Multi-agent Reinforcement Learning on 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
Spring, 2024

Abstract

Contents

1 Introduction **1**

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 Federated Learning 14

2 Literature review **18**

3 Modular Learning in SNNs for Optimal Multi-Agent Decision-Making **25**

3.1 Introduction 25

3.2 The ATD problem and SNN-based solution 26

3.3 Learning using R-STDP 27

3.4 Network structure and encoding method 31

3.5 Results 35

3.5.1 Simulation without noise 38

3.5.2 Simulation with noise 41

3.6 Conclusion 43

4 Integration of R-STDP and Federated Learning **44**

4.1 Consensus Flying Problem 44

4.2 Proposed Method 46

4.2.1 Network Structure 46

4.2.2 Training algorithm 51
4.2.3 Weight Stabilization using Reward-Modulated Competitive Synaptic Equilibrium (R-CSE) 52
4.2.4 Federated Learning for Consensus Flying 57
4.3 Results and Discussion 62
4.3.1 Simulation without Federated Learning (FL) 62
4.3.2 Simulation with Federated Learning (FL) and Reward-Modulated Competitive Synaptic Equilibrium (RCSE) 72
4.4 Conclusion 75

5 Proposed Method **77**

List of Figures

1.1 Schematic diagram to illustrate distributed learning and Federated Learning (FL) [1]. 1

3.1 Active target defense game with three agents (LOS angles are shown for both agents). 27

3.2 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. 28

3.3 Relative velocities and LOS angles used in the reward function 30

3.4 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 ($\alpha = 0$) and an operating phase when ($\alpha = 1$). 32

3.5 Spiking Neural Network (SNN)'s performance during training. Hollow circles show the initial positions. 36

3.6 Changes in synaptic weights during training. Only Maximum synaptic weights for both agents are shown. 37

3.7 An illustration of input current to Spiking Neural Network (SNN) and synaptic current to the output neurons after training for a single connection. 39

3.8 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. 39
3.9 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. 40
3.10 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. . . . 42

4.1 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. 45
4.2 Spiking Neural Network (SNN) structure with encoding and decoding layers. Each sub-layer consists of a fuzzy encoder and a Fuzzy-to-Spiking 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. 47
4.3 The fuzzy encoding principle for the input sub-layer. 49
4.4 Synaptic weight change for $\lambda = 5$, $\Psi^{\mathcal{S}} = 15.5$, and $\mathcal{AR}^{G}_{max} = 0.5$ 57
4.5 Derivation of Synaptic weight for $\lambda = 5$, $\Psi^{\mathcal{S}} = 15.5$, and $\mathcal{AR}^{G}_{max} = 0.5$ 57
4.6 The measured distances during the test for evaluating the performance and detecting the collisions. 65
4.7 Distances during the test phase (Reward-Modulated Competitive Synaptic Equilibrium (RCSE) method) 68
4.8 Agents' trajectory during the test phase (Reward-Modulated Competitive Synaptic Equilibrium (RCSE) method) 68

4.9 Change in commanded distance during the test phase (Reward-Modulated Competitive Synaptic Equilibrium (RCSE) method) 68
4.10 Distances during the test phase (FACL method) 68
4.11 Agents' trajectory during the test phase (FACL method) 68
4.12 Change in commanded distance during the test phase (FACL method) 68
4.13 Agents' path after reward change (test phase) 70
4.14 Distances after reward change (test phase) 70
4.15 Synaptic Weights before Reward change 71
4.16 Synaptic Weights after Reward change 71
4.17 Synaptic weights increase after reward change 72
4.18 Synaptic weights decrease after reward change 72
4.19 Distances during the test phase before reward change in the proposed aggregation method. 73
4.20 Distances during the test phase after reward change in the proposed aggregation method. 73
4.21 Frobenius norm of the agents during the learning phase. The reward changes for the Leader after 600 s. 74
4.22 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. 74

List of Tables

3.1 Parameter values for LIF neuron model [10] 35

3.2 Parameter values for Reward-modulated Spike-Timing-Dependent Plasticity (R-STDP) 36

4.1 Parameter values for Leaky Integrate-and-Fire (LIF) neuron model [68] . . . 63

4.2 Simulation Parameters 64

4.3 Simulation Parameters for the FACL Algorithm 67

4.4 Comparative Performance Analysis of the Proposed Aggregation Algorithm+Reward-Modulated Competitive Synaptic Equilibrium (RCSE), Reward-Modulated Competitive Synaptic Equilibrium (RCSE), and FACL 75

Acronyms

FACL Fuzzy Actor-Critic Learning

FL Federated Learning

GRF Gaussian Receptive Fields

LIF Leaky Integrate-and-Fire

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, such as smartphones and IoT devices, without the need to centralize 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 in our increasingly connected digital age, making it a critical area of research for advancing machine learning technologies that are both accessible and secure.

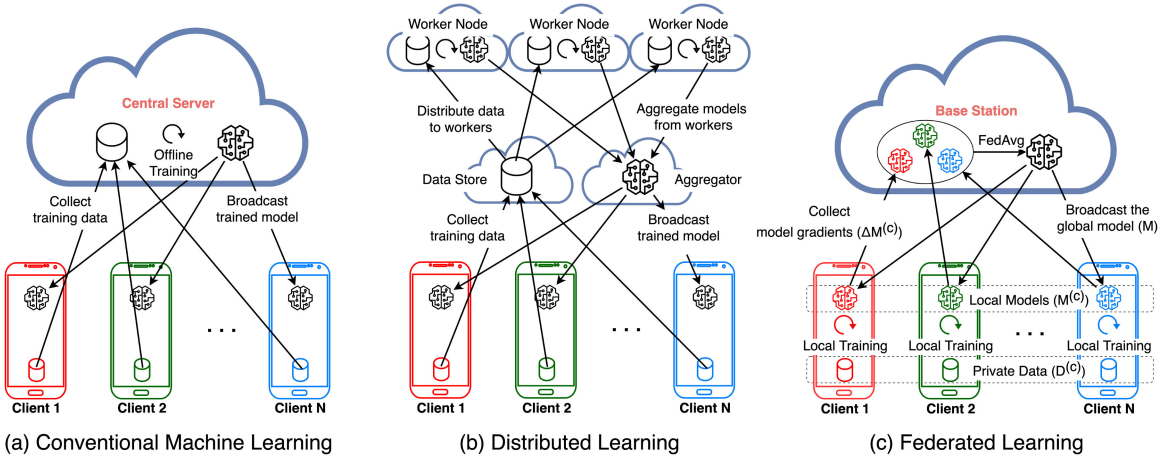

Figure 1.1: Schematic diagram to illustrate distributed learning and FL [1].

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 in an offline manner. 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 base station which aggregates the gradients and updates the global model, and broadcasts it back to all the clients. Each of these processes is iterative and models at 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 devices participating in FL. This characteristic of SNNs supports the development of low-power, distributed learning solutions, enabling more devices 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 model update quantization 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. This section provides an overview of several key neuron models, including their equations and characteristics.

Leaky-Integrate and Fire (LIF) model

The 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],

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

where $V_m(t)$ is the neuron's membrane potential, 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].

$$r_{[Hz]} = \frac{1}{t_{isi}\ [s]} \tag{1.2}$$

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,

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

where τ_m is the membrane time constant.

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

$$E_l + R_m I - V_{th} > 0 \tag{1.4}$$

or

$$I > \frac{V_{th} - E_l}{R_m} \tag{1.5}$$

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,

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

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

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

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

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

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

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

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.

Hodgkin-Huxley Model

The Hodgkin-Huxley model offers a detailed description of the ionic mechanisms underlying the initiation and propagation of action potentials in neurons [13]:

$$C_m \frac{dV}{dt} = I_{ext} - \bar{g}_K n^4 (V - V_K) - \bar{g}_{Na} m^3 h (V - V_{Na}) - \bar{g}_L (V - V_L) \tag{1.10}$$

$$\frac{dn}{dt} = \alpha_n(V)(1 - n) - \beta_n(V) n \tag{1.11}$$

$$\frac{dm}{dt} = \alpha_m(V)(1 - m) - \beta_m(V) m \tag{1.12}$$

$$\frac{dh}{dt} = \alpha_h(V)(1 - h) - \beta_h(V) h \tag{1.13}$$

where C_m is the membrane capacitance, V is the membrane potential, I_{ext} is the external current, $\bar{g}_i$ and V_i are the maximum conductances and reversal potentials for the K^+, Na^+, and leak (L) currents, and n, m, and h are gating variables.

The variables n, m, and h are dimensionless and vary between 0 and 1, representing the proportion of ion channels in a particular state. The dynamics of these gating variables are critical for the model's ability to replicate the complex temporal patterns of neuronal action potentials. The rates $\alpha_{n,m,h}$ and $\beta_{n,m,h}$ represent the voltage-dependent transition rates for the gating variables, where α corresponds to the rate of transition from a closed to an open state, and β represents the rate of transition from an open to a closed state. These rates are essential for modeling how the gating variables respond to changes in membrane potential, thereby influencing the ion flow across the membrane and the generation of action potentials.

FitzHugh-Nagumo Model

The FitzHugh-Nagumo model simplifies the Hodgkin-Huxley model into a two-variable system to capture the essential features of excitability [14]:

$$\frac{dV}{dt} = V - \frac{V^3}{3} - U + I_{ext} \tag{1.14}$$

$$\frac{dU}{dt} = \frac{1}{\tau}(V + a - bU) \tag{1.15}$$

where V represents the membrane potential, U is a recovery variable, and I_{ext} is the external current. The parameter a represents a threshold value influencing the activation of V, b controls the recovery variable's influence on the system, and τ is the time constant for the recovery variable U, determining its rate of adjustment in response to changes in V.

Izhikevich Model

The Izhikevich model combines the biological plausibility of the Hodgkin-Huxley model with the computational efficiency of Leaky Integrate-and-Fire (LIF) models [15]:

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I \tag{1.16}$$

$$\frac{du}{dt} = a(bv - u) \tag{1.17}$$

when $v \geq 30$ mV, then $v \leftarrow c$ and $u \leftarrow u + d$. Here, v and u represent the membrane potential and membrane recovery variable, respectively, and a, b, c, and d are parameters of the model, with I representing the current.

1.1.2 Learning Approaches in SNNs

Hebbian Learning

Before we turn to spike-based learning rules, we first review the basic concepts of rate-based learning. Firing rate models have been used extensively in the field of SNNs.

Consider a single synapse with efficacy w_{ij} that transmits signals from a presynaptic neuron j to a postsynaptic neuron i, where the activity of the presynaptic neuron is denoted

by ν_j and that of the postsynaptic neuron by ν_i. The change in synaptic efficacy can be described by the differential equation [16]:

$$\frac{d}{dt}w_{ij} = F(w_{ij}; \nu_i\nu_j), \tag{1.18}$$

where F is a function representing the change rate of synaptic coupling strength, reflecting the principle of locality in Hebbian plasticity. This function is dependent on the pre- and postsynaptic firing rates and the current synaptic efficacy.

There are two aspects of Hebb's postulate that are particularly important: locality and joint activity. Hebb's postulate emphasizes the necessity of simultaneous pre- and postsynaptic activity for synaptic modification. This can be mathematically expressed in a Taylor series expansion around $\nu_i = \nu_j = 0$ [16]:

$$\frac{d}{dt}w_{ij} = c_0(w_{ij}) + c^1_{\text{pre}}(w_{ij})\nu_j + c^1_{\text{post}}(w_{ij})\nu_i + c^2_{\text{pre}}(w_{ij})\nu_j^2 + c^2_{\text{post}}(w_{ij})\nu_i^2 + c^{11}_{\text{corr}}(w_{ij})\nu_i\nu_j + O(\nu^3). \tag{1.19}$$

where $c_0(w_{ij})$ accounts for the baseline rate of change of the synaptic weight, independent of neuronal activity, and $c^1_{\text{pre}}(w_{ij})$ and $c^1_{\text{post}}(w_{ij})$ modulate the synaptic weight change linearly with the activity of the presynaptic and postsynaptic neurons, respectively. The $c^2_{\text{pre}}(w_{ij})$ and $c^2_{\text{post}}(w_{ij})$ reflect the quadratic terms, indicating the rate of weight change with the square of the activity levels of the presynaptic and postsynaptic neurons, respectively, suggesting nonlinear effects. The $c^{11}_{\text{corr}}(w_{ij})$ represents the interaction term between the presynaptic and postsynaptic activities, emphasizing the Hebbian principle that the synaptic modification is strongest when both neurons are simultaneously active. The $O(\nu^3)$ indicates that this expansion includes up to the second order of neuronal activities and that higher-order terms are omitted for simplicity.

The synaptic change is governed by local variables, such as the synaptic efficacy and the pre- and postsynaptic firing rates, without dependency on the activity of other neurons. Simplifying the Taylor expansion to include only the bilinear term for joint activity leads to:

$$\frac{d}{dt}w_{ij} = c_{\text{corr}}^{11}\nu_i\nu_j, \tag{1.20}$$

indicating that synaptic strength increases with simultaneous firing of pre- and postsynaptic neurons.

To prevent unlimited growth of synaptic weights, models introduce a dependency of the coefficient c_{corr}^{11} on the current weight value, enabling a saturation effect. Two common approaches are [17]:

- **Hard Bound**: c_{corr}^{11} remains constant until a maximum weight $w_{\max}$ is reached, after which it becomes zero.
- **Soft Bound**: $c_{\text{corr}}^{11}(w_{ij}) = \gamma(w_{\max} - w_{ij})^{\beta}$, allowing for gradual slowing of weight growth as it approaches $w_{\max}$.

Real systems require mechanisms for decreasing synaptic weights as well. This can be represented by [18]:

$$\frac{d}{dt}w_{ij} = \gamma(1 - w_{ij})\nu_i\nu_j - \gamma_0 w_{ij}, \tag{1.21}$$

where γ_0 represents a decay term, leading to spontaneous weakening of synapses in the absence of stimulation.

Building upon the foundational principles introduced in the initial formulation of Hebb's rule, this section delves into famous models and refinements that offer a richer understanding of synaptic plasticity. These elaborations include the Covariance Rule, Oja's Rule, and the Bienenstock-Cooper-Munro (BCM) Rule, each providing unique insights into the dynamics of learning and memory in SNNs.

Covariance Rule

The Covariance Rule introduces a nuanced perspective on synaptic modification, emphasizing the importance of the correlation between fluctuations in pre- and postsynaptic activity rates from their mean values. Mathematically, it can be expressed as:

$$\frac{d}{dt}w_{ij} = \gamma(\nu_i - \overline{\nu_i})(\nu_j - \overline{\nu_j}), \tag{1.22}$$

where $\overline{\nu_i}$ and $\overline{\nu_j}$ represent the running averages of the postsynaptic and presynaptic neuron firing rates, respectively. This rule suggests that synaptic strength is adjusted based on the covariance of firing rates, promoting a more dynamic and responsive mechanism for synaptic plasticity that accounts for temporal variations in neural activity.

Oja's Rule

This approach offers a self-organizing and normalization mechanism for synaptic weights, ensuring stability in the learning process. The rule is given by [19]:

$$\frac{d}{dt}w_{ij} = \gamma(\nu_i\nu_j - w_{ij}\nu_i^2), \tag{1.23}$$

This formulation inherently limits the growth of synaptic weights, ensuring that the sum of the squares of the weights converges to unity. This normalization aspect introduces competition among synapses converging on the same postsynaptic neuron, balancing potentiation with necessary synaptic weakening for others, thereby maintaining overall network stability.

Bienenstock-Cooper-Munro (BCM) Rule

The BCM rule introduces a mechanism where synaptic modification thresholds evolve based on the neuron's activity history. The rule can be described as:

$$\frac{d}{dt}w_{ij} = \eta\nu_i(\nu_i - \nu_\theta)\nu_j, \tag{1.24}$$

where ν_θ is a dynamic threshold that adjusts based on the average postsynaptic activity. This adaptive threshold introduces a form of synaptic plasticity that is both potentiation and depression capable, contingent upon the postsynaptic activity level relative to ν_θ.

