

Adaptive Hierarchical Spiking Neural Networks for Event-Driven Multi-Agent Cooperation on the Edge

by

Mohammad Tayefe Ramezanlou

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Electrical Engineering
Ottawa-Carleton Institute of Electrical and Computer Engineering Department of
Systems and Computer Engineering

Carleton University

Ottawa, Ontario

Fall, 2025

Copyright ©

2025, Mohammad Tayefe Ramezanlou

Abstract

This thesis presents a comprehensive study of multi-agent reinforcement learning using Spiking Neural Networks (SNNs) for decentralized control in edge computing environments. Chapter 1 introduces the motivation and provides a literature review, highlighting the need for biologically inspired neural models to achieve robust and energy-efficient learning in dynamic multi-agent systems.

Chapter 2 develops adaptive mechanisms for SNNs, focusing on stabilizing learning under sparse conditions for optimal problems and balancing synaptic plasticity with network equilibrium.

Chapter 3 explores federated learning as a lightweight strategy for sharing matured SNN policies among agents, aiming to reduce communication overhead while maintaining decentralized autonomy. This chapter demonstrates that event-triggered policy exchanges can improve efficiency without requiring full synchronization or centralized orchestration.

Chapter 4 presents the main simulation results, introducing a vision-based multi-agent docking framework that combines Dynamic Vision Sensor (DVS) event streams, adaptive light modulation, and deep SNN controllers. The proposed system demonstrates robust cooperative docking, precise formation control, and resilience to actuation constraints, validating the effectiveness of neuromorphic approaches for complex multi-agent coordination tasks.

Contents

1	Introduction	1
1.1	Spiking Neural Networks: Models and Learning Algorithms	3
1.1.1	Neuron model	3
1.1.2	Izhikevich model	6
1.1.3	Learning Approaches in SNNs	7
1.2	Literature Review	9
2	Modular Learning in SNNs for Optimal Multi-Agent Decision-Making	14
2.1	Introduction	14
2.2	The ATD problem and SNN-based solution	15
2.3	Learning using R-STDP	15
2.4	Network structure and encoding method	21
2.5	Results	27
2.5.1	Simulation without noise	31
2.5.2	Simulation with noise	32
2.6	Conclusion	34
3	Integration of R-STDP and Federated Learning	35
3.1	Consensus Flying Problem	35
3.2	Proposed Method	37
3.2.1	Network Structure	37

3.2.2	Training algorithm	42
3.2.3	Weight Stabilization using Reward-Modulated Competitive Synaptic Equilibrium (RCSE)	43
3.2.4	Federated Learning for Consensus Flying	54
3.3	Results and Discussion	60
3.3.1	Simulation without Federated Learning (FL)	60
3.3.2	Simulation with Federated Learning and Reward-modulated Competitive Synaptic Equilibrium	70
3.4	Conclusion	73
4	Vision-Based Multi-Agent Docking with Deep Spiking Neural Networks	75
4.1	Preliminaries	77
4.1.1	Problem definition: Docking Missions with Swarm Robots	77
4.1.2	Event-Based Camera Simulation	81
4.1.3	Dynamic Model of the Payload and Agents	84
4.1.4	Neuron model	87
4.2	Proposed Neuromorphic Framework	87
4.2.1	Deep Spiking Neural Network (SNN) Structure	87
4.2.2	Training Deep Spiking Neural Network (SNN) with Reward-modulated Spike-Timing-Dependent Plasticity (R-STDP)	99
4.3	Results	132
4.3.1	Multi-Agent Docking Task Simulation Setup	132
4.4	Conclusion	152
5	Discussion and Conclusion	154

List of Figures

1.1	Simple model of a Spiking Neural Network (SNN). The spike pattern shows that the neurons spike whenever the voltage of the neuron reaches a threshold.	3
2.1	Active target defense game with three agents (LOS angles are shown for both agents).	16
2.2	Two Spiking Neural Network (SNN)s simultaneously play and control the invader and defender. Each agent uses the LOS angle and relative velocities for training.	16
2.3	An illustration of input current to Spiking Neural Network (SNN) and synaptic current to the output neurons after training for a single connection.	17
2.4	Relative velocities and LOS angles used in the reward function	20
2.5	Network structure and encoding process for the input layer (Defender). Each neuron is associated with a membership function in Gaussian Receptive Fields (GRF). The Gaussian Receptive Fields (GRF) encodes an input State (S^t) at each time step. There is both a training phase when ($\xi = 0$) and an operating phase when ($\xi = 1$). The “F2S Conversion” block converts fuzzy membership values to currents for the neurons according to (2.12).	23
2.6	Reward, eligibility trace, and weight change during the simulation for W^{3-3} . The $C(t)R(t)$ changes the synaptic weight by considering activation strength and reward value.	24

2.7	Spiking Neural Network (SNN)’s performance during training. Hollow circles show the initial positions.	28
2.8	Changes in synaptic weights during training. Only Maximum synaptic weights for both agents are shown.	29
2.9	Spiking Neural Network (SNN)’s performance after training. The highlighted region shows the defender’s dominant region. The purple dot is the optimal capture point. The CO Type-E shows the reachable regions of the Invader and the Defender.	31
2.10	Spiking Neural Network (SNN)’s performance in noisy conditions. A white Gaussian noise with a variance of 0.01 is added to the measured inputs.	33
3.1	The central server (the leader) and the surrounding follower agents (white drones). The follower agents learn to fly in a formation to maintain the commanded distance. The Local models trained individually by follower agents are sent to the leader. The leader aggregates the models and sends back the global model for another round of training of the follower agents.	36
3.2	Spiking Neural Network (SNN) structure with encoding and decoding layers. Each input sub-layer consists of a fuzzy encoder followed by the fuzzy-to-spiking current conversion defined in (3.2). During the training phase, the output layer receives input only from the random action selector, which is replaced by synaptic weight inputs during the testing phase.	39
3.3	The fuzzy encoding principle for the input sub-layer.	40
3.4	Synaptic weight change for $\lambda = 5$, $\Psi^S = 15.5$, and $\mathcal{AR}_{max}^G = 1$	52
3.5	Reward-based learning rate and decay rate functions. In the blue region (active learning rate), the reward adjusts the weights, and in the red region (active decay rate), the RCSE method controls synaptic growth.	52

