allowing them to move in a 2D space, and a Dynamic Vision Sensor (DVS)

mounted onboard provides them with visual feedback about the environment and neighboring agents. Additionally, a DVS camera is mounted at the center of the central hub, offering a global event-based view of the payload and the agents during the docking process.

The primary objective of this paper is to develop a decentralized vision-based swarm docking system utilizing Deep SNNs trained with R-STDP, which leverages the asynchronous, sparse, and high-temporal-resolution output of DVS cameras to guide the swarm behavior effectively. Each agent in the swarm runs its own local SNN onboard, processing sensory information independently and learning control policies in a distributed manner, which allows the swarm to remain scalable and robust to individual agent failures.

In this environment, Active LED Marker (ALM)s are installed on both the payload and the agents to enable robust object detection and differentiation within the event-based vision system. Each ALM transmits light at a specified frequency but with unique phase offsets assigned to the payload and each agent. This phase-based encoding ensures that at each detection interval, only one object's ALM is illuminated, allowing the system to isolate and identify the payload or specific agents without ambiguity in the DVS data stream. This structured blinking strategy facilitates clean separation of objects in the spatiotemporal event data, enabling the network to distinguish between the payload and agents during the docking mission while maintaining the asynchronous and high-temporal-resolution advantages of event-based sensing.

When the agents are flying and have not made contact with the payload, they rely solely on their onboard DVS for environment perception, capturing events corresponding to nearby agents and the payload. Once an agent makes physical contact with the payload, its onboard force contact sensor activates, and the onboard DVS data is replaced by the central hub DVS, providing a more comprehensive and centralized perspective of the payload and the surrounding agents.

### 4.1.2 Event-Based Camera Simulation

This section presents the simulation methodology for modeling brightness-based visual input and generating asynchronous events for SNNs using DVS. Two distinct DVS camera models are used in this study: one mounted on each agent and one placed at a fixed position on the central hub. Both cameras independently perceive the environment and produce separate event streams based on the change in perceived brightness over time.

The environment is populated by multiple agents and a payload object, each modeled as circular shapes with defined physical properties such as radius and centroid location. Each DVS camera discretizes its field of view into a grid of  $N_y \times N_x$  pixels, where each pixel corresponds to a fixed location in the simulated physical space. The spatial mapping between the continuous world coordinates and the discrete DVS pixel grid is defined through a static transformation.

For each DVS view, a brightness map is generated at every simulation time step. Let  $X_w$  and  $Y_w$  represent the 2D coordinate meshgrids over the DVS image plane in world units. The initial brightness grid  $E(x, y)$  is set to a constant background value  $I_{\text{bg}}$  across all pixels. The contribution of each object (agent or payload) to the brightness grid is computed by rendering its circular projection into the grid and modulating its intensity based on relative spatial orientation and distance to one or more luminous sources.

Let  $(x_o, y_o)$  denote the position of an object with radius  $r_o$ . For each pixel with world coordinates  $(x_w, y_w)$ , we first compute the squared Euclidean distance to the object center:

$$d^2 = (x_w - x_o)^2 + (y_w - y_o)^2. \quad (4.1)$$

Pixels satisfying  $d^2 \leq r_o^2$  are considered to lie within the object's projection. For these pixels, brightness is calculated based on Lambertian reflectance with multiple light sources. Let  $(x_s^{(l)}, y_s^{(l)})$  denote the position of the  $l$ -th light source. We define the vector from the

pixel to the light source as:

$$\mathbf{v}_{\text{light}}^{(l)} = \begin{bmatrix} x_w - x_s^{(l)} \\ y_w - y_s^{(l)} \end{bmatrix}, \quad \|\mathbf{v}_{\text{light}}^{(l)}\| = \sqrt{(x_w - x_s^{(l)})^2 + (y_w - y_s^{(l)})^2}. \quad (4.2)$$

Similarly, the vector from the pixel to the object center is:

$$\mathbf{v}_{\text{center}} = \begin{bmatrix} x_o - x_w \\ y_o - y_w \end{bmatrix}, \quad \|\mathbf{v}_{\text{center}}\| = \sqrt{(x_o - x_w)^2 + (y_o - y_w)^2}. \quad (4.3)$$

The angle  $\theta^{(l)}$  between these two vectors is used to compute the cosine-based brightness contribution from light source  $l$ :

$$\theta^{(l)} = \arccos \left( \frac{\mathbf{v}_{\text{light}}^{(l)} \cdot \mathbf{v}_{\text{center}}}{\|\mathbf{v}_{\text{light}}^{(l)}\| \cdot \|\mathbf{v}_{\text{center}}\| + \varepsilon} \right), \quad (4.4)$$

$$I^{(l)} = \max(0, \cos(\theta^{(l)}) \cdot 0.5 + 0.5), \quad (4.5)$$

where  $\varepsilon$  is a small constant to avoid division by zero. The final brightness value at each pixel is computed as the maximum contribution across all light sources and overlapping objects:

$$E(x_w, y_w) = \max \left( I_{\text{bg}}, \max_{\text{objects}} \max_l I^{(l)} \right). \quad (4.6)$$

Once the brightness map  $E(x, y)$  is generated, it is passed to the event generation module that models the behavior of the DVS sensor. To emulate the logarithmic response of biological photoreceptors, the brightness is first clamped to a minimum value and transformed into logarithmic intensity:

$$E_{\text{clamped}}(x, y) = \max(E(x, y), I_{\min}), \quad (4.7)$$

$$\mathcal{L}_{\text{current}}(x, y) = \ln(E_{\text{clamped}}(x, y)). \quad (4.8)$$

The temporal contrast is computed as the difference between the current and previously

stored logarithmic intensity:

$$\Delta\mathcal{L}(x, y) = \mathcal{L}_{\text{current}}(x, y) - \mathcal{L}_{\text{prev}}(x, y). \quad (4.9)$$

This change in intensity is integrated into a membrane potential variable  $V(x, y)$  for each pixel:

$$V_t(x, y) = V_{t-1}(x, y) + \Delta\mathcal{L}(x, y), \quad (4.10)$$

subject to optional clipping to prevent numerical instability.

The DVS event generation rule triggers ON and OFF events based on threshold crossing behavior. Define thresholds  $\vartheta_{\text{on}} > 0$  and  $\vartheta_{\text{off}} > 0$ . For each pixel, if:

$$\begin{cases} \text{if } V(x, y) \geq \vartheta_{\text{on}} : & e(x, y) = +1, \quad V(x, y) \leftarrow V(x, y) - \vartheta_{\text{on}}, \\ \text{if } V(x, y) \leq -\vartheta_{\text{off}} : & e(x, y) = -1, \quad V(x, y) \leftarrow V(x, y) + \vartheta_{\text{off}}. \end{cases} \quad (4.11)$$

Otherwise, no event is generated and  $e(x, y) = 0$ . These updates are performed for both agent-mounted and hub-mounted DVS views independently, using their respective brightness maps and potential histories. The  $\mathcal{L}_{\text{current}}$  map is stored as  $\mathcal{L}_{\text{prev}}$  for the next time step.

This detailed simulation allows each agent to perceive a localized view of the environment while the central hub maintains a global perspective.

### 4.1.3 Dynamic Model of the Payload and Agents

Each agent in the swarm is modeled as a point mass of mass  $m$  and radius  $\mathcal{R}$ . The dynamic state of each agent includes its position, velocity, thrust-based control forces, and interaction forces arising from collisions with other agents and the payload. Additionally, a central payload object with its own dynamics is influenced by contacts with the agents.

The state of agent  $i$  at time  $t$  is defined as  $\mathbf{p}_i(t) = \begin{bmatrix} X_i(t) \\ Y_i(t) \end{bmatrix}$  and  $\mathbf{v}_i(t) = \begin{bmatrix} V_{x_i}(t) \\ V_{y_i}(t) \end{bmatrix}$ , and the control thrusts along the X and Y axes are derived from spike activity in the output layer of a SNN. Agents and the payload may collide with each other, and these interactions

are modeled using a spring-damper system. Let object  $a$  with position  $\mathbf{p}_a$ , velocity  $\mathbf{v}_a$ , and radius  $R_a$  interact with object  $b$  with position  $\mathbf{p}_b$ , velocity  $\mathbf{v}_b$ , and radius  $R_b$ . The collision force between objects  $a$  and  $b$  is computed as:

$$\delta_{ab} = R_a + R_b - \|\mathbf{p}_a - \mathbf{p}_b\|, \quad (4.12)$$

$$\mathbf{n}_{ab} = \frac{\mathbf{p}_a - \mathbf{p}_b}{\|\mathbf{p}_a - \mathbf{p}_b\|}, \quad (4.13)$$

$$\mathbf{v}_{ab}^{rel} = \mathbf{v}_a - \mathbf{v}_b, \quad (4.14)$$

$$\mathbf{F}_{ab}^{\text{coll}} = k \cdot \delta_{ab} \cdot \mathbf{n}_{ab} + c \cdot (\mathbf{v}_{ab}^{rel} \cdot \mathbf{n}_{ab}) \cdot \mathbf{n}_{ab}, \quad (4.15)$$

where  $k$  and  $c$  are the spring and damping coefficients, respectively. The collision force  $\mathbf{F}_{ab}^{\text{coll}}$  is applied to object  $a$  while an equal and opposite force  $-\mathbf{F}_{ab}^{\text{coll}}$  is applied to object  $b$ . This unified formulation applies to all pairwise interactions within the system, including agent-agent and agent-payload collisions, enabling consistent force computation across the entire environment.

The total force applied to each object  $a$  in the system, which can represent either an agent or the payload, is defined as the sum of all interaction forces from other objects and the control thrust applied to the object. The total force is given by:

$$\mathbf{F}_a^{\text{total}}(t) = -\mathbf{F}_a^{\text{thrust}}(t) + \sum_{b \neq a} \mathbf{F}_{ab}^{\text{coll}}(t), \quad (4.16)$$

where  $\mathbf{F}_a^{\text{thrust}}(t)$  denotes the applied control thrust on object  $a$  generated by the SNN. The continuous-time motion of each object  $a$  is then governed by Newton's second law:

$$\frac{d^2 \mathbf{p}_a(t)}{dt^2} = \frac{\mathbf{F}_a^{\text{total}}(t)}{m_a}, \quad (4.17)$$

where  $\mathbf{p}_a(t)$  denotes its position at time  $t$ .