Rarely Correlating Hebbian Plasticity (RCHP)

RCHP proposes a model where synaptic changes are driven not merely by the simultaneous activation of connected neurons but by the infrequent yet pivotal coincidences in their activity. This model addresses the potential issue of over-strengthening synaptic con-

nections due to common but unimportant activity patterns, focusing instead on enhancing those connections that contribute to significant neural events.

The RCHP rule is defined by an update mechanism that adjusts synaptic weights based on a thresholded measure of pre- and post-synaptic activity correlations. The rule can be formally expressed as follows [20]:

$$\Delta w_{ji} = \begin{cases} +0.5 & \text{if } y_j(t-\Delta t)y_i(t) > \theta^+, \\ -1 & \text{if } y_j(t-\Delta t)y_i(t) < \theta^-, \\ +0 & \text{otherwise,} \end{cases} \tag{1.25}$$

where Δw_{ji} denotes the change in synaptic weight between the pre-synaptic neuron j and the post-synaptic neuron i, $y_j(t-\Delta t)$ represents the activity of the pre-synaptic neuron at a time Δt before the current time t, and $y_i(t)$ is the current activity of the post-synaptic neuron. The parameters θ^+ and θ^- set the thresholds for potentiation and depression, respectively, aiming to identify rare yet meaningful correlations that surpass these thresholds.

The RCHP rule introduces a dynamic approach to synaptic plasticity, where the focus shifts from frequent activity correlations to those rare instances of significant synchrony between neurons. By setting specific thresholds for activity correlations, RCHP ensures that only those synaptic connections that facilitate important neural events are strengthened, while others are either weakened or left unchanged. This selective reinforcement mechanism aids in the fine-tuning of neural circuits, optimizing them for critical functions and responses.

At the heart of modern RL formulations within SNNs is the adaptation of neo-Hebbian learning rules, which introduce the concept of a reward or value as a third critical factor modulating the synaptic strength between pre- and post-synaptic neurons. This modulation is achieved through various scaling or gating mechanisms in response to global reward signals, predominantly mediated by the neuromodulator dopamine.

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 [21]. 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. Conversely, unsuccessful firings before positive rewards lead to LTD, but before negative rewards, they trigger LTP.

Distal rewards and credit assignment

This model introduces a nuanced 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 [22]:

$$\frac{dC_{ji}}{dt} = -\frac{C_{ji}}{\tau_C} + \mathrm{STDP}(t_{\mathrm{post}} - t_{\mathrm{pre}})\delta(t - t^{(f)}) \tag{1.26}$$

Here, $C_{ji}(t)$ denotes the eligibility trace for the synapse between pre-synaptic neuron j and post-synaptic neuron i, evolving over time with a decay governed by τ_C, and modulated by the STDP rule applied to the timing difference between pre- and post-synaptic spikes $(t_{\mathrm{post}} - t_{\mathrm{pre}})$. 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.

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

$$\frac{dw_{ji}}{dt} = C_{ji}(t) \cdot R(t) \tag{1.27}$$

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, thereby embodying a direct manifestation of reward-based learning within the synaptic framework.

Hypothesis Testing Plasticity (HTP)

The HTP model focuses on the nuanced evolution of synaptic strengths through the interaction with both short-term and long-term components of synaptic weights [23]. The HTP model delineates the dynamics of synaptic strength through a dual-component approach. This approach segregates synaptic weight into short-term and long-term components, facilitating the exploration of synaptic plasticity under varying reward conditions. In the following, the mathematical representation of this model, including the dynamics of short-term synaptic changes and the conditions for long-term consolidation is discussed.

Short-term synaptic weight changes are governed by,

$$\Delta w_{ji}^{st}(t) = -\frac{w_{ji}^{st}(t)}{\tau_{st}} + M(t) \cdot \mathrm{RCHP}_{ji}(t) \tag{1.28}$$

where $M(t)$, representing the dynamic concentration of reward, evolves as:

$$\Delta M(t) = -\frac{M(t)}{\tau_M} + \alpha R(t) - b \tag{1.29}$$

Long-term synaptic changes are then formulated as:

$$\Delta w_{ji}^{lt}(t) = \beta \mathcal{H}(w_{ji}^{st}(t) - \Phi) \tag{1.30}$$

where $\mathcal{H}$ denotes the Heaviside step function, introducing a binary condition for the consolidation of synaptic changes based on the magnitude of short-term synaptic efficacy relative to the threshold Φ, and β represents the consolidation rate.

This framework integrates the dynamics of synaptic plasticity with mechanisms of reward-based learning, offering a comprehensive model for understanding the biological and computational foundations of learning and memory.

1.2 Federated Learning

A cornerstone of FL is the aggregation method, which synthesizes model updates from clients to improve a global model. The efficiency, robustness, and overall performance of FL significantly hinge on the choice of aggregation technique. This section delves into the main aggregation algorithms in FL, highlighting their operational principles and underlying equations.

FedAvg

FedAvg, or Federated Averaging, is the seminal aggregation method in FL. It operates by averaging the weights of models updated locally by clients. The global model update in FedAvg is mathematically represented as [24]:

$$w_{\text{glob}}^{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_k^{t+1} \tag{1.31}$$

where w_{glob}^{t+1} denotes the global model parameters after aggregation, n_k is the number of data points at client k, n is the total number of data points across all clients, and w_k^{t+1} represents the parameters of the model updated locally by client k.

FedProx

FedProx addresses heterogeneity issues in client data distributions by introducing a proximal term to the local optimization problem. This approach moderates the deviation of local models from the global model. The local update rule in FedProx is formulated as [25]:

$$w_{\text{glob}}^{t+1} = \min_{w} \mathcal{L}_k(w) + \frac{\mu}{2}\|w - w_{\text{glob}}^{t}\|^2 \tag{1.32}$$

where $\mathcal{L}_k(w)$ is the loss on client k, w_{glob}^{t} is the global model parameters at round t, and μ is a regularization parameter controlling the proximity between local and global models.

Scaffold

Scaffold addresses the challenges of statistical heterogeneity and client drift in federated learning environments. It combats client drift and enhances convergence by employing control variates to correct the local updates towards the global objective, effectively steering them closer to the global optimum and reducing the variance in updates caused by data non-IIDness across clients. The control variate updates are defined as [26]:

$$c_k^{+} = c_k - c + \frac{1}{\eta L}(w_{\text{glob}}^{t} - w_k^{t}) \tag{1.33}$$

where c_k and c represent the control variates for client k and the global model, respectively. Here, L denotes the number of local updates, and η is the learning rate. This formula reflects how the local and global control variates are adjusted to mitigate the divergence of client updates from the global update direction.

By introducing control variates on both the server and client sides, Scaffold corrects the direction of local updates before they are aggregated. This not only ensures consistency among local updates but also addresses the slow convergence rates often experienced in traditional federated learning due to client drift. Specifically, the mechanism leverages the discrepancy between local and global control variates to adjust the updates, making Scaffold particularly effective in environments with heterogeneous data distributions.

FedNova

FedNova introduces a normalization mechanism to adjust the aggregation process according to the number of local updates performed by each client, accommodating non-IID data

scenarios more effectively. Its aggregation rule is expressed as [27]:

$$w_{glob}^{t+1} = w_{glob}^{t} - \tau_{\text{eff}} \sum_{k=1}^{K} \frac{n_k}{n} \cdot \eta d_k^t \tag{1.34}$$

$$d_k^t = G_k^t a_k^t / \|a_k^t\|_1 \tag{1.35}$$

where G_k^t is the stack of gradients received from client k at round t, a_k^t is a non-negative vector representing the local update rate, and τ_{eff} is the effective step size.

MOON

MOON, or Model-contrastive FL, minimizes the contrastive loss between the local and global models to preserve model consistency across clients. The contrastive loss is given by [28]:

$$l_{\text{con}} = -\log \frac{\exp(\text{sim}(w_k^t, w_{\text{glob}}^t)/\tau)}{\exp(\text{sim}(w_k^t, w_{\text{glob}}^t)/\tau) + \exp(\text{sim}(w_k^t, w_k^{\text{prev}})/\tau)} \tag{1.36}$$

where $\text{sim}(\cdot)$ denotes the cosine similarity between models, and τ is a parameter moderating the contrastive loss.

These aggregation methods embody the evolution of FL towards accommodating diverse client conditions and data distributions. Each method introduces unique adjustments to the aggregation process, aiming to enhance convergence, reduce communication overhead, or improve robustness against non-IID data and adversarial clients. The selection of an appropriate aggregation technique is pivotal for the success of FL applications, necessitating a careful consideration of the specific challenges and requirements of the deployment context.

When delving into the intricacies of FL and various aggregation techniques, a set of evaluation metrics comes into play. These insightful metrics measure the effectiveness and efficiency of algorithms and their robustness. Some of the commonly utilized metrics include:

Accuracy: A measure of the model's predictive precision, calculated as the percentage of correctly predicted output to the actual output.

Loss Function: An indicator of the disparity between the predicted outputs and the

actual ones. Frequently used loss functions comprise mean squared error (MSE), cross-entropy loss, and hinge loss.

Convergence Rate: An estimation of the model's speed in attaining an optimal solution. It is typically gauged by the number of training iterations needed to achieve a certain level of accuracy or loss.

Communication Overhead: In FL, data exchange between local devices and the central server is key. This metric quantifies the volume of data shared during the training process.

Privacy Preservation: A key feature of FL is data privacy. By assessing the leakage of sensitive information during the training process, the effectiveness of privacy preservation can be evaluated.

These evaluation metrics offer a broad understanding of the performance of aggregation methodologies in FL. Utilizing these metrics, researchers can critically compare and analyze the effectiveness of different algorithms, leading to informed decisions about their implementation.

Chapter 2

Literature review

Spiking neural networks (SNNs) and their learning algorithms have garnered significant attention for their potential to model complex neural dynamics and learning processes observed in biological systems. This literature review explores various facets of SNNs, focusing on their learning mechanisms, including spike-timing-dependent plasticity (STDP), and their applications in solving computational tasks that mirror biological learning behaviors.

The work of Probst et al. (2015) presents a foundational approach to implementing probabilistic inference in networks of leaky integrate-and-fire (LIF) neurons, illustrating how Bayesian networks can be transformed and computed within spiking neural systems. The study demonstrates the potential of SNNs to perform complex computational tasks, such as probabilistic reasoning, by utilizing the dynamics of spiking neurons and synaptic plasticity, thereby bridging the gap between biological neural processes and computational models [29].

Markowska-Kaczmar and Koldowski (2015) explored the capabilities of spiking neural networks compared to traditional multilayer perceptrons (MLPs) in controlling virtual racing cars within a computer game environment. Their findings suggest that SNNs, despite their simplified neuron model, can outperform MLPs in tasks requiring real-time adaptability and decision-making, highlighting the efficiency of spike-based processing in dynamic environments [30].

Ponulak and Hopfield (2013) introduced an innovative model for rapid, parallel path

planning using wavefronts of spiking neural activity. This model, inspired by hippocampal functioning, demonstrates how SNNs can efficiently solve spatial navigation problems through the propagation of activity waves, offering insights into the neural basis of navigation and the potential for neuromorphic computing applications [31].

Vassiliades, Cleanthous, and Christodoulou (2011) investigated multiagent reinforcement learning using spiking and nonspiking agents within the Iterated Prisoner's Dilemma framework. Their study reveals that SNNs can achieve comparable, if not superior, performance to traditional neural networks in strategic decision-making tasks, emphasizing the role of temporal spike patterns in learning and adaptation [32].

Markram, Gerstner, and Sjöström (2011) provided a comprehensive review of the development and significance of spike-timing-dependent plasticity (STDP) as a fundamental learning rule within neural circuits. Their historical overview underscores STDP's role in synaptic modification based on the precise timing of spikes, contributing to our understanding of learning and memory formation in the brain [33].

Shouval, Wang, and Wittenberg (2010) proposed that STDP emerges as a consequence of more fundamental learning rules that govern synaptic plasticity. By focusing on intracellular calcium levels as mediators of synaptic change, this work challenges the primacy of STDP, suggesting that it represents a manifestation of deeper biophysical processes that regulate synaptic strength [34].

Nordlie, Tetzlaff, and Einevoll (2010) delved into the rate dynamics of leaky integrate-and-fire neurons with strong synapses, examining how synaptic strength influences neural firing patterns. Their findings provide insights into the mechanisms of synaptic integration and the impact of strong synaptic connections on the computational properties of neurons [35].

Izhikevich (2007) addressed the distal reward problem by linking STDP with dopamine signaling, offering a model that captures the essence of reward-based learning in the brain. This work highlights the intricate interplay between neuromodulation and synaptic timing in shaping learning processes, with implications for understanding addiction and motivation

[36].

Song, Miller, and Abbott (2000) investigated competitive Hebbian learning through STDP, revealing how synaptic efficacy is modulated by the timing of pre- and postsynaptic spikes. Their model demonstrates how STDP can lead to competitive learning outcomes without global signaling, emphasizing the significance of temporal dynamics in synaptic modification [37].

DeWolf et al. (2016) present a spiking neural model designed for adaptive arm control. This model integrates anatomical structures and mechanisms from the motor cortex and cerebellum to control a simulated arm. Through the use of a stability framework, the model adapts to dynamic and kinematic changes, such as arm growth or movement through different media. The work demonstrates how spiking neural networks can be applied to complex motor control problems, offering insights into both biological motor control and robotics [38].

Pérez et al. (2018) introduce a bio-inspired SNN for nonlinear system control, specifically focusing on UAV obstacle avoidance. Utilizing a hierarchical structure and learning through Reward-Modulated Spike-Timing-Dependent Plasticity (R-STDP), their network demonstrates high efficiency in processing visual data for control tasks. The research illustrates the potential of SNNs in directly processing sensory inputs for real-time control in robotics, showcasing a significant step towards integrating neuromorphic computing with autonomous systems [39].

Teeter et al. (2018) developed generalized leaky integrate-and-fire (GLIF) models to classify multiple neuron types. By fitting GLIF models to electrophysiological data from a wide range of neuron types, they demonstrate the model's ability to capture the diversity of neuronal behavior. This work contributes to the understanding of how simple neuron models can be used to represent complex neural dynamics, facilitating the study of neural circuits and their computational properties [40].

Mozafari et al. (2018) explored first-spike-based visual categorization using reward-modulated STDP in SNNs. Their work highlights the efficiency of SNNs in performing visual categorization tasks by exploiting the temporal dynamics of spikes. The study shows

that SNNs can achieve high performance in pattern recognition tasks while maintaining low computational costs, making them suitable for neuromorphic hardware implementations [41].

Guo et al. (2017) investigated hierarchical Bayesian inference in SNNs, introducing a learning algorithm based on synaptic plasticity for pattern recognition. Their model demonstrates the potential of SNNs to perform complex computational tasks, such as pattern recognition, through the interplay of hierarchical structure and plasticity. This study underscores the adaptability and efficiency of SNNs in solving computational problems that require hierarchical processing [42].

Koul and Horiuchi (2019) focused on waypoint path planning using synaptic-dependent spike latency in SNNs for UAV control. By simulating the spiking activity of neurons in response to environmental inputs, their model efficiently generates path planning for UAVs. This work exemplifies the application of SNNs in navigation and control tasks, highlighting their potential in real-time decision-making and control in autonomous systems [43].

Tavanaei and Maida (2019) proposed BP-STDP, an algorithm that approximates backpropagation using spike timing dependent plasticity in SNNs. This method bridges the gap between the biologically inspired mechanisms of SNNs and the efficiency of traditional artificial neural networks. Their work contributes to the development of learning algorithms for SNNs, making them more accessible and applicable in machine learning tasks [44].