3.6 RCSE working principle in inhibiting the adjacent synaptic connections. The heatmap shows the synaptic weight matrix. Neurons have different firing strengths due to the difference in fuzzy membership values, which affects the increase or decrease rate and shapes the patterns in the synaptic weight matrix.	53
3.7 Measured distances used for evaluating swarm flight performance and collision detection.	63
3.8 Agents' trajectory during the test phase	63
3.9 Variation of distances within the swarm during the test phase	64
3.10 Adaptive response to commanded distance adjustments - reconfiguration during the test phase	64
3.11 Trajectory adaptations of following agents in response to reward change for the leader during the test phase.	66
3.12 Variations in distances after reward changes and Leader becomes Obstacle - test phase	66
3.13 Synaptic Weights before Reward change in Reward-Modulated Competitive Synaptic Equilibrium (RCSE) method.	68
3.14 Synaptic Weights after Reward change in Reward-Modulated Competitive Synaptic Equilibrium (RCSE) method.	68
3.15 Synaptic Weights before Reward change in Multiplicative Synaptic Normalization method.	68
3.16 Synaptic Weights after Reward change in Multiplicative Synaptic Normalization method.	68
3.17 Synaptic weights increase after reward change in Reward-Modulated Competitive Synaptic Equilibrium (RCSE) method.	70
3.18 Synaptic weights decrease after reward change in Reward-Modulated Competitive Synaptic Equilibrium (RCSE) method.	70
3.19 Distances during the test phase before reward change in the proposed event-triggered Federated Learning (FL) method.	71

3.20	Distances during the test phase after reward change in the proposed event-triggered Federated Learning (FL) method.	71
3.21	Frobenius norm of the Agent 1's weighs during the learning phase. The reward changes for the Leader after 600 s.	72
3.22	Communication times for agents and the Central Server (Leader). Red and blue dots show the times that agents and the Central Server have sent their model, respectively.	72
4.1	3D schematic of the multi-agent docking mission environment. Multiple agents work collaboratively to push a central payload towards a Soft Capture System (SCS) for docking alignment.	79
4.2	2D representation of robots performing docking mission. Each agent pushes the payload towards the Soft Capture System (SCS) to align them together. The Soft Capture System (SCS) and agents are equipped with a Dynamic Vision Sensor (DVS) that streams high-frequency data for the proximity mission.	80
4.3	Deep Spiking Neural Network (SNN) architecture with entropy-based adaptive pooling, multi-repository hidden layers, and biologically-inspired inhibitory and excitatory synapses. Purple, red, and blue arrows denote excitatory and yellow arrows denote inhibitory pathways.	88
4.4	Structured First Hidden Layer Architecture with payload, and Agent Perception Repositories for Visual Perception.	92
4.5	Structured Second Hidden Layer architecture showing the Rendezvous Phase Repository, Docking Phase Repository, Rendezvous Phase Inhibitor, and Docking Phase Inhibitor repositories interconnected through inhibitory (yellow) and excitatory (purple) synapses. External stimulation from contact sensors triggers mission-phase transitions.	95
4.6	Output Layer Architecture with Lateral Inhibition between Opposing Direction Neurons to Enforce Winner-Take-All Dynamics within Each Control Axis.	97
4.7	Computation of the activity matrix (A) from the time-averaged spiking activity.	100

4.8 Input layer activity ($\mathbf{A}^{\text{input}}$), Payload Attention Activity ($\mathbf{A}_{\text{pay}}^{\text{H1}}$), and Neighboring Agent Attention Repository ($\mathbf{A}_{\text{agt}}^{\text{H1}}$) computation pipeline when the agent's Active LED Marker (ALM) is turned off and most of the events are generated by the payload because of its larger weight in comparison with the agent. This feature puts more attention on the payload position.	102
4.9 Computation of the Attention Mechanism for Reward Generation in the First Hidden Layer. The attention map ($\mathcal{A}^{\text{Attention}}$) is computed by comparing the input layer activity ($\mathbf{A}^{\text{input}}$) with the spatial positions of the first hidden layer neurons using a Gaussian-weighted mapping. The resulting attention map is then used to calculate the reward signal (r^{H1}) for each neuron in the first hidden layer by comparing it with the hidden layer activity (\mathbf{A}^{H1}).	103
4.10 Reward function pipeline for the payload. The pipeline illustrates the raw Dynamic Vision Sensor (DVS) events, the attention-based map, the normalized hidden layer activity during the initial training phase, and the reward signal calculated as the difference between the attention map and the hidden layer activity ($r^{\text{H1}}(q)$).	104
4.11 Reward function pipeline for the agents. This pipeline demonstrates the raw Dynamic Vision Sensor (DVS) events for multiple agents, the attention map, the hidden layer's normalized activity, and the modulation signal computed for reward.	105
4.12 Numerical example setup for Gaussian attention computation in the first hidden layer. (a) shows the input layer activity matrix representing time-averaged firing rates from the input layer (4.39). (b) illustrates the spatial arrangement of hidden-layer neurons, each serving as a query point ($q(2, 2)$) for attention calculation.	107