These equations fully describe the continuous-time dynamics of all agents and the payload within the environment, capturing the adaptive response of each object to control inputs and

collision forces with other objects in the system. This unified formulation ensures consistent and scalable multi-object simulation, supporting distributed control and local collision handling across all components in the environment.

#### 4.1.4 Neuron model

Biologically inspired spiking neuron models often exhibit rich dynamical behaviors such as excitability, bursting, and bistability. The well-known Izhikevich model [23] strikes a balance between biophysical realism and computational efficiency, using only two ordinary differential equations plus a reset condition. The standard Izhikevich model is given by

$$\begin{cases} \frac{dv}{dt} = 0.04 v^2 + 5v + 140 - u + I(t), \\ \frac{du}{dt} = a(bv - u), \end{cases} \quad (4.18)$$

with a reset rule:

$$\text{if } v \geq 30 \text{ mV, then } \begin{cases} v \leftarrow c, \\ u \leftarrow u + d. \end{cases} \quad (4.19)$$

where  $a$  and  $b$  are the neuron parameters,  $v$  is the membrane potential (in mV),  $u$  is a slow recovery variable,  $I(t)$  is the external input current, and  $c$  and  $d$  define how the system is reset after a spike. Building on the neuron model and initialization strategies described in the Preliminaries, we now present the structured architecture and training scheme for the proposed deep SNN.

#### Random Synaptic Weight Initialization

The initial synaptic weight matrix  $S \in \mathbb{R}^{N_{\text{post}} \times N_{\text{pre}}}$  is created based on the binary connectivity matrix  $\mathcal{C} \in \{0, 1\}^{N_{\text{post}} \times N_{\text{pre}}}$ . Each element of  $S$  is given by

$$S(i, j) = \begin{cases} w_{ij}, & \mathcal{C}(i, j) = 1, \\ 0, & \text{otherwise,} \end{cases} \quad (4.20)$$

so that non-existent connections remain zero.

A scalar parameter  $p_{\text{active}} \in (0, 1]$  (default value 0.10) sets the fraction of total incoming weight that may be active at each time step. If  $\mathcal{W}_{\text{in}}$  denotes the maximum input strength for a given postsynaptic neuron, then the threshold on the sum of raw weights is

$$\Theta = \frac{\mathcal{W}_{\text{in}}}{p_{\text{active}}}. \quad (4.21)$$

The algorithm proceeds layer by layer. For layer index  $\ell = 2, \dots, n_L$ , let  $\mathcal{P}_\ell$  be the set of postsynaptic neuron indices in layer  $\ell$  and  $\mathcal{Q}_{\ell-1}$  the set of presynaptic indices in the previous layer. To generate a spatially smooth random gradient over the presynaptic population, the  $N = |\mathcal{Q}_{\ell-1}|$  presynaptic indices are embedded onto a two-dimensional grid of size  $\sqrt{N} \times \sqrt{N}$ , and the grid coordinates are

$$\{(x_k, y_k)\}_{k=1}^N \subset \{1, \dots, \sqrt{N}\}^2. \quad (4.22)$$

A random center

$$\boldsymbol{\mu} = \left( \frac{\sqrt{N+1}}{2}, \frac{\sqrt{N+1}}{2} \right) + \delta, \quad \delta \sim \mathcal{U}([-0.5 \sqrt{N}, 0.5 \sqrt{N}]^2), \quad (4.23)$$

and width  $\sigma = \sqrt{N}/3$  define a Gaussian kernel with additive uniform noise:

$$g_k = \exp\left(-\frac{(x_k - \mu_x)^2 + (y_k - \mu_y)^2}{2\sigma^2}\right) + \eta_k, \quad \eta_k \sim \mathcal{U}([-0.5, 0.5]). \quad (4.24)$$

Normalizing  $g_k$  by its maximum,

$$\bar{g}_k = \frac{g_k}{\max\{g_k\}}, \quad (4.25)$$

yields values in  $[0, 1]$ . These are then mapped into the allowed weight range by

$$w_k = W_{\min} + (W_{\max} - W_{\min}) \bar{g}_k. \quad (4.26)$$

For each postsynaptic neuron  $i \in \mathcal{P}_\ell$ , let  $\mathcal{I}_i \subset \{1, \dots, N\}$  be the indices of grid points

corresponding to nonzero connections. The raw incoming weights are  $\{w_j\}_{j \in \mathcal{I}_i}$ . If their sum exceeds the threshold  $\Theta$ , they are uniformly rescaled:

$$\text{if } \sum_{j \in \mathcal{I}_i} w_j > \Theta, \quad w_j \leftarrow w_j \frac{\Theta}{\sum_{k \in \mathcal{I}_i} w_k}, \quad j \in \mathcal{I}_i. \quad (4.27)$$

Finally, these weights are placed into the initialization matrix by

$$S(i, \mathcal{Q}_{\ell-1}[j]) = w_j, \quad j \in \mathcal{I}_i, \quad (4.28)$$

ensuring that each neuron's incoming synaptic strengths obey both the spatial gradient structure and the total-input constraint.

### Analytic Determination of Inhibitory Synaptic Weight

We consider a Class 1 Izhikevich neuron receiving an excitatory drive  $I_{\text{exc}}(t)$  and a fixed inhibitory pulse  $I_{\text{inh}}(t)$ . Spiking ceases when the  $v$ -nullcline in (4.18)

$$0.04 v^2 + 5v + 140 - u + I = 0 \quad (4.29)$$

collides with its extremum, i.e. when

$$\frac{\partial}{\partial v} (0.04 v^2 + 5v + 140 - u + I) = 0 \implies 0.08 v + 5 = 0. \quad (4.30)$$

Thus

$$v^* = -\frac{5}{0.08} = -62.5, \quad u^* = b v^*. \quad (4.31)$$

Substituting back, the total critical current  $I_{\text{crit}}$  satisfies

$$0.04 (v^*)^2 + 5 v^* + 140 - u^* + I_{\text{crit}} = 0 \implies I_{\text{crit}} = u^* - 0.04 (v^*)^2 - 5 v^* - 140. \quad (4.32)$$

For the canonical Class 1 choice  $a = 0.02$ ,  $b = -0.1$ ,  $c = -55$ ,  $d = 6$ , one finds

$$v^* = -62.5, \quad u^* = b v^* = 6.25, \quad (4.33)$$

$$0.04 (v^*)^2 = 156.25, \quad 5 v^* = -312.5, \quad (4.34)$$

$$I_{\text{crit}} = 6.25 - 156.25 + 312.5 - 140 = 22.5. \quad (4.35)$$

Because each spike increments  $u$  by  $d$ , the inhibitory weight must compensate for both  $I_{\text{crit}}$  and this adaptation. Therefore we choose

$$w_{\text{inh}} = I_{\text{crit}} + d = 22.5 + 6 = 28.5, \quad (4.36)$$

so that an inhibitory pulse of amplitude  $-28.5$  silences the neuron. Empirically, we confirmed that  $w_{\text{inh}} = -28.5$  immediately vetoes spiking and allows rapid recovery when released.

## 4.2 Proposed Neuromorphic Framework

### 4.2.1 Deep SNN Structure

The neural network implemented in this study consists of four layers: an input layer that processes sensory information from a DVS, the first hidden layer that learns the visual perception, the second hidden layer that manages the mission phases and separates the policies for formation flying and docking phase, and an output layer responsible for generating control signals to actuate the motors. Each of these layers employs a distinct parameterization of the Izhikevich neuron model to achieve its respective function within the network.

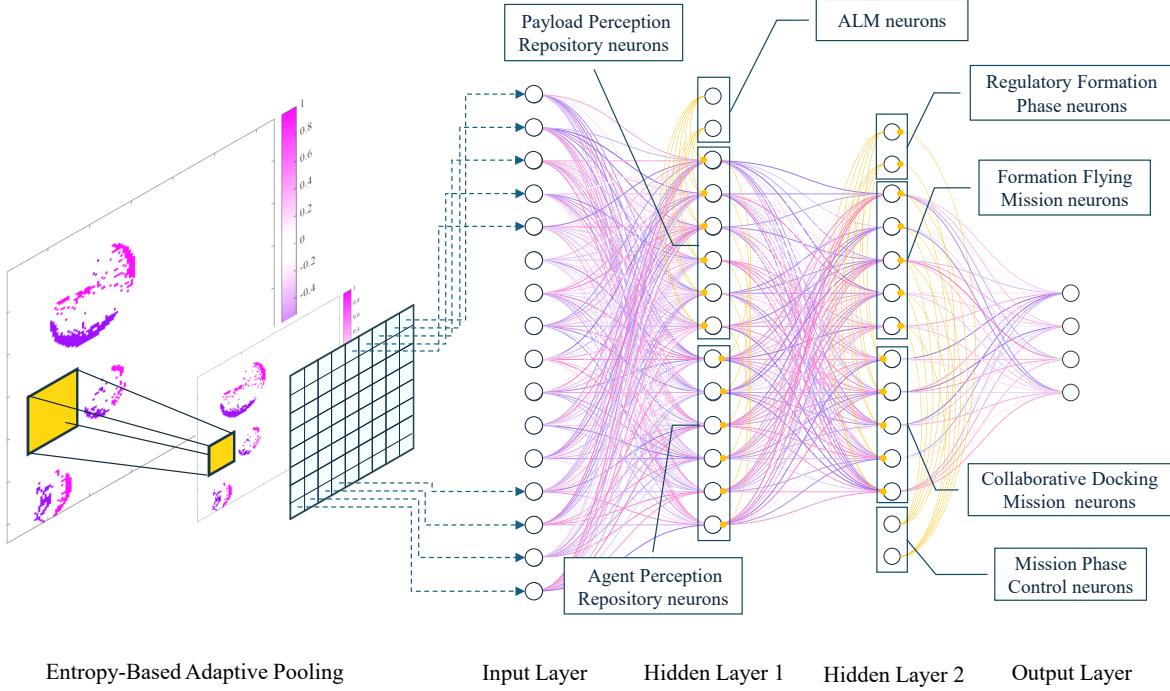


Figure 4.2: Deep SNN Architecture with Entropy-Based Adaptive Pooling and Structured Hidden Layers for Swarm Robotics. The purple connections are excitatory, and the yellow connections are inhibitory.