Bing et al. (2020) studied indirect and direct training methods for SNNs in the context of end-to-end control of a lane-keeping vehicle. Their research demonstrates the effectiveness of both training approaches in utilizing SNNs for real-world control tasks, such as autonomous driving. This study advances the understanding of how SNNs can be trained and applied in complex control systems [45].

Hur et al. (2020) introduced N3-CPL, a neuroplasticity-based learning method for neuromorphic networks, emphasizing cell proliferation in SNNs. This approach simulates the growth and adaptation mechanisms of biological neural networks, enhancing the learning capabilities of SNNs. The work presents a novel direction in neuromorphic computing, focusing on the dynamic adaptation and growth of neural networks [46].

Salt et al. (2019) investigated parameter optimization in an SNN model for UAV obstacle avoidance, targeting neuromorphic processors. By employing various optimization techniques, their work addresses the challenge of tuning SNN parameters for specific tasks. This research highlights the importance of optimization in the effective application of SNNs, especially in tasks requiring real-time processing and decision-making [47].

Liu et al. (2021) introduced a methodology for multi-task autonomous learning in mobile robots using SNNs. Their approach leveraged modified Integrate-and-Fire and Leaky Integrate-and-Fire neuron models, employing a task switch mechanism inspired by lateral inhibition and a novel learning algorithm based on reward-modulated Spike-Timing-Dependent Plasticity (STDP). The system demonstrated efficient autonomous learning and adaptation in tasks like obstacle avoidance and target tracking, highlighting the potential of SNNs in robotic control systems [48].

Lu et al. (2021) presented a framework that combines SNNs with binary Convolutional Neural Networks (CNNs) for reinforcement learning from images and sparse rewards. Their architecture utilizes a pre-trained binary CNN for feature extraction, enhancing the learning efficiency of SNNs in high-dimensional input scenarios. This hybrid model showcases the complementarity of SNNs and CNNs, offering promising directions for efficient and robust learning in visual environments [49].

Zhan et al. (2021) proposed an effective transfer learning framework for SNNs, addressing the challenge of training deep SNNs with limited labeled data. By employing Centered Kernel Alignment for domain distance measurement and optimizing a domain-invariant representation, their work demonstrates the transferability and scalability of SNNs in learning tasks, suggesting a substantial step forward in the development of adaptive and intelligent neural network models [50].

Skatchkovsky, Jang, and Simeone (2021) explored neuromorphic communications, integrating SNNs with wireless communication systems. Their approach aimed at enhancing the energy efficiency and processing capabilities of battery-powered devices in bandwidth-constrained environments. By employing federated learning and neuromorphic sensing, the

study opens new avenues for low-power edge intelligence and efficient communication systems [51].

Chu and Le Nguyen (2021) investigated the constraints on weights learned by Hebbian learning and STDP in spiking neurons, offering a mathematical analysis of the relationships between weight promotion/demotion probabilities and normalized weights. Their findings provide practical methods for optimizing SNN training, potentially advancing unsupervised learning mechanisms in neuromorphic computing systems [52].

Schmidgall et al. (2021) introduced "Spikepropamine," a framework that incorporates differentiable plasticity and neuromodulated synaptic plasticity within SNNs. This approach enables the networks to adaptively learn from environmental interactions, demonstrating significant improvements in task performance and adaptability, paving the way for future advancements in neuromorphic computing and robotics [53].

Golosio et al. (2021) developed NeuronGPU, a GPU-accelerated library for simulating large-scale networks of spiking neurons. Their work highlights the performance and scalability of NeuronGPU in handling complex neuronal dynamics, contributing to the computational neuroscience field by enabling more efficient simulations of SNNs [54].

Shen et al. (2021) proposed HybridSNN, a deep adaptive SNN architecture that combines the strengths of shallow SNNs with machine learning techniques. Through data-driven greedy optimization and a topological tree structure, HybridSNN achieves high performance and energy efficiency in pattern recognition tasks, demonstrating the versatility and potential of SNNs in artificial intelligence applications [55].

Chevtchenko and Ludermir (2021) explored the combination of STDP learning and binary networks in reinforcement learning tasks involving images and sparse rewards. Their methodology leverages the unique processing capabilities of SNNs for efficient learning from high-dimensional visual inputs, offering a novel approach to reinforcement learning with potential applications in visual perception systems [56].

Gupta et al. (2021) introduced QC-SANE, a robust control framework in DRL utilizing quantile critic with spiking actor and normalized ensemble. Their approach significantly

enhances the performance and robustness of DRL systems, particularly in continuous control problems, demonstrating the potential of combining SNNs with advanced learning algorithms for efficient and robust artificial intelligence systems [57].

Chapter 3

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

3.1 Introduction

Moving from Chapter 3, where we looked at controlling one drone (the leader) using a 5G network, Chapter 4 takes us into SNNs and how they can be used to control drones (the followers). In Chapter 3, we carefully studied how well different types of controllers worked for quadrotors with cellular and 5G networks. This study showed us that the leader drone could be efficiently controlled remotely while satisfying control and communication constraints.

Expanding on what we learned, this chapter looks at two main things: how well SNNs handle noise, which is important because drones operate in real-world conditions with lots of disturbances, and how they can handle complex scenarios like optimal decision-making. This detailed study shows us the potential of SNNs in managing drones' 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 [58].

In the ATD, a defender tries to protect a target, while a superior invader with higher velocity than the defender and target tries to reach the target and escape the defender at the same time. In defense strategy, they aid in analyzing movements and determining optimal strategies for aircraft. Law enforcement benefits from these games by minimizing escape possibilities. They 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. [59].

There are three distinct outcomes to the ATD, characterized by two or three termination sets [60]. 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. Finally, the third outcome is realized when the invader and defender reach the target at the same time.

3.2 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 [60]. 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 3.1 shows LOS angles used as the input for the SNNs [61].

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

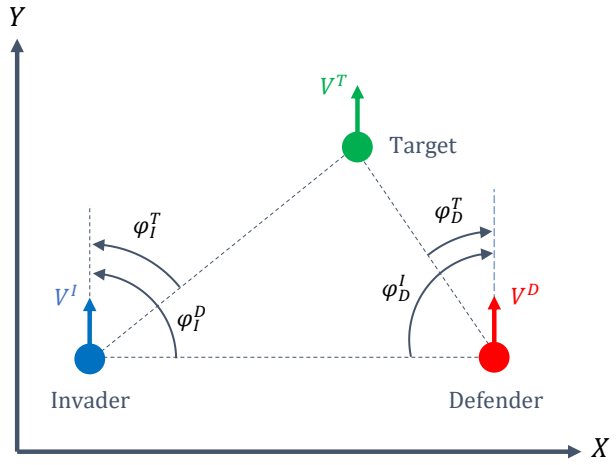

Figure 3.1: Active target defense game with three agents (LOS angles are shown for both agents).

and the target) and calculate the steering angle for the agent (Figure 3.2). In this figure, the ϕ_I^T represents the LOS angle to the target relative to the invader.

3.3 Learning using R-STDP

R-STDP is a biological learning algorithm that is believed to underlie certain learning mechanisms in the brain [62]. The R-STDP algorithm is based on the idea that if a pre-synaptic neuron fires just before a post-synaptic neuron, the strength of the synapse between the two neurons should be increased. Conversely, if the post-synaptic neuron fires just before the pre-synaptic neuron, the strength of the synapse should be decreased.

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 [22],

$$\dot{C}^{i-j} = -C^{i-j}/\tau_C + STDP^{i-j}(\tau)\delta(t - t_{pre/post}) \tag{3.1}$$

, where the C^{i-j} is the eligibility trace for the synaptic connection between neurons i and j, τ, is the spike timing difference between the input and output spike times, τ_C is the time

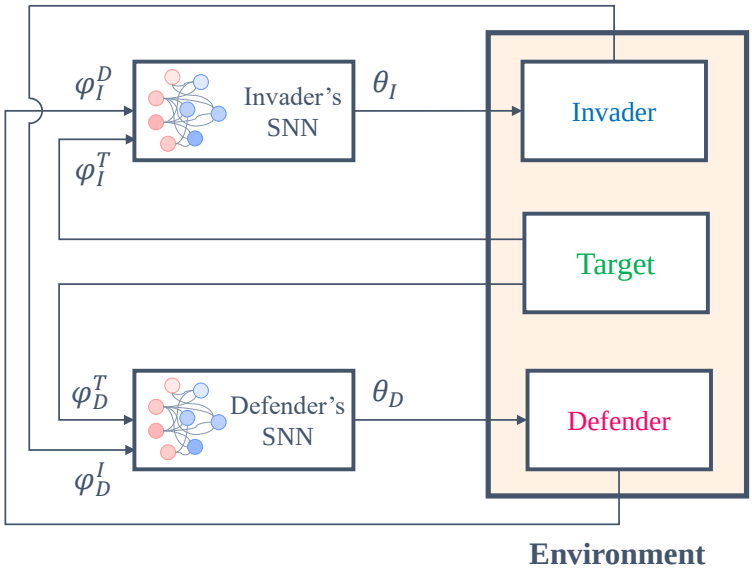

Figure 3.2: Two SNNs simultaneously play and control the invader and defender. Each agent uses the LOS angle and relative velocities for training.

constant for the synaptic plasticity, δ 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,

$$STDP^{i-j}(\tau) = \begin{cases} A_+ exp\left(\frac{-\tau}{\tau_s}\right) & \text{if } \tau > 0 \\ A_- exp\left(\frac{\tau}{\tau_s}\right) & \text{if } \tau < 0 \end{cases} \tag{3.2}$$

where A_+ and A_- are the amplitude of the exponential function, and τ_s is the time constant that determines the decaying rate of the R-STDP function. If τ_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,

$$\dot{W}^{i-j}(t) = C^{i-j}(t)R(t) \tag{3.3}$$

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 [62],

$$\overrightarrow{W}^n(t) = \overrightarrow{W}^n(t-1)\left(\frac{I^{max}}{\Sigma_{\iota=1}^{N} W^{\iota}(t)}\right) \tag{3.4}$$

where $W^n(t)$ 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 along the LOS direction. Figure 3.3 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,

$$R_I^T(t) = \eta_I^T\left(V^I + V^T\right) cos\left(\phi_I^T\right) \tag{3.5}$$

and

$$R_I^D(t) = \eta_I^D\left(V^D - V^I\right) cos\left(\phi_I^D\right) \tag{3.6}$$

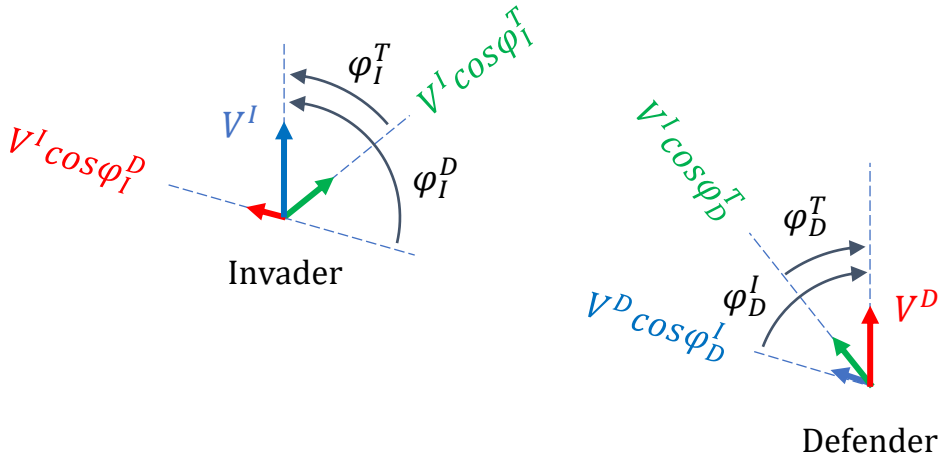

Figure 3.3: Relative velocities and LOS angles used in the reward function

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

$$R_D^T(t) = \eta_D^T \left(V^D + V^T\right) cos\left(\phi_D^T\right) \tag{3.7}$$

and

$$R_D^I(t) = \eta_D^I \left(V^D - V^I\right) cos\left(\phi_D^I\right) \tag{3.8}$$

In (3.5)-(3.8), η_I^T, η_I^D, η_D^T, and η_D^I are constant coefficients. Adjusting these values changes the agent's attention to other agents. For example, in the invader case, increasing the η_I^T and decreasing the η_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),

$$\mathbf{W}_I^T(t) = \mathbf{W}_I^T(t-1) + \mathbf{C}_I^T(t) R_I^T(t) \tag{3.9}$$

and

$$\mathbf{W}_I^D(t) = \mathbf{W}_I^D(t-1) + \mathbf{C}_I^D(t) R_I^D(t) \tag{3.10}$$

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.

3.4 Network structure and encoding method

Figure 3.4 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) [63]. 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. Therefore, 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.

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,

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

or

$$I_\sigma = \frac{\tau_m \left(V_{th} - V_{res}\right)}{\Delta t R_m} FMV_\sigma + \frac{V_{th} - E_l}{R_m} \tag{3.11}$$

where σ is the index of each neuron in the input layer and its corresponding fuzzy membership

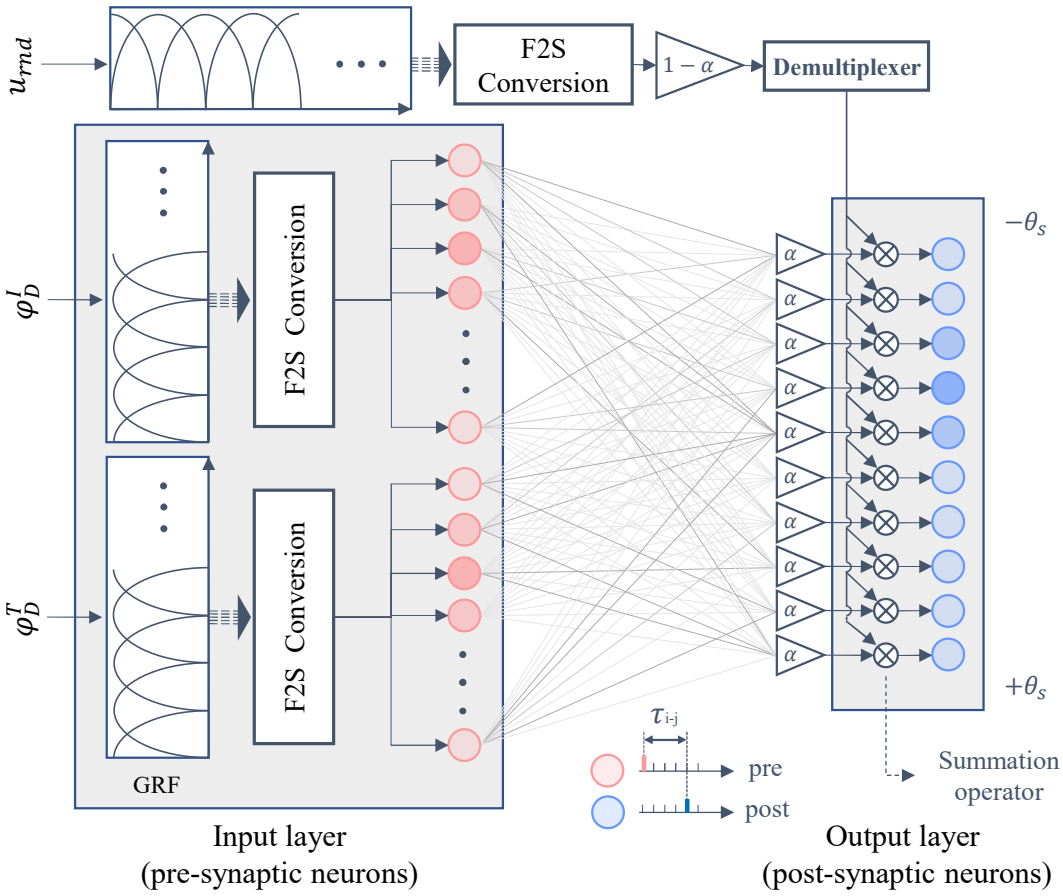

Figure 3.4: Network structure and encoding process for the input layer (Defender). Each neuron is associated with a membership function in GRF. The GRF encodes an input State (S^t) at each time step. There is both a training phase when ($\alpha = 0$) and an operating phase when ($\alpha = 1$).

value in GRF.