4.13 Attention weight computation for hidden neuron $q = (2, 2)$ at position $(x_q, y_q) = (-0.333, -0.333)$. (a) shows the Gaussian weight map γ_q centered at the hidden neuron's location. (b) illustrates the element-wise product of the weight map with the input layer activity matrix, highlighting the contributions to the attention calculation.	110
4.14 The DVS event image is masked by the upsampled binary detection mask derived from the first hidden layer's output, isolating events corresponding to the active object (payload or agent).	116
4.15 Oscillatory behavior of the Gaussian envelope used for computing the motion direction signal \mathcal{H} in the second hidden layer. The amplitude alternates between positive and negative values, shifting focus between the payload and agents based on the active Active LED Marker (ALM).	118
4.16 Payload motion direction detection using a 2D Gaussian reward function with higher standard deviation.	118
4.17 Neighboring agent motion direction detection using a 2D Gaussian reward function with lower standard deviation.	119
4.18 The oscillatory Gaussian envelope used to compute the motion direction signal \mathcal{H} for reward modulation in the second hidden layer. The center can be Soft Capture System (SCS) or the agent's Dynamic Vision Sensor (DVS) center..	120
4.19 Velocity-aligned activity template \mathbf{M} (left) and opposite-direction template $\widetilde{\mathbf{M}}$ (right) for an agent moving with heading $\varphi = \pi/4$ and normalized speed $\bar{\mathcal{V}} = 0.6$	122
4.20 Output layer reward computation based on directional error between commanded and executed motion.	129

4.21 Schematic of the per-layer R-STDP weight update. The eligibility trace is calculated based on the firing rate of the neurons in layer ℓ and $\ell+1$. The per-neuron reward (r_q) is computed based on the layer-specific reward functions. The weight update is obtained by multiplying the eligibility trace by the per-neuron reward.	131
4.22 High-level architecture of the spiking neural network for multi-agent docking. The network processes DVS input through attention and phase repositories to generate thrust commands for docking maneuvers.	134
4.23 Effect of the Active LED Marker (ALM) on visual perception. The ALM significantly enhances the visibility of proximal agents in the DVS input, improving the network's ability to detect and differentiate them from the payload.	136
4.24 Attention maps illustrating the effect of the Active LED Marker (ALM) on visual focus. The Active LED Marker (ALM) enables dynamic shifting of attention from the payload to the proximal agent by generating distinct DVS event patterns. Colorbar values indicate the normalized attention strength, where brighter regions correspond to locations receiving stronger attention. .	137
4.25 SNN structure in the first hidden layer under externally controlled attention signaling: (a) agents' Active LED Marker (ALM) disabled, which corresponds to the absence of ALM illumination and results in payload-focused visual processing; (b) agents' Active LED Marker (ALM) enabled, which corresponds to active ALM illumination and emphasizes proximal agent detection. Inactive neuron repositories and synaptic connections are depicted in gray, while dotted lines represent inactive inhibitory pathways.	138
4.26 Detector repository activity during initial training with Active LED Marker (ALM) off. The payload detector remains active, while the proximal agent detector is suppressed due to inhibitory signals from the payload attention Inhibitor.	139

4.27 Detector repository activity during initial training with Active LED Marker (ALM) on. The proximal agent detector becomes active, while the payload detector is suppressed by ALM-induced inhibition.	140
4.28 Agent detector repository activity after learning convergence with Active LED Marker (ALM) off. The payload detector shows strong, localized activity, while the proximal agent detector remains inactive.	141
4.29 Agent detector repository activity after learning convergence with Active LED Marker (ALM) on. The proximal agent detector shows strong, localized activity, while the payload detector remains inactive.	142
4.30 Second hidden layer activity for the rendezvous and docking phases for Agent 1: (a) Rendezvous Phase Repository (RPR) activity 135 degree that matches the LOS of the payload regarding the Agent 1 and (b) Docking Phase Repository (DPR) activity shows 45 degree LOS toward the docking point.	143
4.31 Hidden layer 2 and output layer activity during the rendezvous phase.	144
4.32 Hidden layer 2 and output layer activity during the docking phase.	145
4.33 Distances during time.	146
4.34 Reward profile during the docking phase	147
4.35 Views during the rendezvous phase at different time steps. In the Dynamic Vision Sensor (DVS) views, yellow regions represent the direction of motion as determined by event polarities, while white lines in (a) and (b) depict the trajectories of the payload and agents.	148
4.36 Views during the docking phase at different time steps. In the Dynamic Vision Sensor (DVS) views, yellow regions represent the direction of motion as determined by event polarities, while white lines in (a) and (b) depict the trajectories of the payload and agents.	149

List of Tables

1.1	Spike Response Function of the neurons in an Spiking Neural Network (SNN).	4
2.1	Parameter values for LIF neuron model [86]	27
2.2	Parameter values for Reward-modulated Spike-Timing-Dependent Plasticity (R-STDP)	28
3.1	Parameter values for Leaky Integrate-and-Fire (LIF) neuron model [86] . . .	61
3.2	Simulation Parameters	62
3.3	Comparative Performance Analysis of the Proposed Aggregation Algorithm+Reward- Modulated Competitive Synaptic Equilibrium (RCSE) and Reward-Modulated Competitive Synaptic Equilibrium (RCSE)	73
4.1	Physical parameters of the payload and agents used in the docking simulations.	132
4.2	Neuron layer configurations with full descriptions of each functional subgroup.	133
4.4	Layer-wise Learning Parameters	134

Acronyms

ALM Active LED Marker

DPI Docking Phase Inhibitor

DPR Docking Phase Repository

DVS Dynamic Vision Sensor

FL Federated Learning

GRF Gaussian Receptive Fields

LIF Leaky Integrate-and-Fire

MIB Minimum-Impulse Bit

NAAI Neighboring Agent Attention Inhibitor

NAAR Neighboring Agent Attention Repository

PAI Payload Attention Inhibitor

PAR Payload Attention Repository

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

RCSE Reward-Modulated Competitive Synaptic Equilibrium

RPI Rendezvous Phase Inhibitor

RPR Rendezvous Phase Repository

SCS Soft Capture System

SNN Spiking Neural Network

Chapter 1

Introduction

Spiking Neural Network (SNN)s represent the third generation of neural models, operating with discrete spikes that carry both temporal and spatial information. Unlike conventional neural networks that rely on continuous activations, SNNs process asynchronous event streams, providing rich temporal dynamics and exceptionally low power consumption. These characteristics make SNNs highly suitable for edge intelligence, real-time robotics, and distributed autonomous systems, where energy efficiency and reaction speed are critical [1,2].