In this study, we design a deep SNN architecture tailored for swarm robotics applications requiring real-time perception, decision-making, and control under strict energy and computational constraints. The network architecture comprises an input layer, two structured hidden layers, and an output layer, each utilizing Izhikevich neuron models to capture biologically realistic spiking dynamics while maintaining computational efficiency. The input layer, containing bistable neurons, processes high-frequency event-based data from DVS, maintaining stimulus-relevant activity patterns over time. The two hidden layers are structured to facilitate mission-phase switching and context-aware policy learning, with the first hidden layer incorporating ALM neurons to create an inhibition mechanism for payload-agent channel separation, while the second layer integrates Mission Phase Switch (MPS) and Mission Phase Control (MPC) neurons for phase-gated control and stability during mission transitions. Finally, the output layer, consisting of four spiking neurons, generates directional

control signals in the  $x$  and  $y$  axes, enabling adaptive motor control based on the learned spiking activity patterns.

This architecture supports distributed onboard training using R-STDP, allowing each agent to learn effective control policies directly from event-based sensory input in a scalable and robust manner.

## Input Layer

The input layer of the proposed SNN receives event-based visual input from a DVS, which generates sparse spatiotemporal signals represented as discrete events: +1 for ON events (increase in brightness), -1 for OFF events (decrease in brightness), and 0 for no activity. Let  $X \in \{-1, 0, +1\}^{n \times n}$  denote the event matrix at a given time step or after temporal accumulation. Due to the high resolution and sparsity of DVS data, an adaptive spatial reduction mechanism is applied prior to encoding into spiking neurons.

**Entropy-Based Adaptive Pooling** To reduce dimensionality while preserving information-rich regions, we implement an entropy-guided pooling method. The matrix  $X$  is partitioned into regions  $\Omega$ , each corresponding to a pixel in the downsampled output. For a region  $\Omega$ , we compute the frequency of event categories:

$$P(c) = \frac{1}{|\Omega|} \sum_{(i,j) \in \Omega} \mathbb{I}(X_{i,j} = c), \quad \text{for } c \in \{-1, 0, +1\}. \quad (4.37)$$

where,  $\mathbb{I}$  is the indicator function. The Shannon entropy of region  $\Omega$  is then calculated as:

$$H(\Omega) = - \sum_{c \in \{-1, 0, +1\}} P(c) \log_2 P(c), \quad (4.38)$$

excluding terms where  $P(c) = 0$ . This entropy quantifies the diversity of event types in the region, with  $H(\Omega) = 0$  indicating uniform activity and  $H_{\max} = \log_2(3) \approx 1.585$  indicating maximum variability.

The entropy value is then mapped to a pooling kernel size  $k \in [k_{\min}, k_{\max}]$ , using a linear

transformation:

$$k(H) = k_{\max} - (k_{\max} - k_{\min}) \cdot \frac{H(\Omega)}{H_{\max}}. \quad (4.39)$$

Low-entropy (homogeneous) regions are pooled using a larger kernel  $k_{\max}$ , while high-entropy (heterogeneous) regions are pooled more finely. For each subregion, we apply sign-preserving max pooling by selecting the element with the highest absolute value. The resulting values form a downsampled matrix  $X' \in \{-1, 0, +1\}^{n' \times n'}$ , where  $n' < n$ .

**Current Encoding and Input Neuron Configuration** Each element  $X'_{i,j}$  in the pooled matrix is mapped to an input current  $I_{i,j}$  via a transformation function  $\Gamma(\cdot)$ :

$$I_{i,j} = \Gamma(X'_{i,j}), \quad (4.40)$$

where  $\Gamma$  maps the discrete event values into continuous current levels suitable for the neuron model. For example, ON events may produce a positive input current, OFF events a negative current, and silent regions yield no current. The input layer consists of Bistable spiking neurons, each implemented using the Izhikevich neuron model operating in the bistable regime. This configuration allows neurons to remain in a stable spiking or resting state, enabling persistent memory of recent inputs. At steady state, setting  $dv/dt = du/dt = 0$  in the Izhikevich model leads to the following fixed-point equations:

$$u = bv, \quad (4.41)$$

$$0.04v^2 + (5 - b)v + (140 + I) = 0. \quad (4.42)$$

The solutions  $v^*$  to the quadratic equation represent the possible membrane potentials at which the neuron may remain at equilibrium. The number and nature of these solutions

depend on the discriminant of the quadratic:

$$\Delta = (5 - b)^2 - 4 \cdot 0.04 \cdot (140 + I). \quad (4.43)$$

When  $\Delta > 0$ , there are two distinct real roots, indicating the existence of two equilibrium points. If one of these is a stable resting state and the other is associated with an unstable point surrounded by a stable limit cycle, the neuron exhibits bistability. This means it can remain in a quiescent (non-spiking) state or transition into repetitive spiking depending on the input and initial conditions.

To examine the local stability of each equilibrium point  $(v^*, u^*)$ , the system is linearized using the Jacobian matrix:

$$J(v^*, u^*) = \begin{pmatrix} \frac{\partial}{\partial v}(0.04v^2 + 5v + 140 - u + I) & \frac{\partial}{\partial u}(0.04v^2 + 5v + 140 - u + I) \\ \frac{\partial}{\partial v}(a(bv - u)) & \frac{\partial}{\partial u}(a(bv - u)) \end{pmatrix} = \begin{pmatrix} 0.08v^* + 5 & -1 \\ ab & -a \end{pmatrix}. \quad (4.44)$$

The trace and determinant of the Jacobian are given by:

$$\text{Tr}(J) = 0.08v^* + 5 - a = 0.08v^* + 4, \quad \det(J) = -a(0.08v^* + 5) + ab = -0.08v^* - 3.5. \quad (4.45)$$

The stability of the equilibrium point depends on the sign of these values. If  $\text{Tr}(J) < 0$  and  $\det(J) > 0$ , then the equilibrium is stable, and if  $\text{Tr}(J) > 0$  or  $\det(J) < 0$ , then the equilibrium is unstable. Because both trace and determinant depend on the specific value of  $v^*$ , it is possible for one root to correspond to a stable resting state (e.g.,  $v^* \approx -60$  mV) while the other leads to an unstable point surrounded by a spiking limit cycle. This configuration underlies the bistable behavior of the neuron, allowing it to encode information persistently across time depending on the strength and polarity of the injected current.

Each neuron receives input from the preprocessed DVS event matrix, where ON events (+1) and OFF events (-1) are mapped to distinct neuron indices. Neurons corresponding to ON events are injected with maximum excitatory current  $I = 100$  nA, and those cor-

responding to OFF events receive strong inhibitory input  $I = -80\text{ nA}$ . All other neurons maintain a baseline current of  $I = -65\text{ nA}$ , representing a resting or non-stimulated state. This behavior is encoded as:

$$I_i(t) = \begin{cases} 100\text{ nA}, & \text{if } X'_i = +1, \\ -80\text{ nA}, & \text{if } X'_i = -1, \\ -65\text{ nA}, & \text{otherwise.} \end{cases} \quad (4.46)$$

This input mapping is crucial in shaping the dynamic behavior of bistable neurons. In the case of an ON event, the neuron receives a transient positive input current that is applied for a single simulation step. However, due to the bistable nature of the model, the neuron does not immediately return to rest after the stimulus disappears. Instead, it remains in an active spiking state, continuously firing until it is explicitly reset by an OFF event. This reset occurs when the same neuron index receives an inhibitory current corresponding to a negative event in the DVS stream.

This mechanism enables the input layer to function as a short-term memory system. Even in the absence of new motion or visual changes in the environment, neurons that were previously excited by object movement continue to spike, thereby maintaining a persistent representation of object locations. As a result, the network retains awareness of the last known positions of salient features, making it especially suitable for processing sparse, asynchronous data and tracking static or intermittently visible targets.

## Hidden Layers

### First Hidden Layer for Visual Processing of DVS Events

The first hidden layer of the SNN is designed to perform early-stage visual feature extraction, dimensionality reduction, and fuzzy separation of the environment while incorporating ALM-based attention and gating mechanisms. This layer contains three repositories of neurons

that collectively process the preprocessed DVS input to distinguish between the payload and neighboring agents while encoding these features in a fuzzy spatial representation.

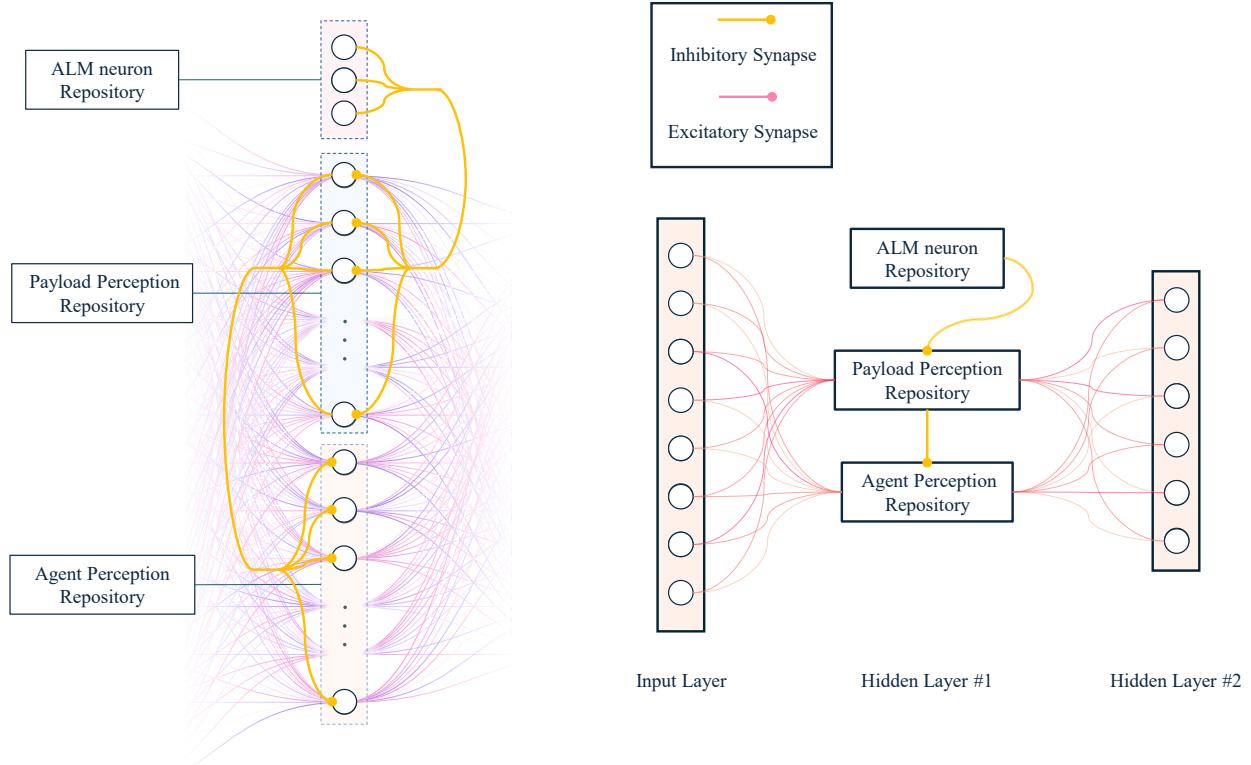


Figure 4.3: Structured First Hidden Layer Architecture with ALM, Payload, and Agent Perception Repositories for Visual Perception.