The output of the SNN is the steering angle (θ_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 τ_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,

$$I_{out} = \alpha I_{syn} + (1 - \alpha)\, I_{rnd} \tag{3.12}$$

where α 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 α is 0 during the training, the I_{syn} does not affect the SNN's output (Equation 3.12).

In the training process, the output neurons 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 3.8. 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

Algorithm 1 Weight training algorithm

for $t = 0 : \Delta t : t_{final}$ **do**
  input ϕ_I^T and ϕ_I^D
  $\zeta \leftarrow$ number of input neurons
  $\kappa \leftarrow$ number of output neurons
  for i = $0{:}2\pi/\zeta{:}2\pi$ **do**
    $Z_T(i,1) = exp\left(-0.5(\phi_I^T - i)/\sigma\right)$
    $Z_D(i,1) = exp\left(-0.5(\phi_I^D - i)/\sigma\right)$
  end for
  $I^T = (I^{max} - I^{min})\, Z_T + I^{min}$
  $I^D = (I^{max} - I^{min})\, Z_D + I^{min}$
  Generate a uniform random number for exploration $u_{rnd} \in [-\theta_s, \theta_s]$
  $j = 1$
  for i = $-\theta_s{:}2\theta_s/\kappa{:}\theta_s$ **do**
    $Z_{rnd}(j,1) = exp\left(-0.5(u_{rnd} - i)/\sigma\right)$
    $j \leftarrow j + 1$
  end for
  $I_{rnd} = (I^{max} - I^{min})\, Z_{rnd} + I^{min}$
  for j = 1 to κ **do**
    $I_{syn}(j,1) = \sum_{i=1}^{\zeta} W^{i-j}\delta(t - t_i)$ ▷ t_i is the firing time of the i^{th} input neuron
  end for
  $I_{in} = \begin{bmatrix} I^T \\ I^D \end{bmatrix}$ Input current
  $I_{out} = \alpha I_{syn} + (1 - \alpha) I_{rnd}$ ($\alpha = 0$ in training phase)
  $\mathbf{V_m}(t + \Delta t) = (1 - \frac{\Delta t}{\tau_m})\mathbf{V_m}(t) + \frac{\Delta t}{\tau_m}\left(\mathbf{E_l} + R_m \begin{bmatrix} I_{in} \\ I_{out} \end{bmatrix}\right)$ {$\mathbf{V_m}$ and $\mathbf{E_l}$ are $[\zeta + \kappa] \times 1$}
  Find fired neurons in input and output layer
  Calculate the $\boldsymbol{\tau}$ matrix that shows the difference in firing time between the fired input neurons and fired output neurons (Figure 3.4)
  Calculate $\mathbf{R - STDP}$ for all connection using (3.2)
  Calculate $\mathbf{C}$ matrix using (3.1) for all the connections
  Calculate $\mathbf{W}$ matrix using (3.3) for all the connections considering reward from (3.5) to (3.8)
  for i = 1 to κ **do**
    if sum of the input weights to ith neuron $\geq I^{max}$ **then**
      Normalize input weights of ith neuron using (3.4)
    end if
  end for
  Set voltage of the fired neurons to reset voltage (V_{res})
end for

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

Parameter	Value	Description
R_m	40 MΩ	Membrane Resistance
τ_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

weight for the input to output neurons in (3.2) is weakened. Future research will include an inhibitory effect where the weights can become negative.

After training, the α 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.

3.5 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 I.

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 ϕ_I^T, and the other half is for the ϕ_I^D. Half of the defender's input neurons are for the ϕ_D^T, and the other half is for the ϕ_D^I. Since each network contains 20 input neurons, the input

Table 3.2: Parameter values for R-STDP

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

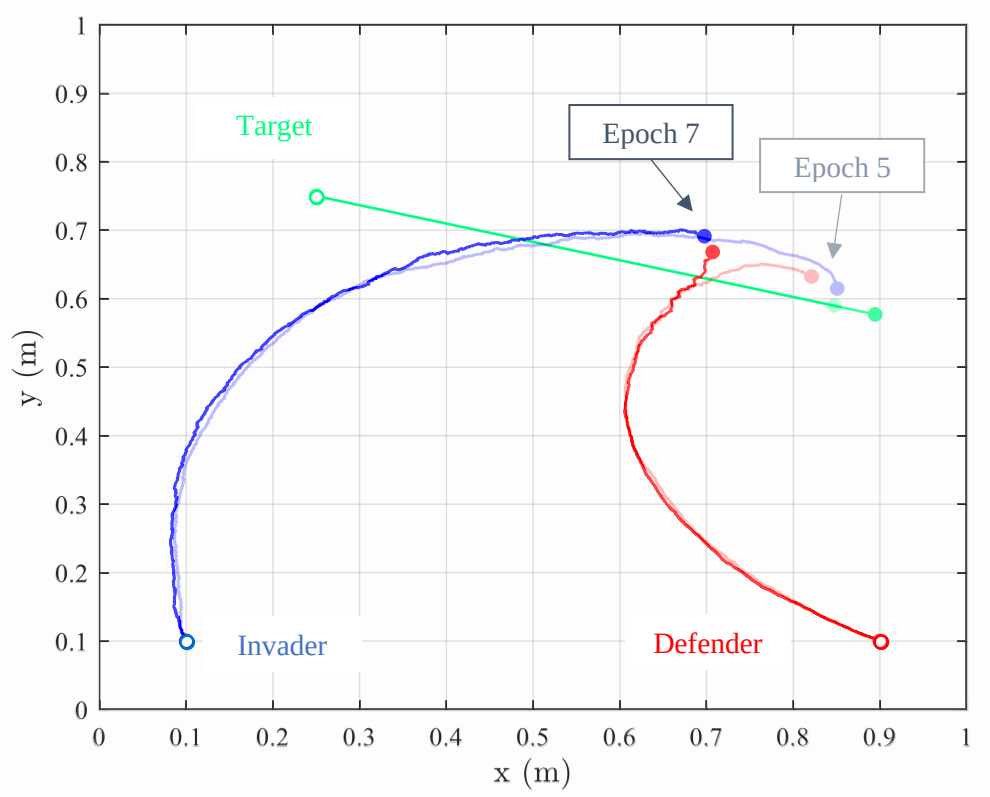

Figure 3.5: SNN's performance during training. Hollow circles show the initial positions.

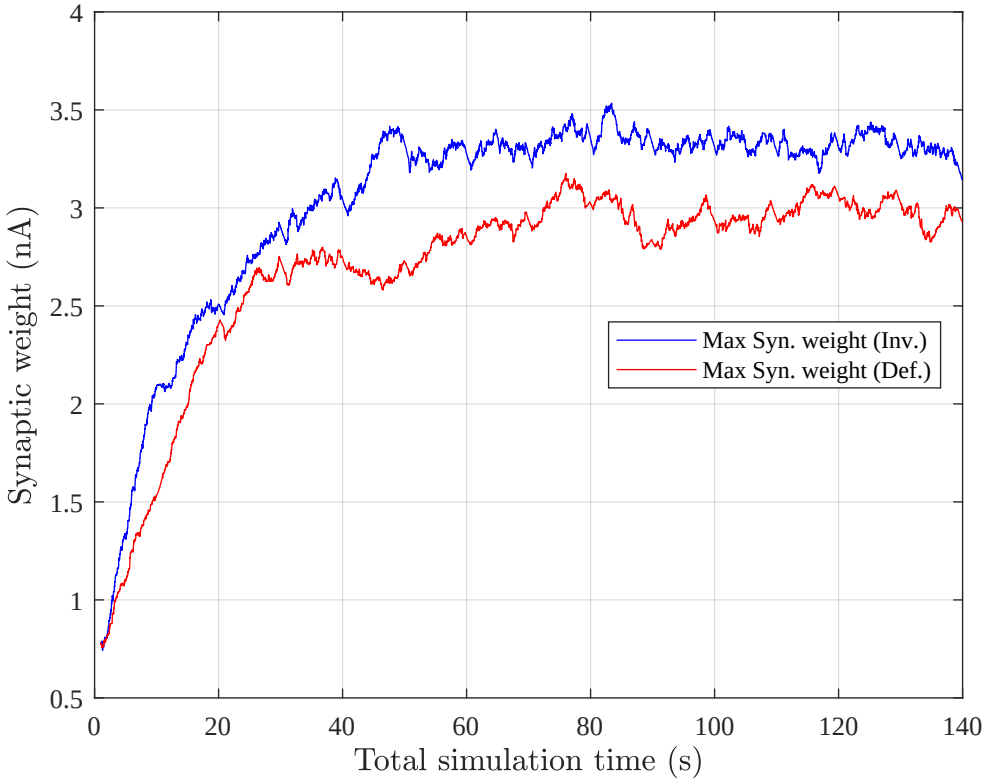

Figure 3.6: Changes in synaptic weights during training. Only Maximum synaptic weights for both agents are shown.

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 m/s. The invader's velocity is 0.3 m/s. The γ is 0.75, so the defender's velocity is 0.225 m/s. The defender's capture radius (ρ) is set to 0.025 m. The maximum simulation time for each epoch is 10 seconds. The θ_s for invader and defender is $\pi/4$, and the σ is set to 1.25. The parameter values for R-STDP, shown in Table II, are set through several simulations.

The reward coefficients are set manually. The τ_s in Table II defines the decaying rate of C in (3.1) that determines the R-STDP sensitivity to prior firings. According to (3.2), higher τ_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 II, the optimal capture point is the closest point from the reachable region to the target position. Figure 3.5 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.

3.5.1 Simulation without noise

Figure 3.9 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

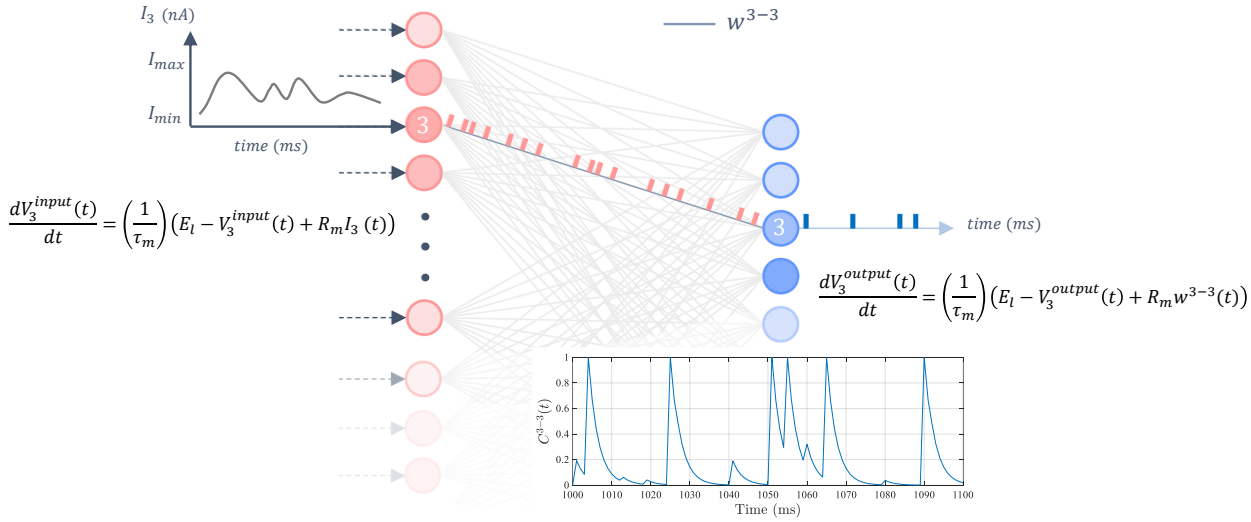

Figure 3.7: An illustration of input current to SNN and synaptic current to the output neurons after training for a single connection.

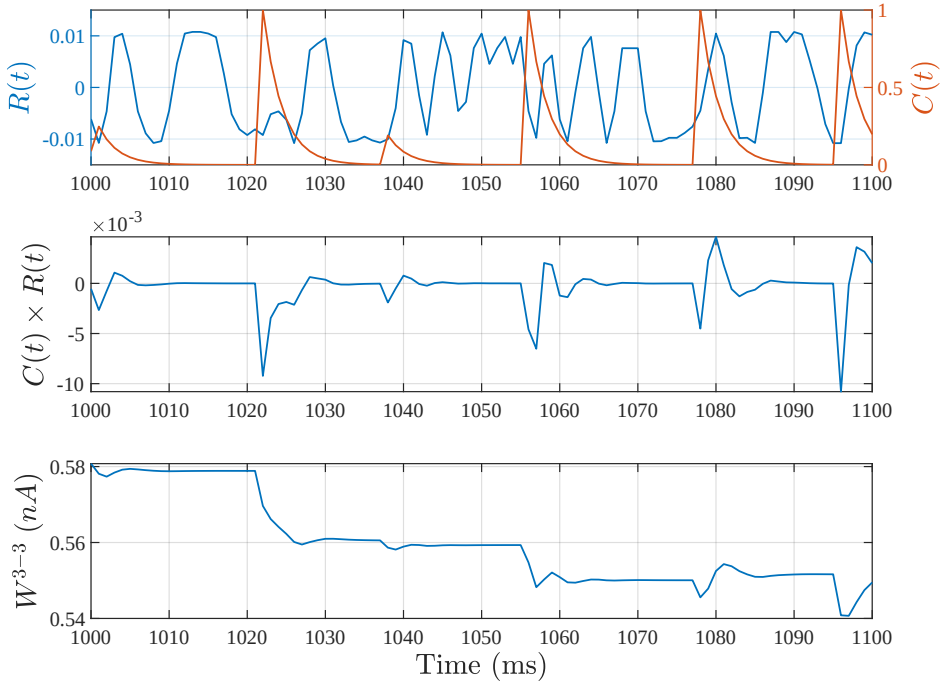

Figure 3.8: Reward, eligibility trace, and weight change during the simulation for W^{3-3}. The $C(t)R(t)$ changes the synaptic weight by considering activation strength and reward value.

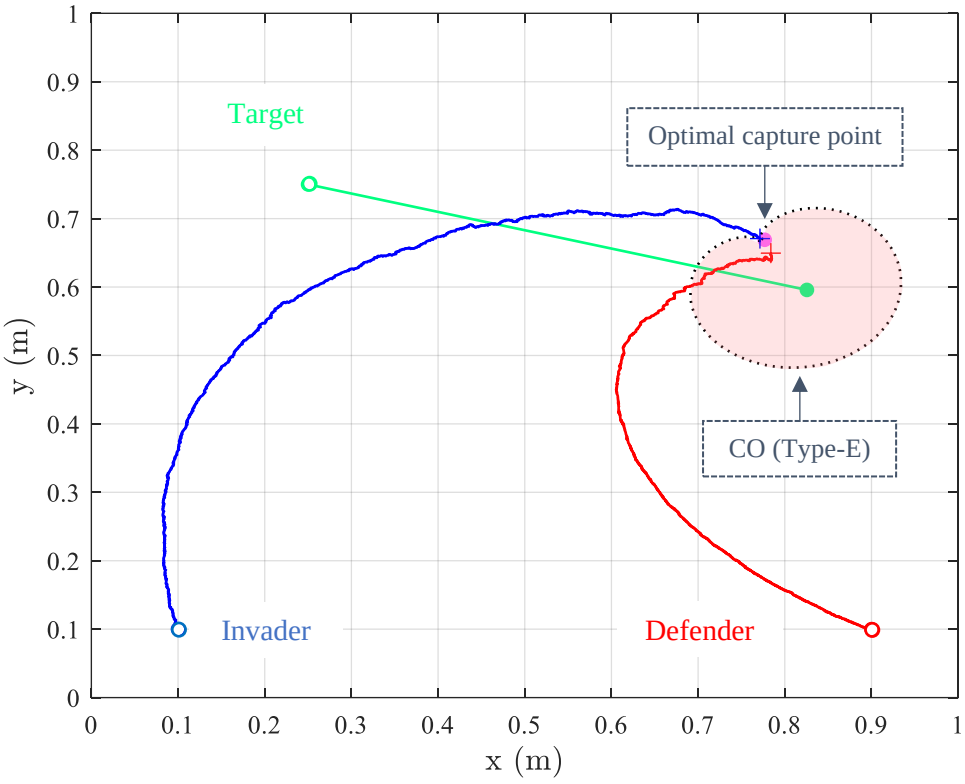

Figure 3.9: SNN's performance after training. The highlighted region shows the defender's dominant region. The purple dot is the optimal capture point.