A major advantage of SNNs lies in their biologically grounded learning mechanisms. Local plasticity rules such as Reward-modulated Spike-Timing-Dependent Plasticity (R-STDP) and reward-modulated STDP enable online adaptation without requiring global error signals. These mechanisms allow SNNs to learn from sparse, delayed, or noisy feedback—conditions commonly encountered in multiagent control. Recent progress in deep SNN architectures extends these ideas by stacking multiple spiking layers, enabling hierarchical feature extraction from high-dimensional sensory streams. However, training deep SNNs remains challenging due to non-differentiability of spikes, temporal credit assignment, and instability of multi-layer plasticity.

To address these challenges, this thesis develops adaptive hierarchical learning algorithms for deep SNNs, combining structured modularity with reward-based plasticity. The proposed

models integrate both local R-STDP mechanisms and temporal population coding, allowing deeper layers to form stable representations while earlier layers handle fast, event-driven signals such as those from Dynamic Vision Sensor (DVS). These contributions enable SNNs to learn complex perception–action loops, maintain short-term temporal memory, and operate robustly in dynamic multiagent environments.

In multiagent systems, one of the main strategies for sharing knowledge across distributed agents is Federated Learning (FL). The FL is a decentralized training framework where agents exchange model updates instead of raw data. In this thesis, FL is integrated with spiking neural networks to enable collaborative learning across multiple event-driven agents, allowing each deep SNN to adapt locally via R-STDP while periodically contributing its learned synaptic structure to a global model without violating bandwidth or energy constraints. FL enables distributed agents to train collaboratively without sharing raw sensory data, preserving privacy and reducing communication loads—qualities particularly relevant for energy-constrained robotic platforms. When combined with the sparse, event-driven nature of SNNs, FL can reduce bandwidth requirements and support decentralized learning across heterogeneous devices [3, 4].

Developing effective training methods for deep spiking neural networks remains an open research challenge, largely due to the non-differentiable nature of spikes, the temporal dependencies across layers, and the difficulty of assigning credit in multi-stage processing pipelines. Existing approaches struggle to balance biological plausibility, computational efficiency, and scalability to deeper architectures. In this thesis, a hierarchical SNN framework is explored in which layered processing, reward-modulated plasticity, and attention-guided reinforcement work together to address these limitations.

1.1 Spiking Neural Networks: Models and Learning Algorithms

1.1.1 Neuron model

In the study of computational neuroscience and the development of neural networks, particularly SNN, various neuron models have been proposed to simulate the electrical activity of neurons. These models range from simple to complex, aiming to capture the essential features of neuronal dynamics. The diagram in Figure 1.1 illustrates a basic network consisting of spiking neurons. In this network, the pre-synaptic neuron (pre-neuron) in the input layer spikes in response to the input (I) and transmits these spikes through synaptic weights to the post-synaptic neuron (post-neuron). Whenever the pre-neuron spikes, the post-neuron receives input current with a value of W .

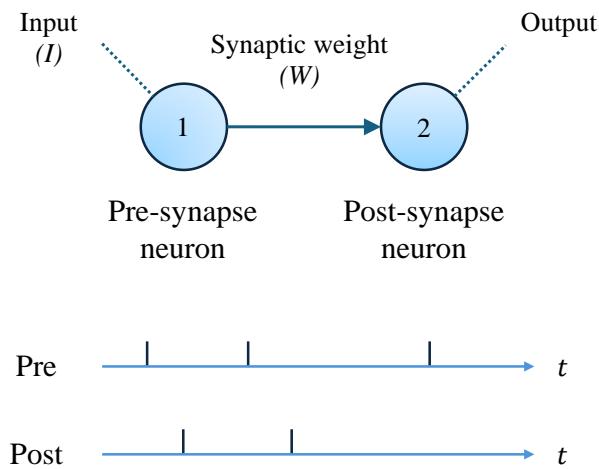


Figure 1.1: Simple model of a SNN. The spike pattern shows that the neurons spike whenever the voltage of the neuron reaches a threshold.

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

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

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

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

Leaky-Integrate and Fire model

The Leaky Integrate-and-Fire (LIF) model is a biological model that can be represented as a circuit with a resistor and capacitor and represents a first-order dynamic system [5],

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

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

The spike rate is a parameter that determines how fast the neuron spikes [6].

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

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

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

where 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} ,

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

or

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

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

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

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

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

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

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

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

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

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

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

1.1.2 Izhikevich model

The Izhikevich model is a widely used spiking neuron model that balances biological plausibility with computational efficiency. It captures a variety of neuronal firing patterns observed in real neurons while remaining relatively simple to implement. The model is defined by the following set of differential equations [7]:

$$\frac{dV}{dt} = 0.04V^2 + 5V + 140 - U + I \quad (1.10)$$

$$\frac{dU}{dt} = a(bV - U) \quad (1.11)$$

where V represents the membrane potential of the neuron, U is a recovery variable that accounts for the activation of potassium ionic currents and inactivation of sodium ionic currents, and I is the input current. The parameters a , b , c , and d are dimensionless constants that can be adjusted to model different types of neurons. The model also includes a reset mechanism that occurs when the membrane potential V reaches a certain threshold (typically 30 mV):

$$\text{if } V \geq 30 \text{ mV, then } \begin{cases} V \leftarrow c \\ U \leftarrow U + d \end{cases} \quad (1.12)$$

The Izhikevich model is capable of reproducing a wide range of neuronal behaviors, including regular spiking, bistability spiking, bursting, and chattering, by appropriately tuning the parameters a , b , c , and d . This versatility makes it a popular choice for simulating large-scale neural networks while maintaining a reasonable level of biological realism.