All neurons in this layer utilize the Izhikevich neuron model configured for Class 1 excitability, enabling the neurons to modulate their spiking frequency smoothly in response to the strength of the injected input currents. To implement ALM-based attention, the ALM neurons in the layer are configured for tonic bursting to ensure persistent spiking during activation windows. For these neurons, the same initialization of the membrane potential and recovery variables ensures consistent readiness for event-driven activation.

The ALMs are mounted on the payload, and each agent transmits light periodically at a frequency designed for phase-shifted activation patterns. The payload's ALM is configured

to blink with a period  $T_P = 20$  ms, remaining active for  $T_P^{\text{on}} = 10$  ms with a phase offset of  $\phi_P = 0$  ms. Each agent's ALM operates with the same period and activation duration but with a phase offset of  $\phi_A = 10$  ms relative to the payload, ensuring that the agents' ALMs are active precisely when the payload's ALM is inactive. This relationship can be expressed as:

$$\text{Active}_{\text{ALM}}(t) = \begin{cases} 1, & \text{if } \langle(t - \phi, T) \rangle < T^{\text{on}}, \\ 0, & \text{otherwise,} \end{cases}$$

where  $\phi$  denotes the phase offset for either the payload or agent.

Each time the agent's ALM activates, the ALM neurons in the first hidden layer receive input currents and fire spikes. To ensure that these neurons maintain activity throughout the ALM's active window and beyond, the stimulation provided to the ALM neurons is randomized within the activation interval, ensuring a distribution of spike timings that prevents synchronous inactivity between time steps. This mechanism supports consistent gating activity, enabling robust attention control within the spiking network.

Functionally, the first hidden layer separates the payload position channel and the neighboring agents' position channel, representing them in a fuzzy spatial format using the spiking activity of its neurons. This fuzzy coding is achieved by mapping the processed DVS input onto the neurons while allowing the network to maintain partial activation across spatial regions, enabling smooth transitions between detected object configurations.

Additionally, the first hidden layer performs dimensionality reduction by compressing the high-resolution DVS input into a lower-dimensional representation suitable for the subsequent layers while preserving the essential features necessary for object tracking and environmental understanding. This reduction is crucial for the scalability of the network and for enabling efficient computation during online learning and control.

The spiking patterns generated within this layer create unique states in the network, allowing the system to distinguish different object configurations in the environment robustly. During operation, when the agents' ALMs are active, the corresponding ALM neurons in the

first hidden layer fire and inhibit the payload detection repository. This gating ensures that during the weight update phase of the network, only the connections associated with the agent detection repository are updated. When the agents' ALMs turn off and the payload's ALM activates, the payload detection neurons fire, and the connections associated with the payload detection repository are updated in the subsequent weight update phase.

Through this mechanism, the first hidden layer dynamically manages the flow of information based on the phase-synchronized ALM signals, ensuring clear separation between agent and payload representations, maintaining fuzzy spatial coding, reducing input dimensionality, and creating structured, distinguishable states for downstream processing in the SNN.

### Second Hidden Layer for Mission Phase Switching

The second hidden layer facilitates mission-phase switching and policy learning by leveraging structured neuron repositories that enable the network to autonomously transition between formation flying and docking phases while learning appropriate control policies through spiking activity.

Within this layer, Formation Flying Mission (FFM) neurons, MPS neurons, MPC neurons, and Collaborative Docking Mission (CDM) neurons are interconnected to enforce phase-gated learning. The FFM and CDM neurons act as repositories through which synaptic connections to the output layer are routed, enabling phase-specific learning by ensuring that only active pathways during a particular mission phase undergo synaptic plasticity.

The neurons within this layer employ models capable of producing persistent spiking, bistability, and rapid state transitions required for dynamic mission control. The FFM and CDM neurons are configured to exhibit tonic spiking behavior to sustain stable outputs during their respective active phases. In contrast, the MPS neurons exhibit bistable firing patterns, enabling them to maintain persistent activity while providing controlled inhibition during phase transitions. The MPC neurons are designed to generate rapid, transient spiking patterns that effectively trigger phase switches by inhibiting the active repositories when

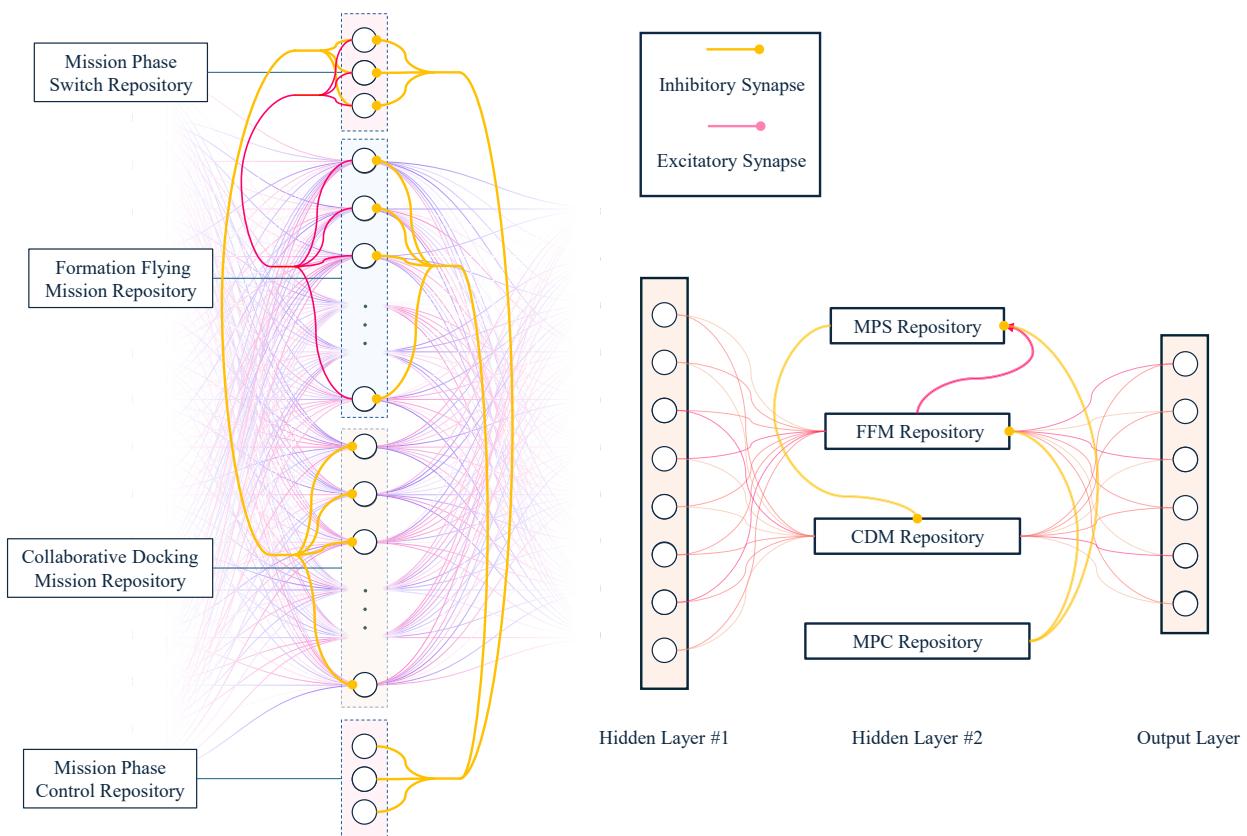


Figure 4.4: Structured Second Hidden Layer Architecture with Mission-Phase and Regulatory Neuron Repositories.

mission conditions change.

During *formation flying*, the agent has not contacted the payload, and the FFM neurons maintain persistent spiking activity. Let the activity vector of the FFM neurons at time  $t$  be denoted by

$$\mathbf{s}_{\text{FFM}}(t) \in \{0, 1\}^{N_{\text{FFM}}}, \quad (4.47)$$

where  $N_{\text{FFM}}$  is the number of FFM neurons and  $s_{\text{FFM},i}(t) = 1$  if the  $i$ -th neuron spikes at time  $t$ . To ensure continuous gating, the FFM neurons receive random excitatory current  $I_{\text{rand}}(t)$  to maintain consistent spiking, leading to persistent inhibition of the CDM neurons.

The inhibition of CDM neurons by MPS neurons is modeled as

$$W_{ij}^{\text{CDM} \leftarrow \text{RFP}} = -\gamma, \quad (4.48)$$

where  $\gamma > 0$ , and the resulting inhibitory current is

$$I_j^{\text{inh}}(t) = \sum_i W_{ij}^{\text{CDM} \leftarrow \text{RFP}} s_{\text{RFP},i}(t). \quad (4.49)$$

Simultaneously, the FFM neurons provide excitatory input to the MPS neurons to sustain their spiking:

$$W_{ij}^{\text{RFP} \leftarrow \text{FFM}} = \delta, \quad (4.50)$$

where  $\delta > 0$ , producing the excitatory current

$$I_j^{\text{exc}}(t) = \sum_i W_{ij}^{\text{RFP} \leftarrow \text{FFM}} s_{\text{FFM},i}(t). \quad (4.51)$$

During this phase, the active learning pathway extends from the input layer to the visual processing layer, through the FFM, to the output layer, where synaptic connections within the FFM repository are updated according to the R-STDP rule. Upon payload contact, the MPC neurons are activated, signaling the transition to the docking phase. The MPC

neurons inhibit both FFM and MPS neurons:

$$W_{ij}^{\text{FFM}\leftarrow\text{MPC}} = -\beta, \quad W_{ij}^{\text{RFP}\leftarrow\text{MPC}} = -\beta, \quad (4.52)$$

where  $\beta > 0$ , and the induced inhibition is

$$I_j^{\text{MPC-inh}}(t) = \sum_i W_{ij}^{\text{FFM/RFP}\leftarrow\text{MPC}} s_{\text{MPC},i}(t). \quad (4.53)$$

This inhibition silences FFM and MPS activity, releasing the CDM neurons from inhibition and allowing them to fire, thereby enabling the network to transition to the *docking phase*.