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 3.9, the invader's SNN can find the optimal [60] 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 3.6 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 the figure, after almost 100 seconds of simulation time, the MSN process causes the synaptic weights to converge.

3.5.2 Simulation with noise

In Figure 3.10, 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.

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.

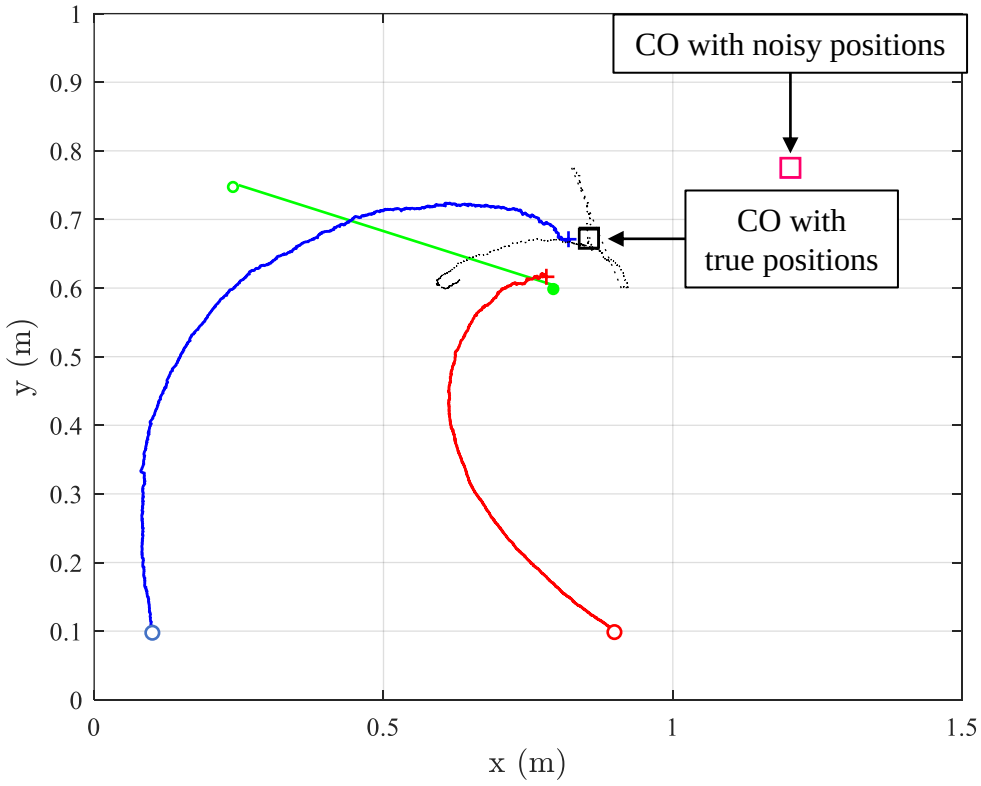

Figure 3.10: SNN's performance in noisy conditions. A white Gaussian noise with a variance of 0.01 is added to the measured inputs.

3.6 Conclusion

This study 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 an analytical solution 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 4

Integration of R-STDP and Federated Learning

4.1 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 4.1, 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 from neighboring agents.

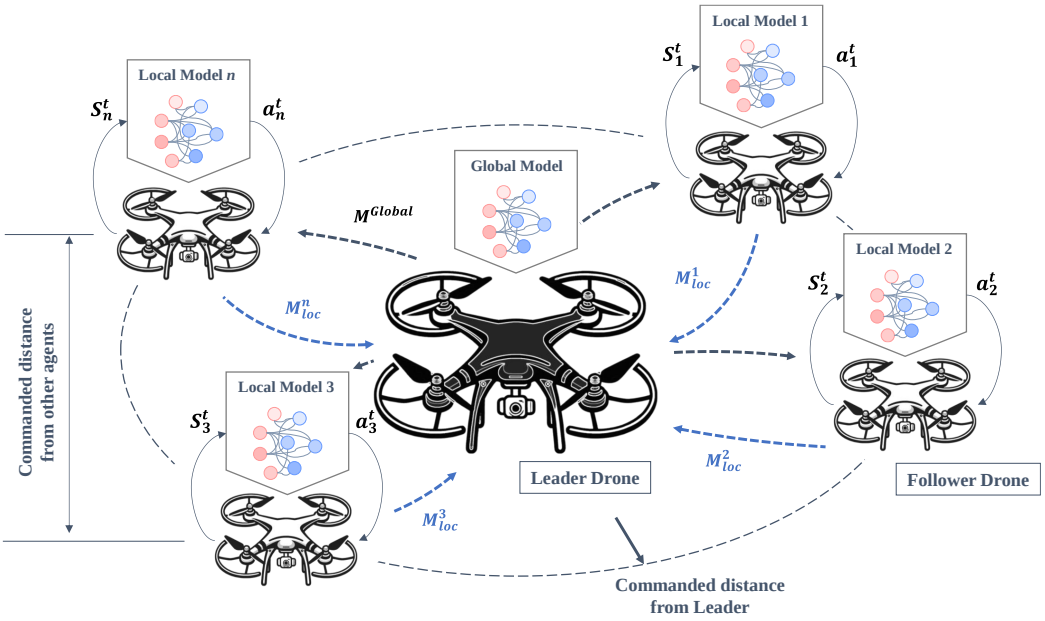

Figure 4.1: The central server (the leader) and the surrounding follower agents (white drones). The follower agents learn to fly in a formation to maintain the commanded distance. The Local models trained individually by follower agents are sent to the leader. The leader aggregates the models and sends 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^n_{loc}) 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 considered 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 an innovative decay rate adjustment based on the status of synaptic weights. This approach not only enhances the responsiveness to weight changes but also preserves the shape of the Gaussian function formed by fuzzy encoding.

- 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, which improves network usage by the agents. 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.

4.2 Proposed Method

4.2.1 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 angle and the distance. Each agent's neural network consists of three sub-layers in the input layer, as shown in Figure 4.2. 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.

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 Line-of-Sight angle is encoded with

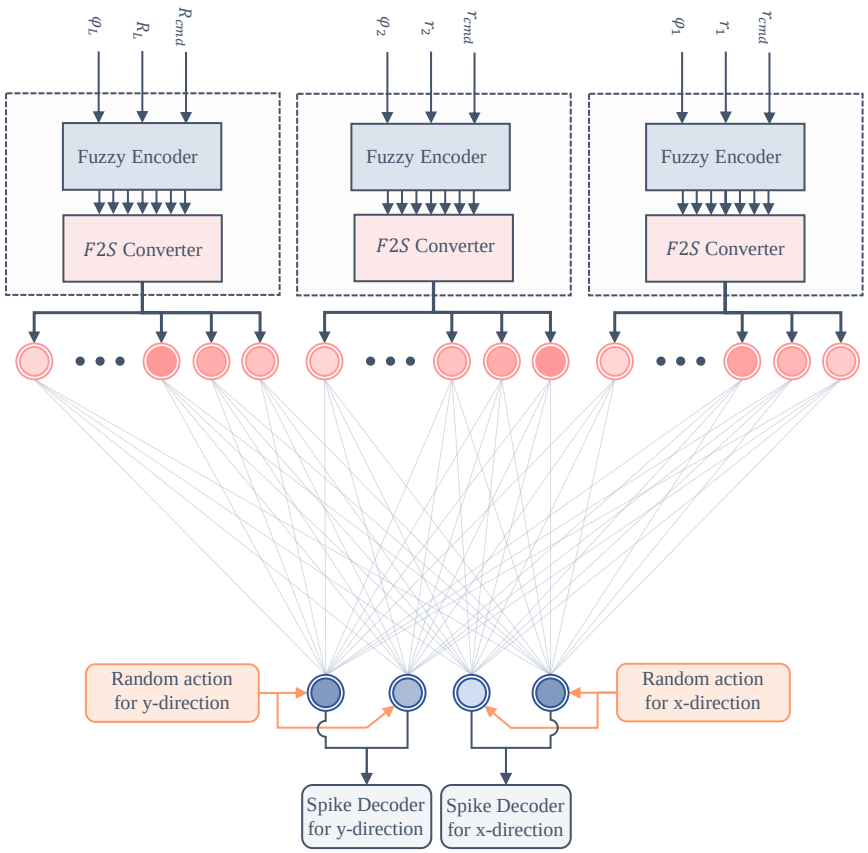

Figure 4.2: SNN structure with encoding and decoding layers. Each sub-layer consists of a fuzzy encoder and a Fuzzy-to-Spiking Converter, with the output layer receiving inputs from synaptic weights and a random action selector. During the training phase, the output layer receives input only from the random action selector, which then shifts to synaptic weight inputs after the training.

fuzzy membership functions. The difference between the current and commanded distance is represented as the error that changes the maximum amplitude of the fuzzy membership functions. 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,

$$\mu_I(\phi_i, r_i) = |tanh(r - r_i)| \cdot \exp\left(-\frac{(\phi_i - \zeta)^2}{2\sigma^2}\right) \tag{4.1}$$

where ζ and σ are the Gaussian membership functions' center and standard deviation. The r_i is the distance from the corresponding agent, ϕ_i is the line-of-sight angle, and μ_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 [64],

$$I_{sub-layer} = \left(I^{max} - I^{min}\right) \mu_I(\phi_i, r_i) + I^{min} \tag{4.2}$$

or

$$I_{sub-layer} = \frac{\tau_m \left(V_{th} - V_{res}\right)}{\Delta t R_m} \mu_I(\phi_i, r_i) + \frac{V_{th} - E_l}{R_m} \tag{4.3}$$

The encoding process is shown in Figure 4.3. The Fuzzy to Spiking (F2S) block uses (4.3) 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 Δx, and the second one determines Δy. 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 Δx and Δy is encoded into the output sub-layers based on the minimum and maximum synaptic weights. Equation (4.3) is used to encode the magnitude of the random action into the output sub-layers. The only difference is that a function called (μ_O) is used to normalize the maximum step between 0

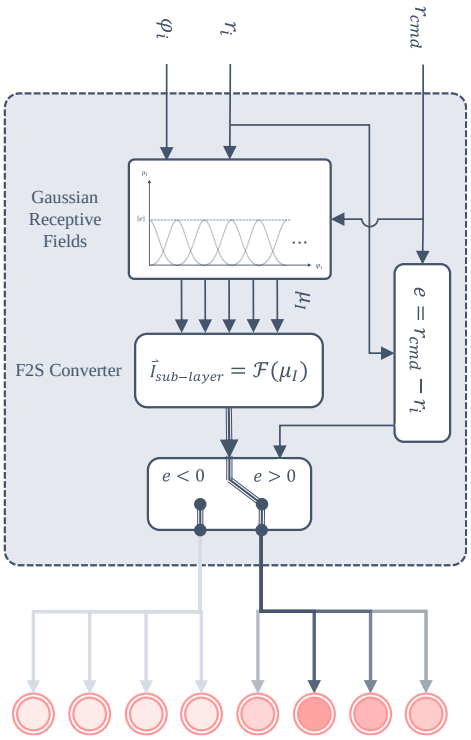

Figure 4.3: The fuzzy encoding principle for the input sub-layer.

and 1 as follows,

$$\mu_x = \frac{\Delta x}{\Delta X_{max}} \tag{4.4}$$

$$\mu_y = \frac{\Delta y}{\Delta Y_{max}} \tag{4.5}$$

where Δx and Δy are selected actions, and ΔX_{max} and ΔY_{max} are maximum steps (displacements) in X and Y directions. Therefore, two random actions, one for Δx and one for Δy, 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 $f^{x+}(t)$ and $f^{x-}(t)$ as the firing rates of the first and second output neurons, respectively, in the x-direction sub-layer at time t. The equation for decoding this activity can be expressed as:

$$\Delta x_{decoded} = \left[\sum_{i=t-\Delta T}^{t} \left(f^{x+}(i) - f^{x-}(i) \right) \right] \Delta X_{max} \tag{4.6}$$

A similar process is applied for decoding in the y-direction:

$$\Delta y_{decoded} = \left[\sum_{i=t-\Delta T}^{t} \left(f^{y+}(i) - f^{y-}(i) \right) \right] \Delta Y_{max} \tag{4.7}$$

The dynamic model of agents can be represented as,

$$\dot{X}(t) = \frac{\Delta x_{decoded}}{\Delta T} \tag{4.8}$$

$$\dot{Y}(t) = \frac{\Delta y_{decoded}}{\Delta T} \tag{4.9}$$

where ΔT 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. To address this, the recursive random number generation method is used to produce correlated random numbers. This method ensures that 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,

$$\amalg_t = \gamma \cdot \amalg_{t-1} + (1 - \gamma) \cdot \epsilon_t \tag{4.10}$$

where $\amalg_t$ is the random action at time t, γ is a correlation coefficient, and ϵ_t is a random number drawn from a standard distribution (e.g., Gaussian) at time t. This equation ensures that the random action at any given time t is a weighted combination of the previous action and a new random number.

4.2.2 Training algorithm

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

$$\mathcal{R}_{Fi}^{Fj}(t) = \mathcal{C}_{Fi}^{Fj}\left[r_{Fi}^{Fj}(t-1) - r_{Fi}^{Fj}(t)\right] tanh(r_{Fi}^{Fj}(t) - r_{cmd}) \tag{4.11}$$

$$\mathcal{R}_{Fi}^{L}(t) = \mathcal{C}_{Fi}^{L}\left[r_{Fi}^{L}(t-1) - r_{Fi}^{L}(t)\right] tanh(r_{Fi}^{L}(t) - R_{cmd}) \tag{4.12}$$

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 $\mathcal{C}_{Fi}^{Fj}$ and $\mathcal{C}_{Fi}^{L}$ show the 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.

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

Controlling the excessive increase of synaptic weights in SNNs is important to maintain network stability 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, leading to all synaptic weights eventually maxing out. 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, the Bienenstock, Cooper, and Munro (BCM) method, a more advanced approach, 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 gradient in the synaptic weights matrix formed due to differences in firing strength from fuzzy membership functions while dynamically adjusting the network when the reward changes.