1.1.3 Learning Approaches in SNNs

Hebbian Learning

Hebbian Learning is a fundamental neural learning principle summarized by the axiom “neurons that fire together, wire together,” describing how simultaneous activation of neurons leads to strengthened connections between them [8]. Hebbian learning, particularly within the context of SNN, primarily revolves around the modulation of synaptic strengths based on the firing rates of pre- and postsynaptic neurons. The principles of locality and joint activity are fundamental, emphasizing that synaptic changes occur only when both neurons are active simultaneously.

Methods of Synaptic Modification:

- **Local Rules:** Synaptic changes are influenced directly by the activities of the connecting neurons without external influences.
- **Bounded Growth:** To avoid uncontrolled increases in synaptic strength, models typically incorporate mechanisms such as hard and soft bounds. Hard bounds prevent any further increase once a maximum weight is achieved, while soft bounds slow down the rate of increase as the maximum is approached [9].
- **Synaptic Decay:** Realistic models also consider mechanisms for reducing synaptic strengths, typically through a decay term that weakens connections in the absence of activity [10].

Advanced Hebbian Models:

- **Covariance Rule:** This model refines the synaptic modification to depend on the deviation of firing rates from their means, enhancing the dynamic response of synapses to changes in neural activity [11].
- **Oja's Rule:** A self-stabilizing rule that ensures synaptic weights do not grow indefinitely by normalizing the weight vector, thereby maintaining the overall stability of the network [12].

- **BCM Rule:** The Bienenstock-Cooper-Munro rule introduces an adaptive threshold for synaptic modification, which evolves based on the historical activity of the neuron, allowing for more refined potentiation and depression based on relative activity levels. [13].
- **RCHP:** Rarely Correlating Hebbian Plasticity focuses on synaptic changes driven by rare, significant coincidences in neuronal activity, aiming to strengthen connections that are crucial for neural function while avoiding over-strengthening due to common activity patterns [14].

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

Neo-Hebbian Learning and Modulation Mechanisms

In neo-Hebbian reinforcement learning, significant advancements come from a global reward signal modulating synaptic plasticity alongside an eligibility trace that decays over time [15]. This trace increases with recent, successful neurotransmission and decreases as time passes, linking closely with Temporal-Difference (TD) learning mechanisms. Such dynamics allow for Long-Term Potentiation (LTP) or Depression (LTD) at synapses based on the timing of synaptic activity and the nature of the reward signal. Specifically, recent successful activities followed by positive rewards enhance LTP, while activities preceding negative rewards lead to LTD.

Distal rewards and credit assignment

This model introduces a sophisticated approach to synaptic modification through the interaction of R-STDP with a modulatory reward signal, 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 horizons. 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 [16]:

$$\frac{dC_{ji}}{dt} = -\frac{C_{ji}}{\tau_C} + \text{STDP}(t_{\text{post}} - t_{\text{pre}})\delta(t - t^{(f)}) \quad (1.13)$$

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

The STDP function can be represented as follows,

$$\text{STDP}(\tau) = \mathcal{A} \exp\left(-\frac{\tau}{\tau_s}\right) \text{ for } \tau \geq 0, (\tau = t_{\text{post}} - t_{\text{pre}}) \quad (1.14)$$

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

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

$$\frac{dw_{ji}}{dt} = C_{ji}(t) \cdot R(t) \quad (1.15)$$

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

1.2 Literature Review

SNNs have attracted increasing attention due to their ability to model neural dynamics using discrete spike events, closely reflecting biological information processing. Unlike conventional artificial neural networks, learning in SNNs is governed by the precise timing of neuronal

spikes, making temporal dynamics a central element of computation and adaptation.

Among neuron models, the Izhikevich model provides a widely adopted compromise between biological realism and computational efficiency, enabling the simulation of diverse firing patterns with low computational cost [17]. This balance makes it suitable for large-scale and real-time learning systems, particularly in control and robotics applications.

Learning in SNNs is predominantly driven by R-STDP, a biologically observed mechanism in which synaptic strength is modulated according to the relative timing of pre- and post-synaptic spikes [18]. R-STDP provides a competitive Hebbian learning framework that enables synaptic specialization through temporal correlations. Extensions of R-STDP incorporating neuromodulatory signals, such as dopamine, address the distal reward problem by linking delayed reward feedback to earlier spike events through eligibility traces [16]. Complementary biophysical studies further attribute R-STDP dynamics to intracellular calcium-dependent mechanisms, suggesting a deeper physiological basis for synaptic modification [19].

The R-STDP has been shown to support reinforcement learning behaviors in SNNs, enabling agents to learn from sparse and delayed feedback. Such learning rules have been applied successfully in decision-making tasks, including multi-agent reinforcement learning scenarios such as the Iterated Prisoner’s Dilemma, where SNNs demonstrate competitive or superior performance compared to conventional neural models [20]. Similar advantages have been observed in dynamic environments, where SNNs outperform multilayer perceptrons due to their efficient temporal processing and rapid response to changing inputs [21].