During docking, the active learning pathway becomes the input layer to the visual processing layer, to the CDM, and continues to the output layer, allowing the network to learn the docking policy explicitly through the active CDM connections.

This structure ensures that only the relevant neuron repository (FFM or CDM) is active during each mission phase, enforcing phase-specific learning while preventing interference between formation flying and docking. This phase-gated connectivity enables energy-efficient, event-driven learning aligned with the sparse nature of SNNs, providing reliable mission-phase transitions for autonomous docking missions.

## Output Layer and PWM Signal Generation

The output layer of the SNN consists of four neurons configured in the Class 1 excitability regime of the Izhikevich model. This configuration enables a smooth transition from resting to spiking states, allowing the neuron to adjust its firing frequency continuously in response to varying input currents. Such behavior is particularly suitable for motor control, where the precise modulation of thrust must reflect the instantaneous demands of the system.

The output layer neurons correspond to the four directional thrust commands: negative X-axis, positive X-axis, negative Y-axis, and positive Y-axis, indexed respectively as  $\{X^-, X^+, Y^-, Y^+\}$ . Each neuron integrates input currents received from the upstream hidden layer and produces spike trains at a temporal resolution determined by the simulation

step size.

The Class 1 neurons fire at low and adjustable frequencies, ensuring that the output spiking rate accurately reflects the magnitude of the input current. To compute the motor thrust commands, the spiking activity of each output neuron is sampled at the end of the control update window. Let  $\Delta t$  denote the simulation time step, and let  $T$  denote the control update interval. The binary spike state of each output neuron at the latest simulation time within the interval is denoted by:

$$D_i = f_i[T/\Delta t], \quad (4.54)$$

where  $f_i[k] \in \{0, 1\}$  indicates whether neuron  $i$  fired a spike at the  $k$ -th time step. This sampling approach effectively interprets the most recent spike activity of each neuron as a pulse-width modulation (PWM) control signal for the corresponding thrust direction.

The net thrust forces along the X and Y axes are computed as the difference between the PWM signals of opposing direction neurons, scaled by the maximum allowable thrust value  $F_{\max}$ :

$$F_X^{\text{thrust}} = F_{\max} \cdot (D_{X^+} - D_{X^-}), \quad (4.55)$$

$$F_Y^{\text{thrust}} = F_{\max} \cdot (D_{Y^+} - D_{Y^-}) \quad (4.56)$$

This formulation ensures that a recent spike in the  $X^+$  neuron, for example, results in a positive thrust command along the X-axis, while a spike in the  $X^-$  neuron results in a negative thrust command along the same axis. The same logic applies to the Y-axis neurons.

By using the most recent spike event rather than a time-averaged duty cycle, the system achieves high reactivity, allowing the motor commands to change instantly in response to environmental feedback or network decisions. This approach reduces computational overhead while preserving a direct and biologically inspired mapping between spiking activity and control output. Furthermore, the use of differential encoding across opposing neuron

pairs ensures that simultaneous contradictory commands cancel each other out, maintaining system stability and ensuring balanced control actions during fast maneuvers and precision positioning tasks.

### 4.2.2 Training Deep SNN with R-STDP

#### Reward Function for the First Hidden Layer

The reward function in the SNN provides structured, layered feedback to guide synaptic plasticity by aligning the network's fuzzy internal representations with the event-based sensory input, ensuring adaptive online learning. At each reward computation step, the fuzzy representation of the input layer activity is calculated as the time-averaged firing activity:

$$x_i^{\text{input}} = \frac{1}{T} \sum_{t-T}^t f_i, \quad (4.57)$$

where  $f_i(\tau) \in \{0, 1\}$  is the binary spiking activity of neuron  $i$  at time  $\tau$ , and  $T$  is the update window. Collecting these across all input neurons forms the input activity matrix:

$$\mathbf{X}_{\text{input}} = [x_1^{\text{input}}, x_2^{\text{input}}, \dots, x_N^{\text{input}}], \quad (4.58)$$

where each element indicates the fuzzy membership value of the corresponding feature in the input.

Similarly, for the first hidden layer, the payload detector and agent detector repositories compute:

$$x_i^{\text{hid}} = \frac{1}{T} \sum_{t-T}^t f_i^{\text{hid}}, \quad (4.59)$$

where  $f_i^{\text{hid}}(\tau)$  is the spiking activity of hidden layer neuron  $i$ . These form the matrices  $\mathbf{X}_{\text{payload}}$  and  $\mathbf{X}_{\text{agent}}$ , representing the fuzzy spatial activity maps for the payload and agents, respectively. To compute the attention map, the input activity matrix  $\mathbf{X}_{\text{input}}$  is compared with the positions represented in the hidden layer using a Gaussian-weighted mapping. For

a hidden layer grid cell at  $(x_q, y_q)$  and an input position  $(x_k, y_k)$ , the squared Euclidean distance is:

$$d_{kq}^2 = (x_k - x_q)^2 + (y_k - y_q)^2. \quad (4.60)$$

The attention weight assigned is:

$$w_{kq} = \exp\left(-\frac{d_{kq}^2}{2s^2}\right), \quad (4.61)$$

where  $s = [0.05, 0.1]$  is a scale parameter controlling the sensitivity. The *modulation signal*  $\alpha_q$ , which serves as the learning rate for each hidden layer neuron, is computed directly by comparing the attention-weighted input activity with the current hidden layer activity:

$$\alpha_q = \frac{\sum_k w_{kq} \cdot x_k^{\text{input}}}{\sum_k w_{kq} + \epsilon} - x_q^{\text{hid}}, \quad (4.62)$$

where  $\epsilon$  ensures numerical stability. This formulation reduces the dimensionality of the input while emphasizing high-activity regions aligned with the ALM areas and directly provides the per-neuron learning signal. When the hidden layer activity  $x_q^{\text{hid}}$  matches the attention-driven target, the learning rate satisfies  $\alpha_q \rightarrow 0$ , stopping the weight updates for that neuron during the iteration.

The reward for each hidden layer neuron is defined as:

$$r_q^{(1)} = x_q^{\text{hid}} + \epsilon, \quad (4.63)$$

ensuring the neuron remains responsive to reward-based modulation while avoiding numerical instability. Collecting the rewards across all neurons yields the reward vector:

$$\mathbf{r}^{(1)} = [r_1^{(1)}, r_2^{(1)}, \dots, r_N^{(1)}]^T. \quad (4.64)$$

Figure 4.5 shows how the reward function pipeline works for the agent detection process in the first hidden layer. The modulation signal provides the value of  $\alpha_q$  for each neuron,

corresponding to each pixel in the representation.

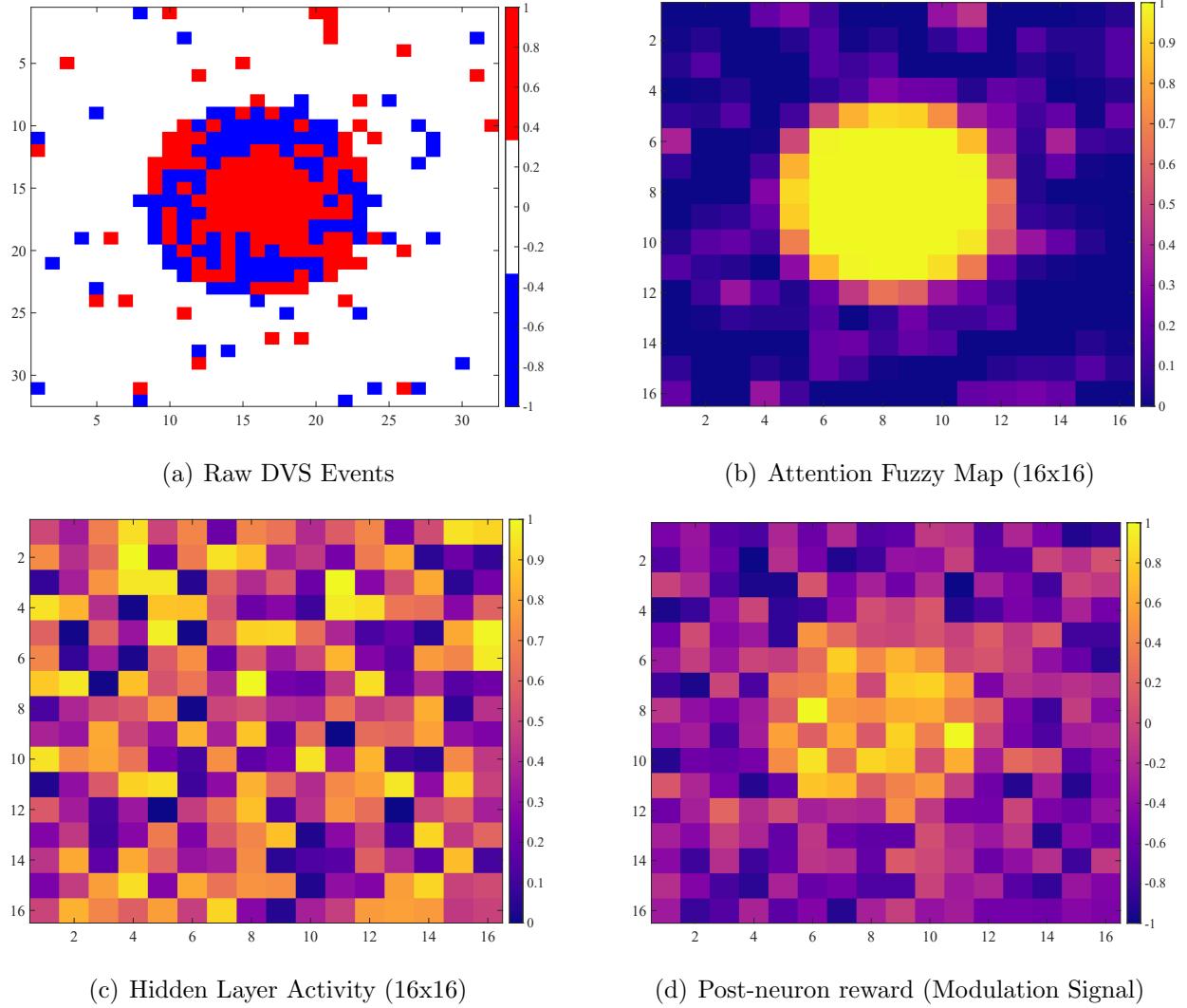


Figure 4.5: Reward function pipeline for the payload case. The pipeline illustrates the raw DVS events, the attention-based fuzzy map, the normalized hidden layer activity during the initial training phase ( $r_q^{(1)}$ ), and the modulation signal calculated as the difference between the attention map and the hidden layer activity ( $\alpha_q$ ).