Let us define $\boldsymbol{\alpha}$ as the learning rate matrix. We will also use $\mathcal{S}$ to represent the set of neurons that fired at time t in one of the network sections. If we consider $W^{\mathcal{S}}_{max}(t)$ as the maximum weight among the firing neurons in set $\mathcal{S}$, then we can characterize the learning rate using a Sigmoid function. This function gradually transitions from 1 to 0 as the learning process advances, as explained below:

$$\boldsymbol{\alpha}^{\mathcal{S}}(t) = \frac{1}{1 + \exp\left(|W^{\mathcal{S}}_{max}(t)| - \Psi^{\mathcal{S}}\right)}, \quad \left(\Psi^{\mathcal{S}} = \frac{\mathcal{R}^{\mathcal{S}}_{max}}{\mathcal{R}^{G}_{max}} I^{max}\right) \tag{4.13}$$

where $\mathcal{R}^{\mathcal{S}}_{max}$ is the maximum reward in the network section (e.g., $max(\mathcal{R}^{Fj}_{Fi})$), and $\mathcal{R}^{G}_{max} = max(\mathcal{R}^{Fj}_{Fi}, \mathcal{R}^{L}_{Fi})$. 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 in specific connections sets the limit for the synaptic weight in that area. The synaptic weight limit is linked to the ratio of the local maximum reward to the global maximum reward. As a result, the network section with the highest local maximum reward ($\mathcal{R}^{\mathcal{S}}_{max} = \mathcal{R}^{G}_{max}$) attains the maximum synaptic weights ($\Psi^{\mathcal{S}} = I^{max}$), while sections with lower local maximum rewards reach only a proportional fraction of the maximum weight. This proportion is based on the local maximum reward in relation to the global maximum reward. The adjustment of the learning rate transforms into a competitive algorithm, modifying the growth rate of individual synaptic weights by considering partial network parameters.

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 remaining at their peak values indefinitely. 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 ($|W^{\mathcal{S}}_{max}(t)| = \Psi^{\mathcal{S}}$), 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. Simultaneously, the learning rate must not exceed the maximum acceptable weight change, which is $\mathcal{A} \times \mathcal{R}^G_{max}$. With these considerations, we propose that the decay rate should be set to $\mathcal{A}/\lambda \times \mathcal{R}^G_{max}$ when $|W^{\mathcal{S}}_{max}(t)| = \Psi^{\mathcal{S}}$ and increase to $\lambda\mathcal{A} \times \mathcal{R}^G_{max}$ when $|W^{\mathcal{S}}_{max}(t)| = 2\Psi^{\mathcal{S}}$, where λ is a coefficient that controls the rate of decay when $|W^{\mathcal{S}}_{max}(t)| > \Psi^{\mathcal{S}}$. By applying the mentioned condition and solving for the SoftPlus function, the decay rate function can be obtained as follows,

$$\Theta^{\mathcal{S}} = \left(\frac{\eta}{\beta}\right) \log\left(1 + \exp\left[\beta\left(|W^{\mathcal{S}}_{max}(t)| - \Psi^{\mathcal{S}}\right)\right]\right) \tag{4.14}$$

where $\eta = \frac{\mathcal{A}\mathcal{R}^G_{max} \ln\left(2^{\lambda^2}-1\right)}{\lambda \Psi^{\mathcal{S}} \log(2)}$ and $\beta = \frac{\ln\left(2^{\lambda^2}-1\right)}{\Psi^{\mathcal{S}}}$. Therefore, (4.14) can be represented as,

$$\Theta^{\mathcal{S}} = \left(\frac{\mathcal{A}\mathcal{R}^G_{max}}{\lambda \log(2)}\right) \log\left(1 + \exp\left[\left(\frac{\ln\left(2^{\lambda^2}-1\right)}{\Psi^{\mathcal{S}}}\right)\left(|W^{\mathcal{S}}_{max}(t)| - \Psi^{\mathcal{S}}\right)\right]\right) \tag{4.15}$$

The choice of setting the decay rate to $\lambda\mathcal{A} \times \mathcal{R}^G_{max}$ when $|W^{\mathcal{S}}_{max}(t)| = 2\Psi^{\mathcal{S}}$ is based on the feature of reward coefficients. Specifically, when the reward coefficient increases, leading to $|W^{\mathcal{S}}_{max}(t)| < \Psi^{\mathcal{S}}$, the $|W^{\mathcal{S}}_{max}(t)|$ is allowed to increase. Conversely, a decrease in the reward coefficient, resulting in $|W^{\mathcal{S}}_{max}(t)| > \Psi^{\mathcal{S}}$, necessitates a higher decay rate to reduce the maximum synaptic weight back to $\Psi^{\mathcal{S}}$. Incorporating the RCSE method into (**??**), the enhanced version of the R-STDP method is expressed as follows,

$$\dot{\boldsymbol{W}}(t) = \boldsymbol{\alpha} \odot \mathbf{STDP}(\tau) \odot \mathcal{R}(t) - \boldsymbol{\Theta}_{ij} \odot \operatorname{sgn}(\boldsymbol{W}) \tag{4.16}$$

where $\odot$ is the Hadamard product.

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

Proof of stability of the RCSE method

***Lemma**: The equilibrium point $W^{\mathcal{S}}_{max}(t) = \Psi^{\mathcal{S}}$ of the RCSE method is asymptotically stable, as the derivative of the Lyapunov function is non-positive.*

We consider the dynamical system given by the equation of synaptic weights for $W^{\mathcal{S}}(t) \geq 0$ as,

$$\begin{aligned}\dot{W}^{\mathcal{S}}(t) =& \frac{1}{1+\exp\left[\frac{1}{\epsilon}\left(W^{\mathcal{S}}_{max}(t)-\Psi^{\mathcal{S}}\right)\right]} STDP(\tau)\mathcal{R}(t) \\ &- \left(\frac{\mathcal{AR}^{G}_{max}}{\lambda\log(2)}\right)\log\left(1+\exp\left[\left(\frac{\ln\left(2^{\lambda^2}-1\right)}{\Psi^{\mathcal{S}}}\right)\left(W^{\mathcal{S}}_{max}(t)-\Psi^{\mathcal{S}}\right)\right]\right),\end{aligned} \tag{4.17}$$

To assess the stability of this system around the equilibrium point $W^{\mathcal{S}}_{max}(t) = \Psi^{\mathcal{S}}$, we introduce a Lyapunov function candidate $V(z)$, where $z = W^{\mathcal{S}}_{max}(t) - \Psi^{\mathcal{S}}$. A common choice for such analyses is a quadratic function of the deviation from the equilibrium:

$$V(z) = \frac{1}{2}z^2. \tag{4.18}$$

This function is positive definite and has a minimum at the equilibrium point, satisfying the essential criteria for a Lyapunov function. The derivative of $V(z)$ with respect to time, $\dot{V}(z)$, is then calculated to determine the rate of change of the Lyapunov function along the trajectories of the system:

$$\dot{V}(z) = z\dot{z} \tag{4.19}$$

Considering the maximum value of $STDP(\tau)\mathcal{R}(t)$ as $\mathcal{AR}^{G}_{max}$, and substituting $\dot{W}^{\mathcal{S}}(t)$ from Eq. 4.17 into the above expression, we have,

$$\dot{V}(z) = z \times \left[\frac{1}{1+\exp\left[\frac{1}{\epsilon}(z)\right]}\mathcal{AR}^{G}_{max} - \left(\frac{\mathcal{AR}^{G}_{max}}{\lambda\log(2)}\right)\log\left(1+\exp\left[\left(\frac{\ln\left(2^{\lambda^2}-1\right)}{\Psi^{\mathcal{S}}}\right)(z)\right]\right)\right] \tag{4.20}$$

A negative $\dot{V}(z)$ for all $W^{\mathcal{S}}(t) \neq \Psi^{\mathcal{S}}$ indicates that the system's energy decreases over time, leading to the conclusion that the equilibrium point is asymptotically stable. Conversely, a positive $\dot{V}(z)$ in any region would suggest the presence of instability or regions of attraction that do not encompass the entire state space.

In the analysis of the system's stability, we focus on the behavior of the derivative of the Lyapunov function, $\dot{V}(z)$, across different regions of z. We decompose the dynamics of $\dot{z}$ into its constituent components to systematically analyze the stability conditions. we assess the relative magnitudes of the two main components influencing $\dot{V}(z)$:

1. The first term, represented as $\frac{1}{1+\exp\left[\frac{1}{\epsilon}z\right]}\mathcal{AR}^{G}_{max}$, denotes the effect of learning rate and is inherently positive.
2. The second term, $\left(\frac{\mathcal{AR}^{G}_{max}}{\lambda \log(2)}\right) \log\left(1+\exp\left[\left(\frac{\ln(2^{\lambda^2}-1)}{\Psi^{\mathcal{S}}}\right) z\right]\right)$, captures the dynamic decay rate of synaptic weights, which is governed by the SoftPlus function.

Analysis for $z < 0$:

For $z < 0$, we observe that the second term is zero. This implies that the contribution of this term to $\dot{V(z)}$ is negligible in this region. Moreover, the first term remains positive throughout, and given that it is multiplied by z (which is negative in this region), the overall contribution to $\dot{V}(z)$ is negative. Consequently, $\dot{V}(z)$ is negative for $z < 0$, indicating that any perturbations from the equilibrium in this region will decrease over time, thereby contributing to the system's stability.

Analysis for $z > 0$**:**

For this region, the magnitude of the second term significantly exceeds that of the first term. This predominance is critical as it is associated with a negative sign in the $\dot{V}(z)$ equation. Therefore, the negative contribution of this component ensures that $\dot{V}(z)$ remains negative throughout this region. It indicates that any deviation from the equilibrium state results in the system's energy decreasing over time, leading to the conclusion that the equilibrium point $W^{\mathcal{S}}_{max}(t) = \Psi^{\mathcal{S}}$ is asymptotically stable for $z > 0$.

Figures 4.4 and 4.5 demonstrate the performance of the RCSE when it regulates the synaptic weights to prevent unbounded growth. The red dotted line represents the value of

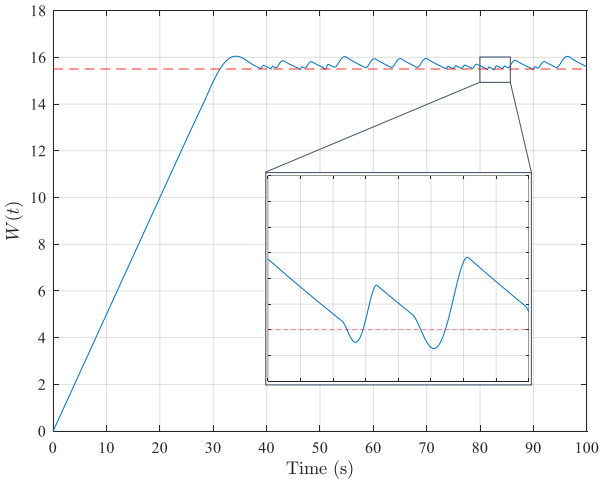

Figure 4.4: Synaptic weight change for $\lambda = 5$, $\Psi^{\mathcal{S}} = 15.5$, and $\mathcal{AR}^{G}_{max} = 0.5$

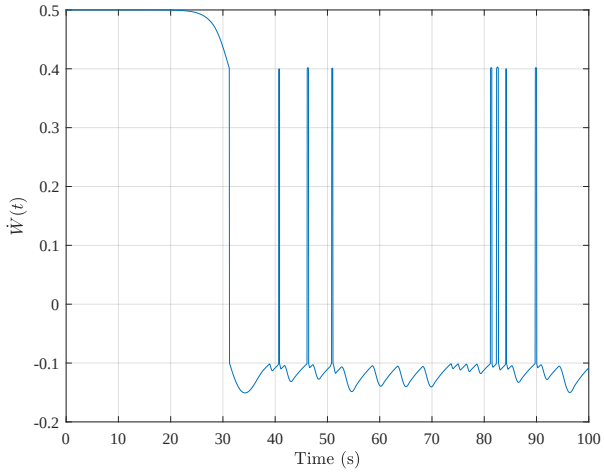

Figure 4.5: Derivation of Synaptic weight for $\lambda = 5$, $\Psi^{\mathcal{S}} = 15.5$, and $\mathcal{AR}^{G}_{max} = 0.5$

$\Psi^{\mathcal{S}}$, which is derived from the maximum reward value of the corresponding network section. As shown in Figure 4.4, the synaptic weight oscillates around $\Psi^{\mathcal{S}}$, and when it drops below $\Psi^{\mathcal{S}}$, the learning rate is set to 1. This allows any changes in the reward function to be applied to the synapse.

4.2.4 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.

Asynchronous FL (AFL) has emerged as a significant advancement in federated systems,

particularly in response to the challenges posed by device heterogeneity. This approach allows clients to upload their local model updates at different times rather than synchronously [65]. 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. 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 dynamic 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 [66], 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 [67], 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 and taking into account the highest excitation (I^{max}) and inhibition ($-I^{max}$), the normalization of synaptic weights is performed as follows:

$$\overline{\boldsymbol{W}}_k(t) = \frac{1}{I_k^{max}}[\boldsymbol{W}_k(t)] \tag{4.21}$$

where $\boldsymbol{W}_k(t)$ represents the matrix of synaptic weights, $\overline{\boldsymbol{W}}_k(t)$ denotes the normalized synaptic weight matrix for agent $k \in \{1, 2, 3, ..\}$, 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,

$$\overline{\boldsymbol{W}}_G(t) = \frac{\sum_{k=1}^{N} \omega_k . \overline{\boldsymbol{W}}_k(t)}{\sum_{k=1}^{N} \omega_k} \tag{4.22}$$

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

$$\omega_k = \frac{1}{\sqrt{mn}} \|\overline{\boldsymbol{W}}_k(t)\|_F \exp\left(-\frac{t - T_k}{\tau_{cs}}\right) \quad (t \geq T_k) \tag{4.23}$$

where the term $\|.\|_F$ is the Frobenius norm, and m and n are the dimensions of the matrix $\overline{\boldsymbol{W}}_k(t)$, 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 τ_{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,

$$\mathcal{D}_a(\overline{\boldsymbol{W}}_k(t), \overline{\boldsymbol{W}}_G(T_{cs}) = \frac{1}{2\sqrt{mn}} \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} (a_{ij} - b_{ij})^2} \tag{4.24}$$

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 $\overline{\boldsymbol{W}}_k(t)$ and the model it recently sent to the server $\overline{\boldsymbol{W}}_k(T_k)$ 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},

$$\mathcal{D}_G(\overline{\boldsymbol{W}}_G(t), \overline{\boldsymbol{W}}_G(T_{cs})) = \frac{1}{2\sqrt{mn}} \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}(a_{ij} - b_{ij})^2} \quad (t \geq T_{cs}) \tag{4.25}$$

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

$$\dot{\boldsymbol{W}}_k(t) = (1-\delta(t-T_{cs}))\left[\boldsymbol{\alpha} \odot \mathbf{STDP}(\tau) \odot \mathcal{R}(t) - \boldsymbol{\Theta} \odot \operatorname{sgn}(\boldsymbol{W}_k(t))\right] + \delta(t-T_{cs}) I_k^{max}\left(\overline{\boldsymbol{W}}_G(t) - \overline{\boldsymbol{W}}_k(t)\right) \tag{4.26}$$

where δ is the Dirac delta function. Algorithm 2 shows the step-by-step implementation process of the proposed method.

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

Initialize the agents and Central Server with default parameters for model publication threshold, Euclidean distance, and model publish status
Initialize the Global Model on the Central Server
if t is greater than 0 **then**
    Normalize synaptic weights of local models using (4.21)
    Aggregate models from all agents at the Central Server using (4.22)
    Calculate the $\mathcal{D}_G$ between the current and previous global models on the Central Server
    if $\mathcal{D}_G >$ the Central Server's threshold **then**
        Publish the global model
        set $T_{cs} = t$
    end if
    for each Agent in the network **do**
        if Central Server publishes a new global model **then**
            Update the local model of the Agent with the global model
        else
            Agents evaluate their local models against the latest global model ($\mathcal{D}_a$) using (4.24)
            if $\mathcal{D}_a >$ the Agent's threshold **then**
                Send the model to the Central Server
                set $T_k = t$
            end if
        end if
    end for
end if

The proposed algorithm allows agents to communicate less often and save energy. It does this by only sending essential updates to the Central Server, which helps when there are many agents with different network models and communication interfaces. This method reduces unnecessary data transmission, making the whole system more efficient. It also decides which agent updates are important based on how much they change the global model.