To bridge the gap between biologically plausible learning and machine learning efficiency, several approaches approximate gradient-based optimization within SNNs. Methods such as BP-R-STDP, SpikePropamine, and other differentiable plasticity frameworks enable multi-layer learning by approximating backpropagation through spike timing and synaptic dynamics [22, 23]. Hierarchical Bayesian inference and probabilistic formulations further extend SNNs to structured learning tasks such as pattern recognition and uncertainty-aware inference [24]. Transfer learning and parameter optimization techniques improve scalability and

generalization, enabling SNNs to be applied to complex control and perception problems [25–27].

In robotics and autonomous systems, SNNs have been applied to real-time control tasks, including autonomous navigation, obstacle avoidance, and multi-task learning. Modified integrate-and-fire models combined with R-STDP-based learning rules enable mobile robots to adapt across tasks such as target tracking and collision avoidance [28]. Hybrid architectures that integrate SNNs with convolutional neural networks further enhance learning efficiency in high-dimensional sensory spaces, particularly under sparse reward conditions [29]. These studies highlight the suitability of SNNs for adaptive control in environments characterized by uncertainty and limited computational resources.

Multi-agent systems (MAS) play a critical role in cooperative tasks such as formation control and docking, particularly in aerospace applications. Event-triggered consensus mechanisms have been widely studied to achieve coordination while reducing communication overhead, including impulsive control strategies for rapid convergence [30], dynamic event-triggered schemes to prevent Zeno behavior [31], and fixed-time consensus methods ensuring convergence within predefined time bounds [32]. Neural-network-based adaptive controllers and deep reinforcement learning frameworks further enhance coordination under nonlinear dynamics and uncertainty [33–35]. Robust docking control under constraints such as sensor faults, input saturation, and strict timing has been addressed through barrier Lyapunov methods and prescribed-time control strategies [36, 37].

Vision-based docking has been demonstrated across marine, aerial, and space domains. Monocular and marker-based systems enable precise relative pose estimation for underwater and aerial docking using lightweight sensors [38, 39]. In spacecraft applications, adaptive controllers enforcing time-varying state constraints ensure safe docking under unknown payload dynamics [40]. Cooperative docking using potential-field guidance and geometric control has been validated in multi-UAV and VTOL platforms [41, 42]. Modular and heterogeneous robot teams further extend docking capabilities through layered control architectures and self-assembling mechanisms [43, 44]. While these approaches demonstrate high precision

in structured scenarios, they rely heavily on model-based design and are less adaptable to unmodeled disturbances.

Event-based vision sensors, such as DVS, provide an alternative sensing paradigm by asynchronously encoding brightness changes as sparse events. This enables microsecond-scale temporal resolution, high dynamic range, and significant reductions in power consumption compared to frame-based cameras [45–47]. These properties are particularly advantageous for proximity operations and docking tasks, where rapid response and energy efficiency are critical.

SNNs naturally complement DVS by processing event streams directly through spike-based computation. R-STDP-based learning has been shown to extract motion features and temporal correlations from raw DVS data with high noise robustness [47, 48]. Hierarchical SNN architectures with lateral inhibition learn complex spatiotemporal features in a single pass, achieving high accuracy with minimal training [49]. Recent work extends these systems to deep architectures and reinforcement learning scenarios, demonstrating substantial energy savings and competitive performance in object localization and UAV navigation tasks [26, 50, 51]. Event-based geometric feature extraction and optical flow estimation further enable low-latency perception suitable for docking and alignment tasks [46, 52].

Reinforcement learning (RL) in SNNs enables agents to learn control policies through trial-and-error interactions, making it well suited for autonomous docking and proximity operations. The R-STDP combined with eligibility traces supports end-to-end policy learning without backpropagation [53–55]. Scaling these methods to deep SNNs requires addressing challenges such as vanishing spike activity and information propagation. Techniques including Spike Activation Lift Training, weighted spike encoding, and optimized parameter initialization improve learning stability and reduce inference latency [56–58]. Neuromorphic hardware platforms further support energy-efficient deployment of deep SNNs for real-time control [59, 60].

Federated Learning (FL) provides a distributed training paradigm that enables multiple agents to collaboratively learn a global model while keeping data local. Classical methods

such as FedAvg, FedProx, Scaffold, FedNova, and MOON address challenges related to data heterogeneity and communication efficiency [61–65]. Extensions of FL focus on selective aggregation, incentive mechanisms, hierarchical coordination, and reinforcement learning-based collaboration in edge and IoT environments [66–77].

Applying FL to SNNs introduces additional challenges due to spike-based dynamics and limited on-device data. Recent studies propose online and asynchronous federated schemes for SNNs that exchange selective synaptic information rather than full gradient updates, achieving improved accuracy–communication trade-offs [1]. Other works address energy constraints, wireless bandwidth limitations, and privacy preservation in distributed neuromorphic learning [3, 78, 79]. These studies indicate that federated neuromorphic learning remains an emerging field, particularly for multi-agent systems operating under strict resource constraints.

Existing research demonstrates that SNNs, when combined with reward-modulated learning, event-based sensing, and federated aggregation, offer a promising foundation for adaptive, energy-efficient, and scalable control in multi-agent docking scenarios. However, integrating biologically plausible learning rules with stable weight regulation, asynchronous federated updates, and event-driven perception remains an open challenge. These limitations motivate the methods developed in this thesis.

Chapter 2

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

2.1 Introduction

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

The neural structure in SNNs helps us implement complex learning systems. One good example of a complex situation is a differential game like the Active Target Defense (ATD) problem [80].

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

modeling attacker-defender interactions [81].

There are two distinct outcomes to the ATD game, characterized by two termination sets [82]. The first outcome happens when the invader reaches the target while the defender is far from it. The second outcome happens when the defender reaches the target while the invader's distance from the target is larger than the defender's distance.