Figure 4.6 shows an example of the reward function pipeline for the agent detector repository in the first hidden layer.

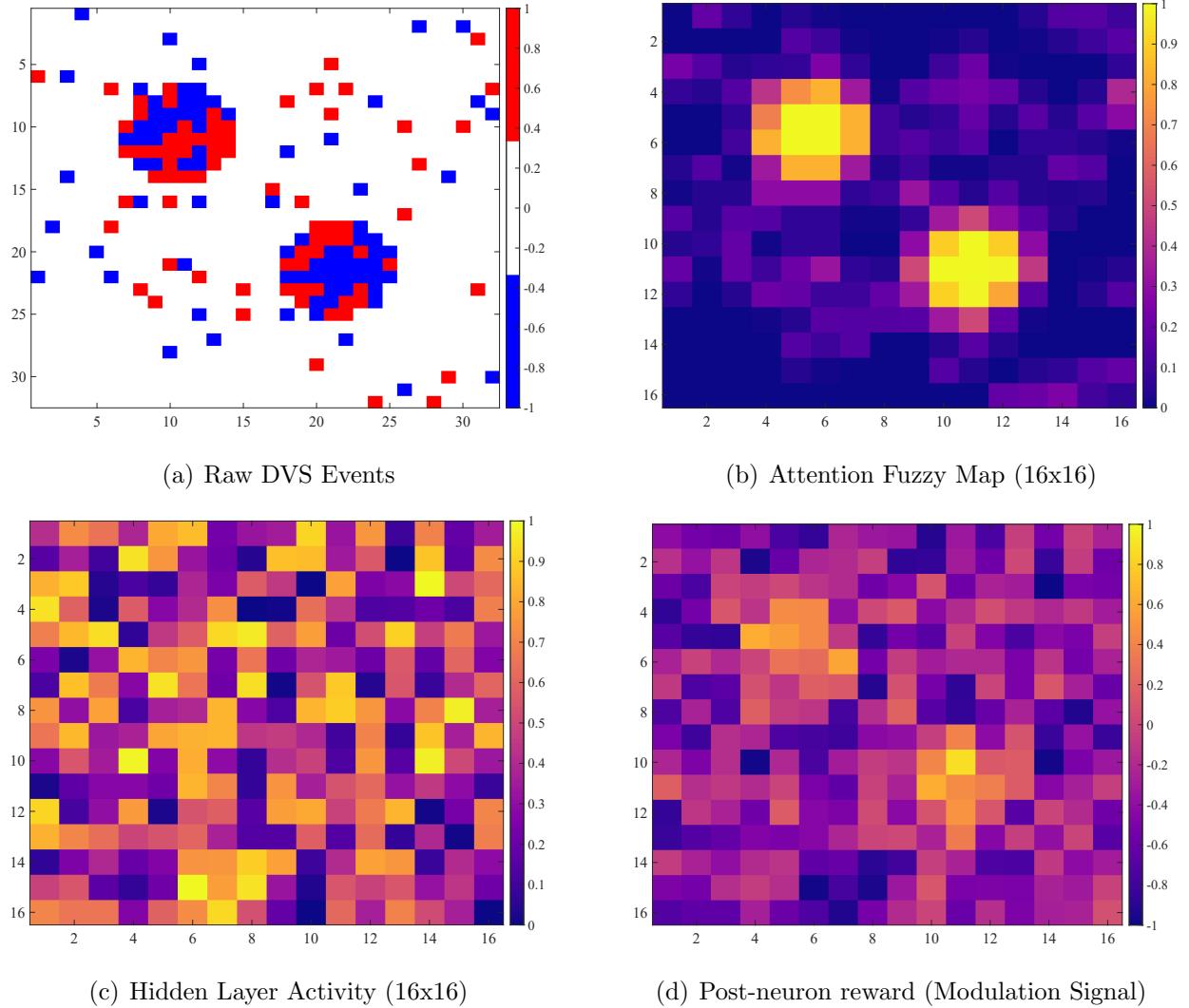


Figure 4.6: Reward function pipeline for the agents case. This pipeline demonstrates the raw DVS events for multiple agents, the reduced entropy-based fuzzy map used for attention, the hidden layer’s normalized activity, and the modulation signal computed for reward-based learning in the spiking neural network.

This design enables the first hidden layer to adaptively learn environmental configurations using event-based, fuzzy attention while regulating learning on a per-neuron basis according to the current match with the input-derived target activity, ensuring efficient, stable convergence in layered SNN learning.

### Reward Function for the Second and Third Hidden Layers

For the second hidden layer and the output layer, the reward is computed using a temporally oscillating Gaussian-centered modulation designed to align the network's learning with the spatial focus of the environment, typically the payload during docking and collaborative tasks.

The oscillation frequency is defined by:

$$f_{\text{osc}} = \frac{1}{T_{\text{ALM}}}, \quad (4.65)$$

where  $T_{\text{ALM}}$  is the ALM blinking period of the agent, and the corresponding angular frequency is  $\omega = 2\pi f_{\text{osc}}$  to synchronize the oscillation with the ALM cycle.

The baseline width of the Gaussian envelope is set to the payload's physical radius  $\sigma_0 = R_P$ , and the modulation depth is chosen as  $\alpha = 0.5 \cdot \sigma_0$ , corresponding to 50% of the baseline width.

Using these parameters, the time-varying width and amplitude of the Gaussian are defined as:

$$\sigma(t) = \sigma_0 + \alpha \cdot \sin(\omega t), \quad (4.66)$$

$$A(t) = \sin(\omega t). \quad (4.67)$$

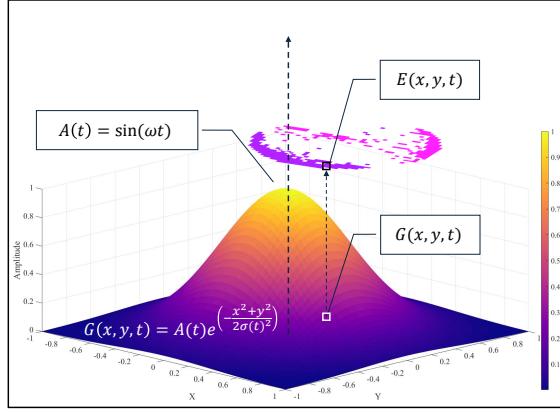
The oscillating Gaussian envelope over normalized spatial coordinates  $(x, y) \in [-1, 1]^2$  is then defined as:

$$G(x, y, t) = A(t) \cdot \exp\left(-\frac{x^2 + y^2}{2\sigma(t)^2}\right). \quad (4.68)$$

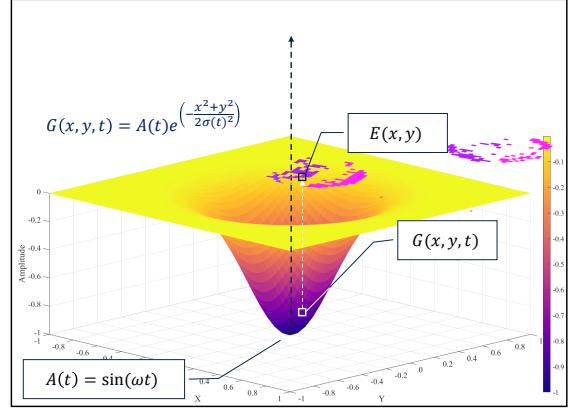
To compute the reward signal, the element-wise product between the normalized sensory event input  $E(x, y)$  and the Gaussian envelope is summed over the grid:

$$\mathcal{R} = \frac{\sum_{x,y} E(x, y) \cdot G(x, y, t)}{\sum_{x,y} |G(x, y, t)| + \epsilon}, \quad (4.69)$$

where  $\epsilon$  is a small constant to ensure numerical stability. This scalar reward  $\mathcal{R}$  quantifies how well the sensory input aligns with the oscillating Gaussian focus, dynamically guiding the network to attend to the center of the environment in synchrony with the ALM cycle.



(a) The oscillatory Gaussian envelope when the payload ALM is active.



(b) The oscillatory Gaussian envelope when the agent ALM is active.

The computed reward  $\mathcal{R}$  is scaled and uniformly assigned to each neuron in the second hidden layer and the output layer, enabling layered reward-modulated synaptic plasticity in which the second hidden layer and output layer utilize global, oscillation-driven rewards, complementing the fine-grained, per-neuron reward modulation of the first hidden layer. This mechanism ensures the network synchronizes learning with the environment's temporal dynamics while maintaining layered learning efficiency, enabling the SNN to focus on critical environmental features during operation.

### Weight Update using R-STDP

During the weight update phase, let  $\mathbf{C}$  denote the submatrix of eligibility traces of synaptic connections from pre-synaptic to post-synaptic neurons in the layer, which changes based on the STDP kernel and firing time of the pre- and post-synaptic neurons as follows:

$$\dot{\mathbf{C}}(t) = -\frac{1}{\tau_c} \mathbf{C}(t) + \mathbf{STDP}(\tau) \delta(t - \mathbf{T}) \quad (4.70)$$

where,  $\tau_c$  is the constant for the decay of  $\mathbf{C}_{\text{post, pre}}(t)$ , the  $\tau$  is the spike time difference between pre and post neurons, and  $\mathbf{T}$  is the matrix that shows the firing times of pre- and post-synaptic neurons. The per-neuron learning modulation using the computed reward and  $\alpha$  for the first hidden layer is:

$$\Delta \mathbf{W}_{\text{post, pre}} = \text{diag}(\boldsymbol{\alpha}_{\text{post}} \circ \mathbf{r}_{\text{post}}) \cdot \mathbf{C}_{\text{post, pre}}, \quad (4.71)$$

where  $\circ$  denotes element-wise multiplication and  $\text{diag}(\cdot)$  forms a diagonal matrix, ensuring neuron-specific, fuzzy, attention-guided plasticity.