4.3 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. The results from this phase were then compared to those obtained using a Fuzzy Actor-Critic Learning (FACL) algorithm. 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.

4.3.1 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 m/s. The architecture of each agent's neural network included 72 input neurons. 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. The network's output layer comprised 4 neurons, divided equally to represent Δx and Δy movements. The SNN model in the simulation is a fully connected network, and the parameters of the LIF neuron are also represented in Table 4.1.

Table 4.1: Parameter values for LIF neuron model [68]

Parameter	**Value**	**Description**
R_m	40 MΩ	Membrane Resistance
τ_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.

Table 4.2: Simulation Parameters

Parameter	**Value**	**Description**
ΔT	10 ms	Weight and state update sample time
τ_s	2 ms	Time constant for R-STDP
$\mathcal{A}$	1	Amplitude in R-STDP function
λ	5	Decay rate coefficient
Δx and Δy	0.01 m	Max step per ΔT
σ	0.5	Gaussian function's std. deviation
Δt	1 ms	Minimum inter-spike interval
I^{min}	0.5	Lower bound of synaptic weight
I^{max}	15.5	Upper bound of synaptic weight
γ	0.95	Correlation Coefficient

Table 4.2 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 4.6 shows the definition of the distances.

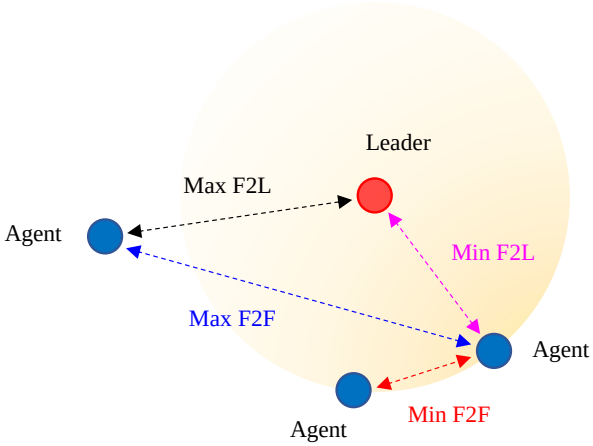

Figure 4.6: The measured distances during the test for evaluating the performance and detecting the collisions.

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's role in the learning process.

To ensure a fair comparison between different learning methods, we have compared our proposed approach with the FACL algorithm [69]. We opted for a fuzzy controller as it provides an explanation for how the SNN works. The FACL algorithm consists of two components, an actor and a critic. The actor is a fuzzy controller whose output serves as the control signal. The critic is a fuzzy inference system that stores the state value and represents the value function. The actor's output, which is the control signal, is computed as follows:

$$u_t = \sum_{l=1}^{L} \Phi^l \omega_t^l, \tag{4.27}$$

where u_t is the control signal at time t, L is the total number of the rules, ω_t^l is the output parameter of rule l at time t, and Φ^l is the firing strength of rule l, which is calculated as

follows,

$$\Phi^l = \frac{\partial u}{\partial \omega_l} = \frac{\prod_{i=1}^{n} \mu^{F_i^l(x_i)}}{\sum_{l=1}^{L} (\prod_{i=1}^{n} \mu^{F_i^l(x_i)})}, \tag{4.28}$$

where $\mu^{F_i^l(x_i)}$ is the membership degree of the ith membership function for rule l. The critic stores an approximation of the value function. In this chapter, we show the critic's output at time t with $\hat{V}_t$, which is as follows,

$$\hat{V}_t = \sum_{l=1}^{L} \Phi^l \zeta_t^l, \tag{4.29}$$

where ζ_t^l is the output parameter of rule l at time t. Finding the proper values for ω^l and ζ^l concerning a reward is the training purpose in the FACL algorithm.

To train the followers, five initial sets of ω^ls and ζ^l are created and set to zero. Each follower generates an action with (4.27). The value of u_t is perturbed via a Gaussian noise with a mean of 0 and variance of ς ($u'_t = u_t + n(0, \varsigma)$). The agents take action u'_t and move to a new state. The reward function returns a signal corresponding to the quality of the actions. In this section, we implemented the reward functions in (4.11) and (4.12), where the weighting coefficients are set to 1. The temporal difference Δ is calculated for each agent as follows,

$$\Delta = \mathcal{R}_t + \Gamma \hat{V}_{t+1} - \hat{V}_t, \tag{4.30}$$

where $\mathcal{R}_t$ is the reward function and is the summation of (4.11) and (4.12). The actor is updated as follows,

$$\omega_{t+1}^l \longleftarrow \omega_t^l + \rho \Delta (u'_t - u_t) \Phi^l. \tag{4.31}$$

The critic is updated as follows,

$$\zeta_{t+1}^{l} \longleftarrow \zeta_{t}^{l} + \kappa \Delta \Phi^{l}. \tag{4.32}$$

In (4.31) and (4.32), κ and ρ are the critic's and actor's learning rates. Table 4.3 shows the parameters used for the FACL algorithm. It should be mentioned that seven membership functions are used for each input, and thus, the total number of rules is $7 \times 7 \times 7 \times 7 \times 7 \times 7 = 117,649$.

Table 4.3: Simulation Parameters for the FACL Algorithm

Parameter	**Value**	**Description**
κ	1.0	Critic's learning rate
ρ	0.5	Actor's learning rate
ς	5	Added exploration noise
Γ	0.0	Discount factor

Figure 4.7 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.

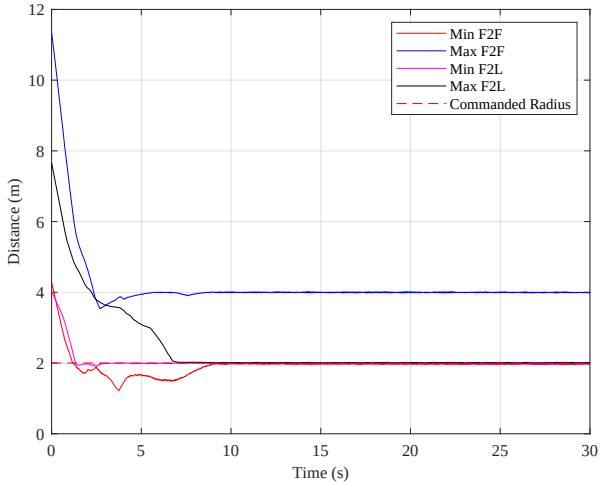

Figure 4.7: Distances during the test phase (RCSE method)

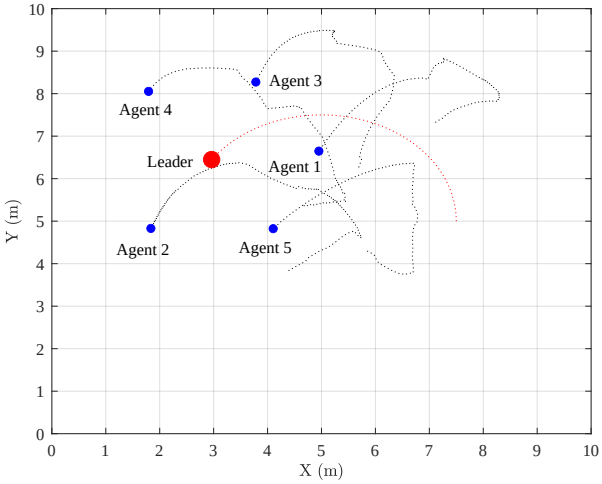

Figure 4.8: Agents' trajectory during the test phase (RCSE method)

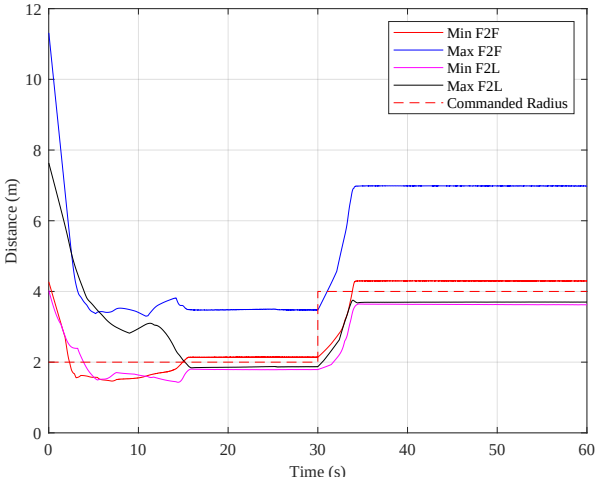

Figure 4.10: Distances during the test phase (FACL method)

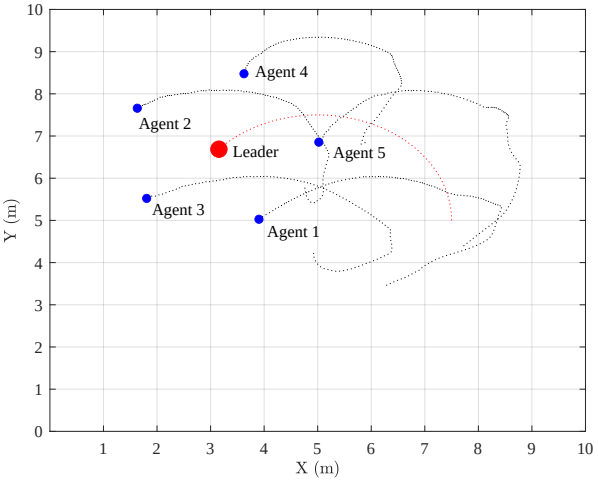

Figure 4.11: Agents' trajectory during the test phase (FACL method)

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 paper 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 4.9 shows the response of the agents to change 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.

Figure 4.10 shows the simulation results for the FACL algorithm. The training time was 1000 seconds, and the algorithm converged to the final solution in 667.26 seconds. According to the results, the agents achieved successful formation in approximately 15.85 seconds. Similarly, Figure 4.12 shows the simulation results for the FACL algorithm, where the commanded distances (r_{cmd} and R_{cmd}) are initially set to 2 m and at $t = 30$, the commanded distance changes to 4 m. The results show that it takes 4.35 seconds for the swarm to adapt to the new command. In Figures 4.10 and 4.12, the distances converge to the commanded distance but stay within a bias. Increasing the number of membership functions can reduce the bias, but it increases the training time of the FACL algorithm.

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 (4.12), is also changed to -1, so the reward function for the leader is changed as follows,

$$\mathcal{R}_{Fi}^L(t) = -\mathcal{C}_{Fi}^L \left[r_{Fi}^L(t-1) - r_{Fi}^L(t) \right] \tag{4.33}$$

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 need to maintain a commanded distance only from each other. Therefore, the commanded distance from the obstacle encoder in the input layer must be removed. Therefore, (4.1) can then be rewritten as follows:

$$\mu_I(\phi_i) = \exp\left(-\frac{(\phi_i - \zeta)^2}{2\sigma^2}\right) \tag{4.34}$$

The simulation proceeded for an additional 1200 seconds, during which the synaptic weights were adjusted in accordance with the new reward function given by (4.33). The results of the reward change are shown in Figure 4.13, which indicates that the agents quickly reduced their initial distance to the commanded distance. 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 4.14 shows the trajectory of each agent after the reward change.

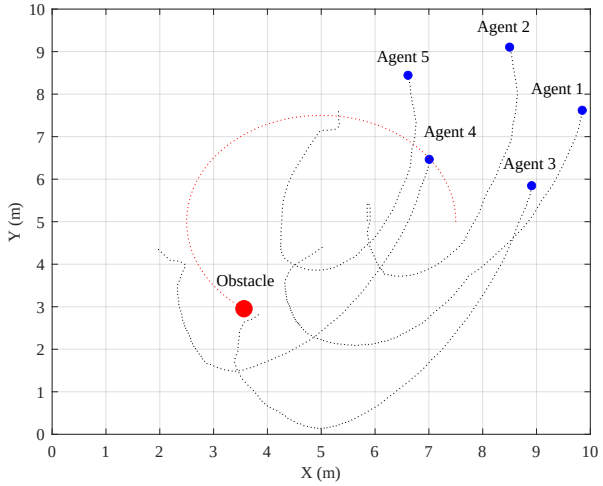

Figure 4.13: Agents' path after reward change (test phase)

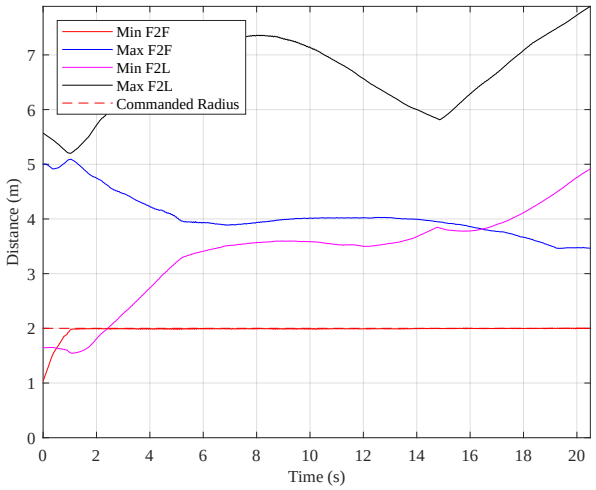

Figure 4.14: Distances after reward change (test phase)

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 4.15 and 4.16, 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 4.2 along with the reward coefficients $\mathcal{C}_{Fi}^{Fj} = 0.02$ and $\mathcal{C}_{L}^{Fj} = 0.07$, and $r_{Fi}^{Fj} = 1\ m/s$ and $r_{Fi}^{L} = 0.1\ m/s$, the maximum rewards at each weight update interval (ΔT) for $\mathcal{R}_{Fi}^{Fj}$ and $\mathcal{R}_{Fi}^{L}$ are calculated as 4×10^{-4} and 7.7×10^{-4}, respectively. Consequently, $\mathcal{R}_{max}^{G} = \max(\mathcal{R}_{Fi}^{Fj}, \mathcal{R}_{Fi}^{L}) = 7.7 \times 10^{-4}$. The Ψ^{S} for the follower section in the network is $\left[\frac{4\times10^{-4}}{7\times10^{-4}}\right] 15.5 = 8.0519$, and for the leader section, it is $\left[\frac{7\times10^{-4}}{7\times10^{-4}}\right] 15.5 = 15.5$. The η and β for the follower section within the network are $\frac{\mathcal{A}\mathcal{R}_{max}^{G} \ln(2^{\lambda^2}-1)}{\lambda\Psi^{\mathcal{S}} \log(2)} = 0.0011$ and $\beta = \frac{\ln(2^{\lambda^2}-1)}{\Psi^{\mathcal{S}}} = 2.152$, respectively. For the leader section, these values are $\frac{\mathcal{A}\mathcal{R}_{max}^{G} \ln(2^{\lambda^2}-1)}{\lambda\Psi^{\mathcal{S}} \log(2)} = 5.719 \times 10^{-4}$ and $\beta = \frac{\ln(2^{\lambda^2}-1)}{\Psi^{\mathcal{S}}} = 1.118$, respectively.