2.2 The ATD problem and SNN-based solution

The ATD problem has various solving methods, such as Apollonius Circle and Cartesian Ovals (CO). This chapter opts for the CO method over the Apollonius Circle because it considers the capture radius and effectiveness against superior invaders [82]. The optimal capture point is considered the minimum distance between the target's position and reachable region if the target is inside the defender's dominant region. The defender's dominant region is a region where the defender can reach the target without letting the invader capture the target.

In this chapter, the problem is solved using reinforcement learning. It is considered that each agent knows the relative velocity of other agents. Figure 2.1 shows LOS angles used as the input for the SNNs [83].

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

2.3 Learning using R-STDP

R-STDP is a biological learning algorithm that is believed to underlie certain learning mechanisms in the brain [84]. The R-STDP algorithm is based on the idea that if a pre-synaptic neuron fires just before a post-synaptic neuron and the network receives a reward, the

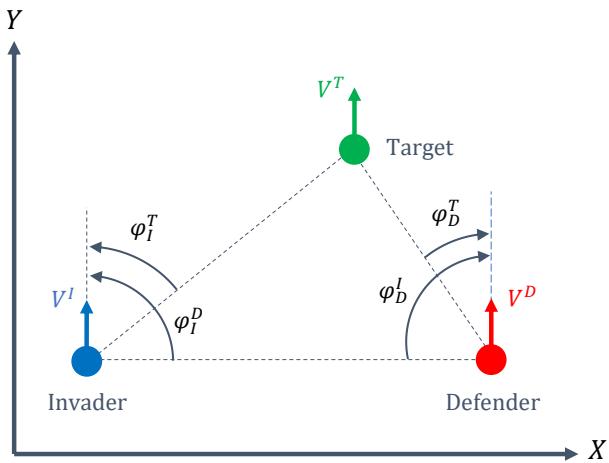


Figure 2.1: Active target defense game with three agents (LOS angles are shown for both agents).

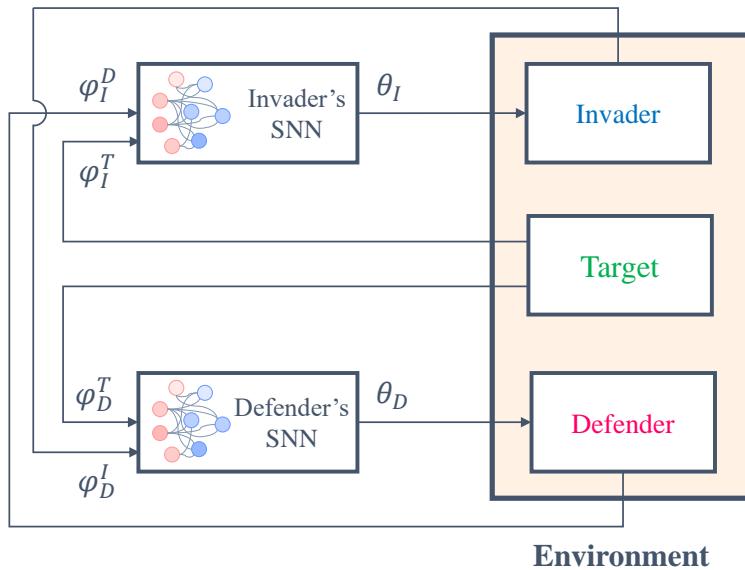


Figure 2.2: Two SNNs simultaneously play and control the invader and defender. Each agent uses the LOS angle and relative velocities for training.

strength of the synapse between the two neurons should be increased.

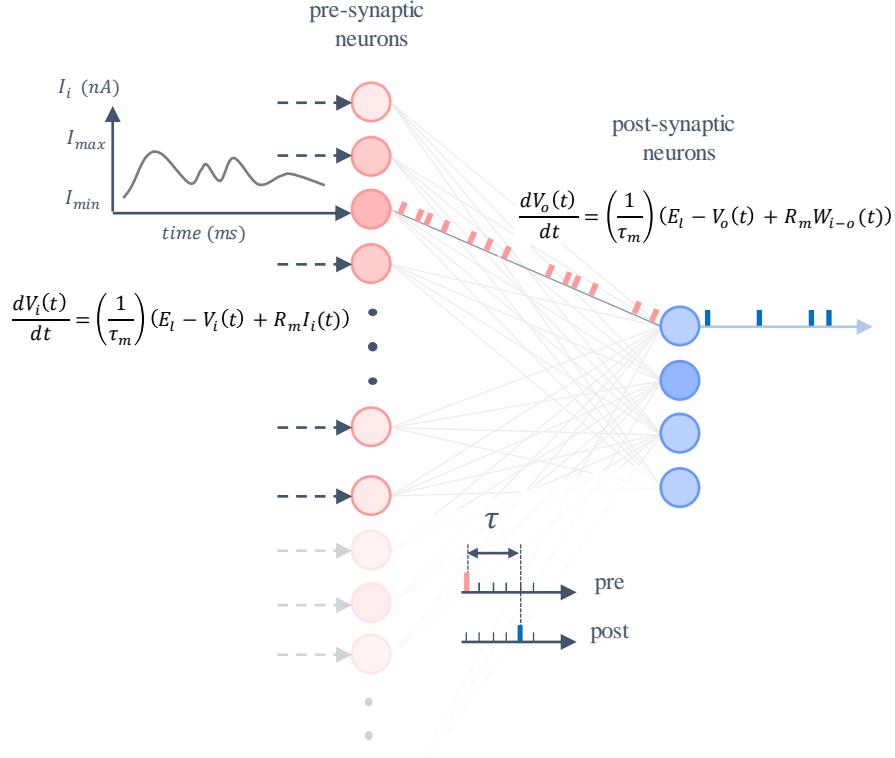


Figure 2.3: An illustration of input current to SNN and synaptic current to the output neurons after training for a single connection.