The R-STDP algorithm for weight update for the second hidden layer and output layer is:

$$\Delta \mathbf{W}_{\text{post, pre}} = \mathbf{R} \cdot \mathbf{C}_{\text{post, pre}}, \quad (4.72)$$

where  $\mathbf{R}$  is broadcast across all post-synaptic neurons, enabling layer-wide, reward-driven synaptic updates aligned with the temporal structure of environmental feedback. This approach ensures that the first hidden layer learns using neuron-specific, attention-modulated updates, while the deeper layers benefit from global, phase-synchronized reward signals, leading to layered and stable online learning within the deep spiking neural network. This structure allows the network to adaptively refine synaptic weights in response to asynchronous event-driven sensory input and reward signals during autonomous multi-agent docking missions.

## 4.3 Results

In our high-fidelity simulation of the multi-agent docking task, the central payload is modeled as a rigid disc of radius  $R_p$  and mass  $m_p$ . A team of identical agents, each represented as a point mass  $m$  with its own collision radius  $R$ , cooperatively pushes the payload towards the docking center. Agent–payload and inter-agent collisions are handled via a linear spring–damper model with stiffness  $k$  and damping coefficient  $c$ . Each agent can exert a

thrust force of up to  $F_{\max}$  in the  $x$ - and  $y$ -directions. The specific parameter values used in all simulations are listed in Table 4.1.

Component	Parameter	Value	Units
Payload	Radius, $R_p$	1.5	m
	Mass, $m_p$	5.0	kg
Agent	Mass, $m$	1.0	kg
	Radius, $R$	0.5	m
	Stiffness, $K$	$1.0 \times 10^5$	kg/s <sup>2</sup>
	Damping, $c_{damp}$	$0.5 \cdot 2\sqrt{k}m \approx 316.2$	kg/s
Max thrust, $F_{\max}$		100	N

Table 4.1: Physical parameters of the payload and agents used in the docking simulations.

In our SNN implementation, we configure three principal layers of Izhikevich neurons, each simulated with a time step of  $\Delta t = 0.1$  ms. The input layer comprises 1024 bistable neurons that encode the preprocessed DVS currents, providing a short-term memory via persistent spiking in response to events. The first hidden layer contains 532 neurons—512 dedicated to visual feature processing and 20 tonic-bursting “ALM” attention units—operating in the Class 1 excitability regime to enable smooth frequency modulation. The second hidden layer comprises 104 Class 1 neurons, of which 64 form the “formation-flying” and “docking” repositories and 40 implement the regulatory MPC/RFP gating mechanisms. Finally, the output layer consists of 4 Class 1 neurons that generate directional PWM control signals for the X- and Y-axis thrust commands. Table 4.2 summarizes these layer configurations.

Layer	Model	Type	number of neurons	$\Delta t$ (ms)
Input	Izhikevich	Bistability	1024	0.1
Hidden 1 (Visual+ALM)	Izhikevich	Class 1	$512 + 20 = 532$	0.1
Hidden 2 (FFM/CDM+MPC/RFP)	Izhikevich	Class 1	$64 + 20 + 20 = 104$	0.1
Output	Izhikevich	Class 1	4	0.1

Table 4.2: Neuron layer configurations: model, excitability type, number of neurons  $N$ , and simulation time step  $\Delta t$ .

The simulation terminates successfully when the payload’s center enters within a 0.1 m radius of the specified docking point and remains there for at least 10 seconds. Weight updates are applied every 10 ms.

To ensure robust inhibitory control and prevent phase-mixing, each of the MPS, MPC, and ALM repositories is implemented with multiple neurons rather than a single unit. By deploying more than one neuron per repository, we guarantee that at least one neuron will fire at each sample time, maintaining a continuous inhibitory or gating signal even under variable spiking conditions. Furthermore, we introduce small random perturbations to the initial membrane potentials and Izhikevich parameters across neurons within each repository. This heterogeneity breaks up pathological synchrony—if all neurons in a repository were identical and initialized identically, they would tend to fire simultaneously, weakening phase separation. Instead, randomized initialization ensures staggered spiking, which improves the temporal coverage of inhibition and strengthens the decoupling of formation-flying and docking learning phases.

# Chapter 5

## Discussion and Conclusion

In this chapter, we summarize the key findings of our research, discuss their implications, and outline potential future work. Our study has demonstrated the effectiveness of Spiking Neural Networks (SNNs) in various applications, particularly in the context of autonomous systems and robotics. We have shown that SNNs can be effectively utilized for tasks such as Autonomous Target Detection (ATD) and consensus flying, leveraging their ability to process temporal information and adapt to dynamic environments. Furthermore, we have explored the integration of SNNs with Dynamic Vision Sensors (DVS) for docking applications, highlighting the advantages of using event-based sensors in robotic systems. The results of our experiments indicate that SNNs can achieve high accuracy and efficiency in these tasks.

### 5.1 Future Work

Looking ahead, there are several avenues for future research. One potential direction is to further enhance the learning algorithms used in SNNs, particularly in the context of online learning and adaptation to new environments.

# Bibliography

- [1] Y. Venkatesha, Y. Kim, L. Tassiulas, and P. Panda, “Federated learning with spiking neural networks,” *IEEE Transactions on Signal Processing*, vol. 69, pp. 6183–6194, 2021.
- [2] N. Skatchkovsky, H. Jang, and O. Simeone, “Federated neuromorphic learning of spiking neural networks for low-power edge intelligence,” in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8524–8528, IEEE, 2020.
- [3] S. A. Tumpa, S. Singh, M. F. F. Khan, M. T. Kandemir, V. Narayanan, and C. R. Das, “Federated learning with spiking neural networks in heterogeneous systems,” in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 1–6, IEEE, 2023.
- [4] N. Skatchkovsky, H. Jang, and O. Simeone, “Spiking neural networks—part iii: Neuromorphic communications,” *IEEE Communications Letters*, vol. 25, no. 6, pp. 1746–1750, 2021.
- [5] J. L. Lobo, J. Del Ser, A. Bifet, and N. Kasabov, “Spiking neural networks and online learning: An overview and perspectives,” *Neural Networks*, vol. 121, pp. 88–100, 2020.
- [6] S. Dora and N. Kasabov, “Spiking neural networks for computational intelligence: an overview,” *Big Data and Cognitive Computing*, vol. 5, no. 4, 2021.
- [7] K. Xie, Z. Zhang, B. Li, J. Kang, D. Niyato, S. Xie, and Y. Wu, “Efficient federated learning with spike neural networks for traffic sign recognition,” *IEEE Transactions on Vehicular Technology*, vol. 71, no. 9, pp. 9980–9992, 2022.

- [8] W. J. Yun, Y. Kwak, H. Baek, S. Jung, M. Ji, M. Bennis, J. Park, and J. Kim, “Slimfl: Federated learning with superposition coding over slimmable neural networks,” *IEEE/ACM Transactions on Networking*, 2023.
- [9] H. Yang, K.-Y. Lam, L. Xiao, Z. Xiong, H. Hu, D. Niyato, and H. Vincent Poor, “Lead federated neuromorphic learning for wireless edge artificial intelligence,” *Nature communications*, vol. 13, no. 1, pp. 42–69, 2022.
- [10] A. M. AbdelAty, M. E. Fouda, and A. M. Eltawil, “On numerical approximations of fractional-order spiking neuron models,” *Communications in Nonlinear Science and Numerical Simulation*, vol. 105, 2022.
- [11] Y.-H. Liu and X.-J. Wang, “Spike-frequency adaptation of a generalized leaky integrate-and-fire model neuron,” *Journal of computational neuroscience*, vol. 10, pp. 25–45, 2001.
- [12] L. Long and G. Fang, “A review of biologically plausible neuron models for spiking neural networks,” *AIAA Infotech@ Aerospace 2010*, 2010.
- [13] D. O. Hebb, *The organization of behavior: A neuropsychological theory*. Psychology press, 2005.
- [14] A. Triche, A. S. Maida, and A. Kumar, “Exploration in neo-hebbian reinforcement learning: Computational approaches to the exploration–exploitation balance with bio-inspired neural networks,” *Neural Networks*, vol. 151, pp. 16–33, 2022.
- [15] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.
- [16] D. L. Young and C.-S. Poon, “Hebbian covariance learning: a nexus for respiratory variability, memory, and optimization?,” *Advances in Modeling and Control of Ventilation*, pp. 73–83, 1998.
- [17] E. Oja, “Simplified neuron model as a principal component analyzer,” *Journal of mathematical biology*, vol. 15, pp. 267–273, 1982.

- [18] T. Toyoizumi, J.-P. Pfister, K. Aihara, and W. Gerstner, “Generalized bienenstock–cooper–munro rule for spiking neurons that maximizes information transmission,” *Proceedings of the National Academy of Sciences*, vol. 102, no. 14, pp. 5239–5244, 2005.
- [19] A. Soltoggio and J. J. Steil, “Solving the distal reward problem with rare correlations,” *Neural computation*, vol. 25, no. 4, pp. 940–978, 2013.
- [20] H. S. Seung, “Learning in spiking neural networks by reinforcement of stochastic synaptic transmission,” *Neuron*, vol. 40, no. 6, pp. 1063–1073, 2003.
- [21] E. M. Izhikevich, “Solving the distal reward problem through linkage of stdp and dopamine signaling,” *Cerebral cortex*, vol. 17, pp. 2443–2452, 2007.
- [22] A. L. Hodgkin and A. F. Huxley, “Currents carried by sodium and potassium ions through the membrane of the giant axon of loligo,” *The Journal of physiology*, vol. 116, no. 4, 1952.
- [23] E. M. Izhikevich, “Simple model of spiking neurons,” *IEEE Transactions on neural networks*, vol. 14, no. 6, pp. 1569–1572, 2003.
- [24] S. Song, K. D. Miller, and L. F. Abbott, “Competitive hebbian learning through spike-timing-dependent synaptic plasticity,” *Nature Neuroscience*, vol. 3, no. 9, pp. 919–926, 2000.
- [25] H. Z. Shouval, S. S.-H. Wang, and G. M. Wittenberg, “Spike timing dependent plasticity: a consequence of more fundamental learning rules,” *Frontiers in computational neuroscience*, vol. 4, 2010.
- [26] H. Markram, W. Gerstner, and P. J. Sjöström, “A history of spike-timing-dependent plasticity,” *Frontiers in synaptic neuroscience*, vol. 3, 2011.
- [27] V. Vassiliades, A. Cleanthous, and C. Christodoulou, “Multiagent reinforcement learning: spiking and nonspiking agents in the iterated prisoner’s dilemma,” *IEEE transactions on neural networks*, vol. 22, no. 4, pp. 639–653, 2011.