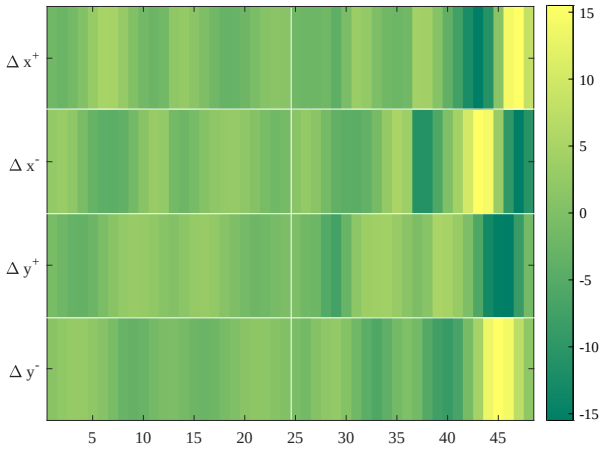

Figure 4.15: Synaptic Weights before Reward change

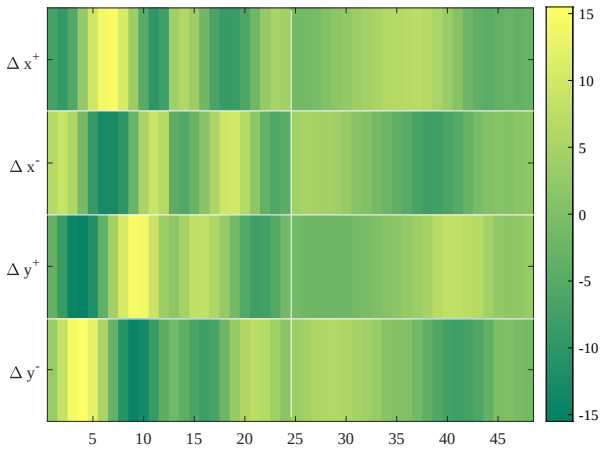

Figure 4.16: Synaptic Weights after Reward change

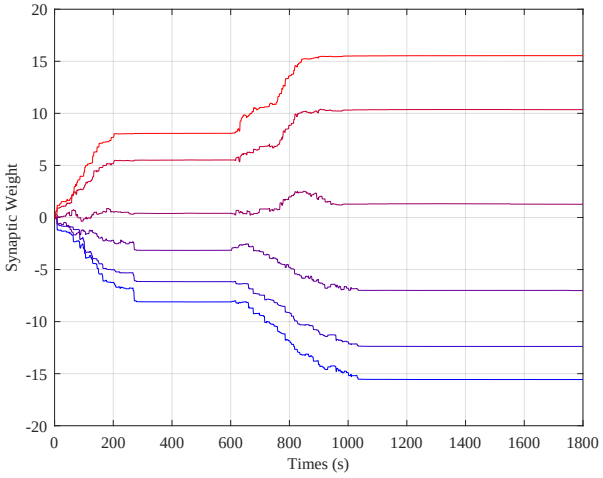

Figure 4.17: Synaptic weights increase after reward change

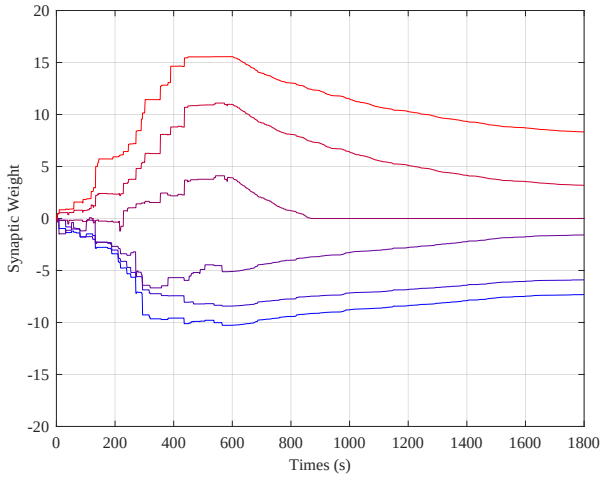

Figure 4.18: Synaptic weights decrease after reward change

According to Figure 4.15, we can see a gradual change in weights because of the fuzzy nature of the input. Since the reward coefficients for followers and leaders are different, there is a difference in the maximum allowed synaptic weight for them. The proposed method for controlling the unbounded growth of synaptic weights has successfully stabilized the network while maintaining the fuzzy patterns of the synaptic connections.

Figure 4.16 shows the synaptic weights after the reward change. In this case, since the reward coefficients are changed, the η and β values 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.

4.3.2 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 a little bit 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 that it waits for the other agents to send their models.

Figure 4.19 shows the distances between agents and the leader before the reward change. According to the figure, the agents converge to the solution faster than the non-federated learning scenario without any error. Figure 4.20 shows the simulation results for the reward change scenario. According to the figure, the proposed aggregation method has improved the learning performance so that the swarm converges to the solution in 6 seconds.

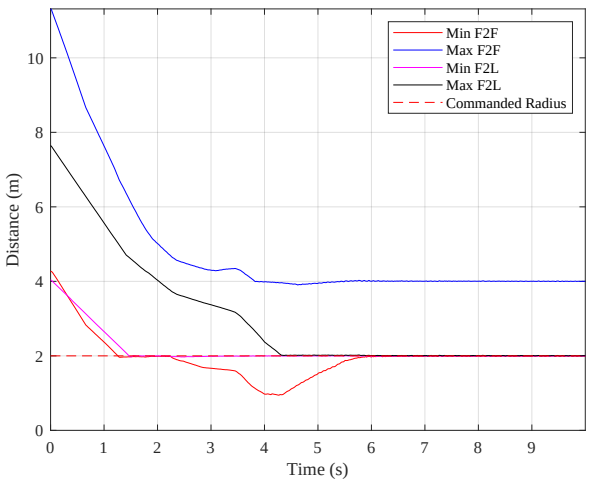

Figure 4.19: Distances during the test phase before reward change in the proposed aggregation method.

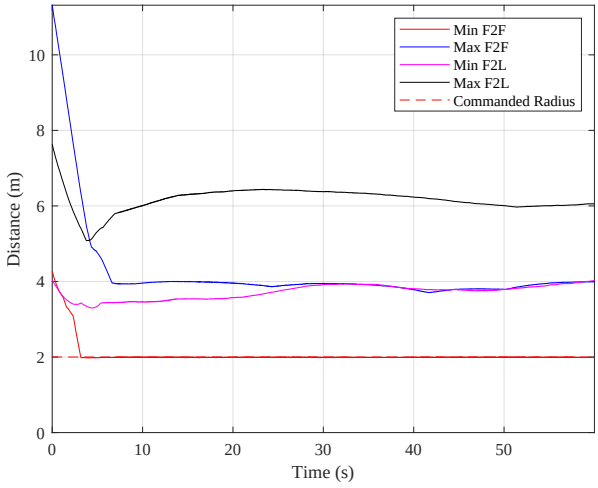

Figure 4.20: Distances during the test phase after reward change in the proposed aggregation method.

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 as an index to adjust the learning process in SNNs. The proposed aggregation method helps SNN models

converge into a single model. Agents communicate with the Central Server (leader) during the training process, and Figure 4.22 shows the communication times. 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 norm of the synaptic weights determines the principle of the aggregation process, which results in small intervals of aggregation when the change rate is high, and larger intervals when it reduces.

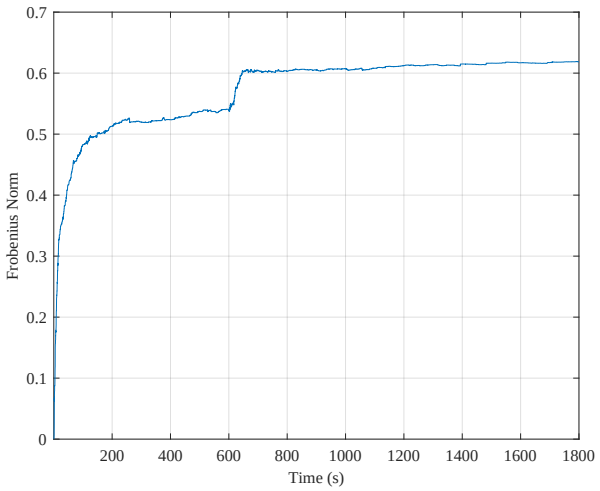

Figure 4.21: Frobenius norm of the agents during the learning phase. The reward changes for the Leader after 600 s.

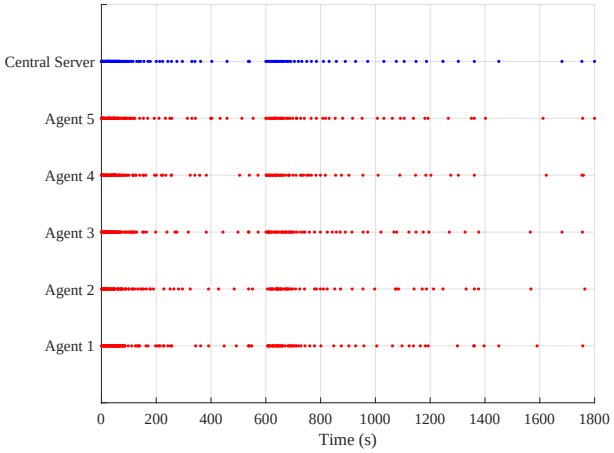

Figure 4.22: Communication times for agents and the Central Server (Leader). Red and blue dots show the times that agents and the Central Server have sent their model, respectively.

Therefore, the aggregation frequency is very high initially, and it reduces after the change in synaptic weights goes to zero because of the learning rate in (4.13). 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 4.4: Comparative Performance Analysis of the Proposed Aggregation Algorithm+RCSE, RCSE, and FACL

	FL+RCSE	**RCSE**	**FACL**
Learning Time - Training Phase (s)	261.92	554.71	667.26
Max Error After Convergence - Test Phase (%)	0.71	1.95	9.85
Convergence Time - Test Phase (s)	5.81	8.94	15.82

Table 4.4 compares the proposed Federated Learning (FL) algorithm with the non-FL approach and FACL algorithm. The comparison focuses on three critical metrics: learning time, maximum error after convergence, and convergence time. The proposed FL algorithm demonstrates 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.

4.4 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, significantly reducing learning time and 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 model's dynamic nature of the model aggregation time adjusts 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 and performs better than non-federated learning and FACL cases. Moreover, the dynamic aggregation interval effectively reduces communication overhead between the agents and the central server, particularly after model convergence. This reduction is critical in scenarios where communication bandwidth is limited or costly. Our findings suggest that integrating R-STDP with federated learning, supported by our dynamic aggregation approach, provides a robust framework for advancing multi-agent reinforcement learning systems.

Chapter 5

Proposed Method

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] 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.

[14] E. M. Izhikevich and R. FitzHugh, "Fitzhugh-nagumo model," *Scholarpedia*, vol. 1, no. 9, 2006.

[15] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on neural networks*, vol. 14, no. 6, pp. 1569–1572, 2003.

[16] D. O. Hebb, *The organization of behavior: A neuropsychological theory*. Psychology press, 2005.

[17] 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.

[18] 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.

[19] E. Oja, "Simplified neuron model as a principal component analyzer," *Journal of mathematical biology*, vol. 15, pp. 267–273, 1982.

[20] A. Soltoggio and J. J. Steil, "Solving the distal reward problem with rare correlations," *Neural computation*, vol. 25, no. 4, pp. 940–978, 2013.

[21] H. S. Seung, "Learning in spiking neural networks by reinforcement of stochastic synaptic transmission," *Neuron*, vol. 40, no. 6, pp. 1063–1073, 2003.

[22] E. M. Izhikevich, "Solving the distal reward problem through linkage of stdp and dopamine signaling," *Cerebral cortex*, vol. 17, pp. 2443–2452, 2007.

[23] A. Soltoggio, "Short-term plasticity as cause–effect hypothesis testing in distal reward learning," *Biological cybernetics*, vol. 109, no. 1, pp. 75–94, 2015.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[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, p. 13, 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] F. Ponulak and J. J. Hopfield, "Rapid, parallel path planning by propagating wavefronts of spiking neural activity," *Frontiers in computational neuroscience*, vol. 7, p. 98, 2013.

[32] 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.

[33] H. Markram, W. Gerstner, and P. J. Sjöström, "A history of spike-timing-dependent plasticity," *Frontiers in synaptic neuroscience*, vol. 3, p. 4, 2011.

[34] 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, p. 19, 2010.

[35] E. Nordlie, T. Tetzlaff, and G. T. Einevoll, "Rate dynamics of leaky integrate-and-fire neurons with strong synapses," *Frontiers in computational neuroscience*, vol. 4, p. 149, 2010.

[36] E. M. Izhikevich, "Solving the distal reward problem through linkage of stdp and dopamine signaling," *Cerebral Cortex*, vol. 17, no. 10, pp. 2443–2452, 2007.

[37] 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.

[38] T. DeWolf, T. C. Stewart, J.-J. Slotine, and C. Eliasmith, "A spiking neural model of adaptive arm control," *Proceedings of the Royal Society B: Biological Sciences*, vol. 283, no. 1843, p. 20162134, 2016.

[39] J. Pérez, J. A. Cabrera, J. J. Castillo, and J. M. Velasco, "Bio-inspired spiking neural network for nonlinear systems control," *Neural Networks*, vol. 104, pp. 15–25, 2018.

[40] C. Teeter, R. Iyer, V. Menon, N. Gouwens, D. Feng, J. Berg, A. Szafer, N. Cain, H. Zeng, M. Hawrylycz, *et al.*, "Generalized leaky integrate-and-fire models classify multiple neuron types," *Nature communications*, vol. 9, no. 1, p. 709, 2018.

[41] M. Mozafari, S. R. Kheradpisheh, T. Masquelier, A. Nowzari-Dalini, and M. Ganjtabesh, "First-spike-based visual categorization using reward-modulated stdp," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 12, pp. 6178–6190, 2018.

[42] 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.

[43] S. Koul and T. K. Horiuchi, "Waypoint path planning with synaptic-dependent spike latency," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 4, pp. 1544–1557, 2019.

[44] A. Tavanaei and A. Maida, "Bp-stdp: Approximating backpropagation using spike timing dependent plasticity," *Neurocomputing*, vol. 330, pp. 39–47, 2019.

[45] 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.

[46] C. Hur, B. Ibrokhimov, and S. Kang, "N3-cpl: Neuroplasticity-based neuromorphic network cell proliferation learning," *Neurocomputing*, vol. 411, pp. 193–205, 2020.

[47] 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.

[48] 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, p. 104362, 2021.

[49] 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.

[50] 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.

[51] 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.

[52] 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.

[53] S. Schmidgall, J. Ashkanazy, W. Lawson, and J. Hays, "Spikepropamine: Differentiable plasticity in spiking neural networks," *Frontiers in neurorobotics*, vol. 15, p. 629210, 2021.

[54] 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, p. 627620, 2021.

[55] J. Shen, Y. Zhao, J. K. Liu, and Y. Wang, "Hybridsnn: Combining bio-machine strengths by boosting adaptive spiking neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 9, pp. 5841–5855, 2021.

[56] S. F. Chevtchenko and T. B. Ludermir, "Combining stdp and binary networks for reinforcement learning from images and sparse rewards," *Neural Networks*, vol. 144, pp. 496–506, 2021.

[57] S. Gupta, G. Singal, D. Garg, and S. Jagannathan, "Qc-sane: Robust control in drl using quantile critic with spiking actor and normalized ensemble," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 9, pp. 6656–6662, 2021.

[58] 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.

[59] 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.

[60] E. Garcia, "Cooperative target protection from a superior attacker," *Automatica*, vol. 131, 2021.

[61] 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.

[62] A. D. Bird, P. Jedlicka, and H. Cuntz, "Dendritic normalisation improves learning in sparsely connected artificial neural networks," *PLOS Computational Biology*, vol. 17, 2021.

[63] 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.

[64] 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.

[65] 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.

[66] 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.

[67] 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.

[68] A. AbdelAty, M. Fouda, and A. Eltawil, "On numerical approximations of fractional-order spiking neuron models," *Communications in Nonlinear Science and Numerical Simulation*, vol. 105, 2022.

[69] L. Jouffe, "Actor-critic learning based on fuzzy inference system," in *IEEE International Conference on Systems, Man and Cybernetics. Information Intelligence and Systems (Cat. No. 96CH35929)*, vol. 1, pp. 339–344, IEEE, 1996.