- [28] F. Ponulak and J. J. Hopfield, “Rapid, parallel path planning by propagating wavefronts of spiking neural activity,” *Frontiers in computational neuroscience*, vol. 7, 2013.
- [29] D. Probst, M. A. Petrovici, I. Bytschok, J. Bill, D. Pecevski, J. Schemmel, and K. Meier, “Probabilistic inference in discrete spaces can be implemented into networks of lif neurons,” *Frontiers in computational neuroscience*, vol. 9, 2015.
- [30] U. Markowska-Kaczmar and M. Koldowski, “Spiking neural network vs multilayer perceptron: who is the winner in the racing car computer game,” *Soft Computing*, vol. 19, pp. 3465–3478, 2015.
- [31] S. Guo, Z. Yu, F. Deng, X. Hu, and F. Chen, “Hierarchical bayesian inference and learning in spiking neural networks,” *IEEE transactions on cybernetics*, vol. 49, no. 1, pp. 133–145, 2017.
- [32] A. Tavanaei and A. Maida, “Bp-stdp: Approximating backpropagation using spike timing dependent plasticity,” *Neurocomputing*, vol. 330, pp. 39–47, 2019.
- [33] Z. Bing, C. Meschede, G. Chen, A. Knoll, and K. Huang, “Indirect and direct training of spiking neural networks for end-to-end control of a lane-keeping vehicle,” *Neural Networks*, vol. 121, pp. 21–36, 2020.
- [34] C. Hur, B. Ibrokhimov, and S. Kang, “N3-cpl: Neuroplasticity-based neuromorphic network cell proliferation learning,” *Neurocomputing*, vol. 411, pp. 193–205, 2020.
- [35] L. Salt, D. Howard, G. Indiveri, and Y. Sandamirskaya, “Parameter optimization and learning in a spiking neural network for uav obstacle avoidance targeting neuromorphic processors,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 9, pp. 3305–3318, 2019.
- [36] J. Liu, H. Lu, Y. Luo, and S. Yang, “Spiking neural network-based multi-task autonomous learning for mobile robots,” *Engineering Applications of Artificial Intelligence*, vol. 104, 2021.

- [37] H. Lu, J. Liu, Y. Luo, Y. Hua, S. Qiu, and Y. Huang, “An autonomous learning mobile robot using biological reward modulate stdp,” *Neurocomputing*, vol. 458, pp. 308–318, 2021.
- [38] Q. Zhan, G. Liu, X. Xie, G. Sun, and H. Tang, “Effective transfer learning algorithm in spiking neural networks,” *IEEE Transactions on Cybernetics*, vol. 52, no. 12, pp. 13323–13335, 2021.
- [39] D. Chu and H. Le Nguyen, “Constraints on hebbian and stdp learned weights of a spiking neuron,” *Neural Networks*, vol. 135, pp. 192–200, 2021.
- [40] S. Schmidgall, J. Ashkanazy, W. Lawson, and J. Hays, “Spikepropamine: Differentiable plasticity in spiking neural networks,” *Frontiers in neurorobotics*, vol. 15, 2021.
- [41] B. Golosio, G. Tiddia, C. De Luca, E. Pastorelli, F. Simula, and P. S. Paolucci, “Fast simulations of highly-connected spiking cortical models using gpus,” *Frontiers in Computational Neuroscience*, vol. 15, 2021.
- [42] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, “On the convergence of fedavg on non-iid data,” *arXiv preprint arXiv:1907.02189*, 2019.
- [43] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, “Federated optimization in heterogeneous networks,” *Proceedings of Machine learning and systems*, vol. 2, pp. 429–450, 2020.
- [44] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, “Scaffold: Stochastic controlled averaging for federated learning,” in *International conference on machine learning*, pp. 5132–5143, PMLR, 2020.
- [45] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, “A novel framework for the analysis and design of heterogeneous federated learning,” *IEEE Transactions on Signal Processing*, vol. 69, pp. 5234–5249, 2021.

- [46] Q. Li, B. He, and D. Song, “Model-contrastive federated learning,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10713–10722, 2021.
- [47] D. Ye, R. Yu, M. Pan, and Z. Han, “Federated learning in vehicular edge computing: A selective model aggregation approach,” *IEEE Access*, vol. 8, pp. 23920–23935, 2020.
- [48] K. Yang, T. Jiang, Y. Shi, and Z. Ding, “Federated learning via over-the-air computation,” *IEEE transactions on wireless communications*, vol. 19, no. 3, pp. 2022–2035, 2020.
- [49] Y. Zhan, P. Li, Z. Qu, D. Zeng, and S. Guo, “A learning-based incentive mechanism for federated learning,” *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6360–6368, 2020.
- [50] M. Chen, D. Gündüz, K. Huang, W. Saad, M. Bennis, A. V. Feljan, and H. V. Poor, “Distributed learning in wireless networks: Recent progress and future challenges,” *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 12, pp. 3579–3605, 2021.
- [51] M. Chen, H. V. Poor, W. Saad, and S. Cui, “Convergence time optimization for federated learning over wireless networks,” *IEEE Transactions on Wireless Communications*, vol. 20, no. 4, pp. 2457–2471, 2020.
- [52] M. A. Ferrag, O. Friha, L. Maglaras, H. Janicke, and L. Shu, “Federated deep learning for cyber security in the internet of things: Concepts, applications, and experimental analysis,” *IEEE Access*, vol. 9, pp. 138509–138542, 2021.
- [53] M. Chen, Z. Yang, W. Saad, C. Yin, H. V. Poor, and S. Cui, “A joint learning and communications framework for federated learning over wireless networks,” *IEEE Transactions on Wireless Communications*, vol. 20, no. 1, pp. 269–283, 2020.

- [54] Z. Yang, M. Chen, W. Saad, C. S. Hong, and M. Shikh-Bahaei, “Energy efficient federated learning over wireless communication networks,” *IEEE Transactions on Wireless Communications*, vol. 20, no. 3, pp. 1935–1949, 2020.
- [55] A. B. Mansour, G. Carenini, A. Duplessis, and D. Naccache, “Federated learning aggregation: New robust algorithms with guarantees,” in *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 721–726, IEEE, 2022.
- [56] A. Jarwan and M. Ibnkahla, “Edge-based federated deep reinforcement learning for iot traffic management,” *IEEE Internet of Things Journal*, vol. 10, no. 5, pp. 3799–3813, 2022.
- [57] Z. Yuan, Z. Wang, X. Li, L. Li, and L. Zhang, “Hierarchical trajectory planning for narrow-space automated parking with deep reinforcement learning: A federated learning scheme,” *Sensors*, vol. 23, no. 8, 2023.
- [58] Q. Wu, X. Chen, T. Ouyang, Z. Zhou, X. Zhang, S. Yang, and J. Zhang, “Hiflash: Communication-efficient hierarchical federated learning with adaptive staleness control and heterogeneity-aware client-edge association,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 5, pp. 1560–1579, 2023.
- [59] G. Xia, J. Chen, C. Yu, and J. Ma, “Poisoning attacks in federated learning: A survey,” *IEEE Access*, vol. 11, pp. 10708–10722, 2023.
- [60] M. Pachter, E. Garcia, and D. W. Casbeer, “Differential game of guarding a target,” *Journal of Guidance, Control, and Dynamics*, vol. 40, pp. 2991–2998, 2017.
- [61] I. E. Weintraub, M. Pachter, and E. Garcia, “An introduction to pursuit-evasion differential games,” in *2020 American Control Conference (ACC)*, pp. 1049–1066, 2020.
- [62] E. Garcia, “Cooperative target protection from a superior attacker,” *Automatica*, vol. 131, 2021.

- [63] M. D. Awheda and H. M. Schwartz, “A decentralized fuzzy learning algorithm for pursuit-evasion differential games with superior evaders,” *Journal of Intelligent & Robotic Systems*, vol. 83, pp. 35–53, 2016.
- [64] A. D. Bird, P. Jedlicka, and H. Cuntz, “Dendritic normalisation improves learning in sparsely connected artificial neural networks,” *PLOS Computational Biology*, vol. 17, 2021.
- [65] J. L. Lobo, I. Laña, J. Del Ser, M. N. Bilbao, and N. Kasabov, “Evolving spiking neural networks for online learning over drifting data streams,” *Neural Networks*, vol. 108, pp. 1–19, 2018.
- [66] M. Tayefe Ramezanlou, H. Schwartz, I. Lambadaris, M. Barbeau, and S. H. R. Naqvi, “Learning a policy for pursuit-evasion games using spiking neural networks and the stdp algorithm,” in *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 1918–1925, 2023.
- [67] P. Qi, D. Chiaro, A. Guzzo, M. Ianni, G. Fortino, and F. Piccialli, “Model aggregation techniques in federated learning: A comprehensive survey,” *Future Generation Computer Systems*, vol. 150, pp. 272–293, 2024.
- [68] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data,” in *Artificial intelligence and statistics*, pp. 1273–1282, PMLR, 2017.
- [69] Z. Wang, Z. Peng, X. Fan, Z. Wang, S. Wu, R. Yu, P. Yang, C. Zheng, and C. Wang, “Fedave: Adaptive data value evaluation framework for collaborative fairness in federated learning,” *Neurocomputing*, vol. 574, 2024.
- [70] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, “Tackling the objective inconsistency problem in heterogeneous federated optimization,” *Advances in neural information processing systems*, vol. 33, pp. 7611–7623, 2020.

- [71] P. G. F. Dias, M. C. Silva, G. P. Rocha Filho, P. A. Vargas, L. P. Cota, and G. Pessin, “Swarm robotics: A perspective on the latest reviewed concepts and applications,” *Sensors*, vol. 21, no. 6, 2021.