In the SNN, pre-synaptic neurons correspond to input neurons, while post-synaptic neurons correspond to output neurons. Synaptic plasticity is governed by an eligibility trace, denoted by $C^{i \rightarrow j}$, which accumulates spike-timing correlations and decays over time. The eligibility trace evolves according to [16]

$$\dot{C}^{i \rightarrow j}(t) = -\frac{C^{i \rightarrow j}(t)}{\tau_C} + \sum_{t_{\text{pre}}, t_{\text{post}}} \text{STDP}^{i \rightarrow j}(t_{\text{post}} - t_{\text{pre}}) \delta(t - t_{\text{post}}), \quad (2.1)$$

where $C^{i \rightarrow j}(t)$ is the eligibility trace associated with the synapse from pre-synaptic neuron i to post-synaptic neuron j , and τ_C is the eligibility decay time constant. The variables t_{pre} and t_{post} denote the firing times of the pre- and post-synaptic neurons, respectively. The Dirac delta function $\delta(t - t_{\text{post}})$ ensures that the eligibility trace is updated only at post-

synaptic spike times, while the magnitude of the update depends on the relative spike timing $t_{\text{post}} - t_{\text{pre}}$ through the R-STDP kernel.

The spike-timing-dependent plasticity function is defined as

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

where $\tau = t_{\text{post}} - t_{\text{pre}}$ is the spike timing difference, A_+ and A_- are the amplitudes of potentiation and depression, respectively, and τ_s is the R-STDP time constant that determines the temporal sensitivity of synaptic modification.

The parameter τ_s controls how strongly synaptic updates depend on precise spike timing. For small τ_s , the exponential terms decay rapidly, and only closely timed pre- and post-synaptic spikes contribute significantly to the eligibility trace. As τ_s increases, the temporal window of interaction widens, allowing spikes separated by larger time intervals to influence synaptic plasticity. In the limiting case $\tau_s \rightarrow \infty$, the exponential terms converge to unity, and the STDP function becomes independent of spike timing. In this regime, synaptic updates no longer encode temporal causality and reduce to constant-amplitude contributions at each post-synaptic spike, eliminating the temporal selectivity that characterizes spike-timing-dependent plasticity.

The synaptic weights are changed according to the following equation,

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

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

The R-STDP time constant τ_s is selected in relation to the synaptic weight update interval ΔT . The objective is to ensure that spike pairs occurring anywhere within the update interval produce a meaningful contributions to the eligibility trace. To this end, τ_s is

chosen such that the exponential R-STDP kernel decays substantially, but not completely, over the duration of ΔT . In this chapter, the weight update interval is set to $\Delta T = 10$ ms, and τ_s is selected as $\tau_s = 3$ ms. With this choice, when pre- and post-synaptic neurons fire close in time near the beginning of the update interval, the resulting eligibility trace retains a nonzero value when the time reaches the end of the interval, allowing the subsequent reward signal to modulate the corresponding synaptic weight.

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 [84],

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

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

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

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

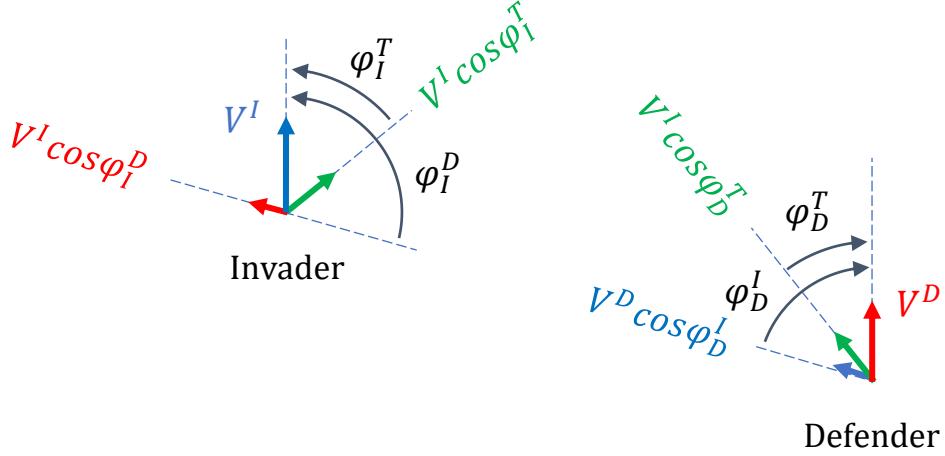


Figure 2.4: Relative velocities and LOS angles used in the reward function

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

and

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

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

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

and

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

In (2.5)-(2.8), 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),

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

and

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

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

2.4 Network structure and encoding method

Figure 2.5 illustrates the defender’s network structure and encoding process. The invader employs the same network architecture with different inputs. The network receives the line-of-sight (LOS) angles and converts them into fuzzy membership values (FMVs) using Gaussian Receptive Fields (GRF) [85]. Since the fuzzy membership values are used solely for encoding continuous inputs into the spiking neural network, the specific choice of the membership function does not affect the computational complexity of the model.

Let $x \in \mathbb{R}$ denote an input variable (e.g., a LOS angle). The Gaussian receptive field associated with the i -th input neuron is defined as

$$\mu_i(x) = \exp\left(-\frac{(x - c_i)^2}{2\sigma_i^2}\right), \quad (2.11)$$

where c_i is the center of the receptive field and σ_i controls its width. Each input neuron is assigned a distinct receptive field, and the corresponding membership value $\mu_i(x) \in [0, 1]$ represents the degree to which the input activates that neuron.

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 are converted into input currents.. This can be done using a linear function and the minimum and maximum inputs.

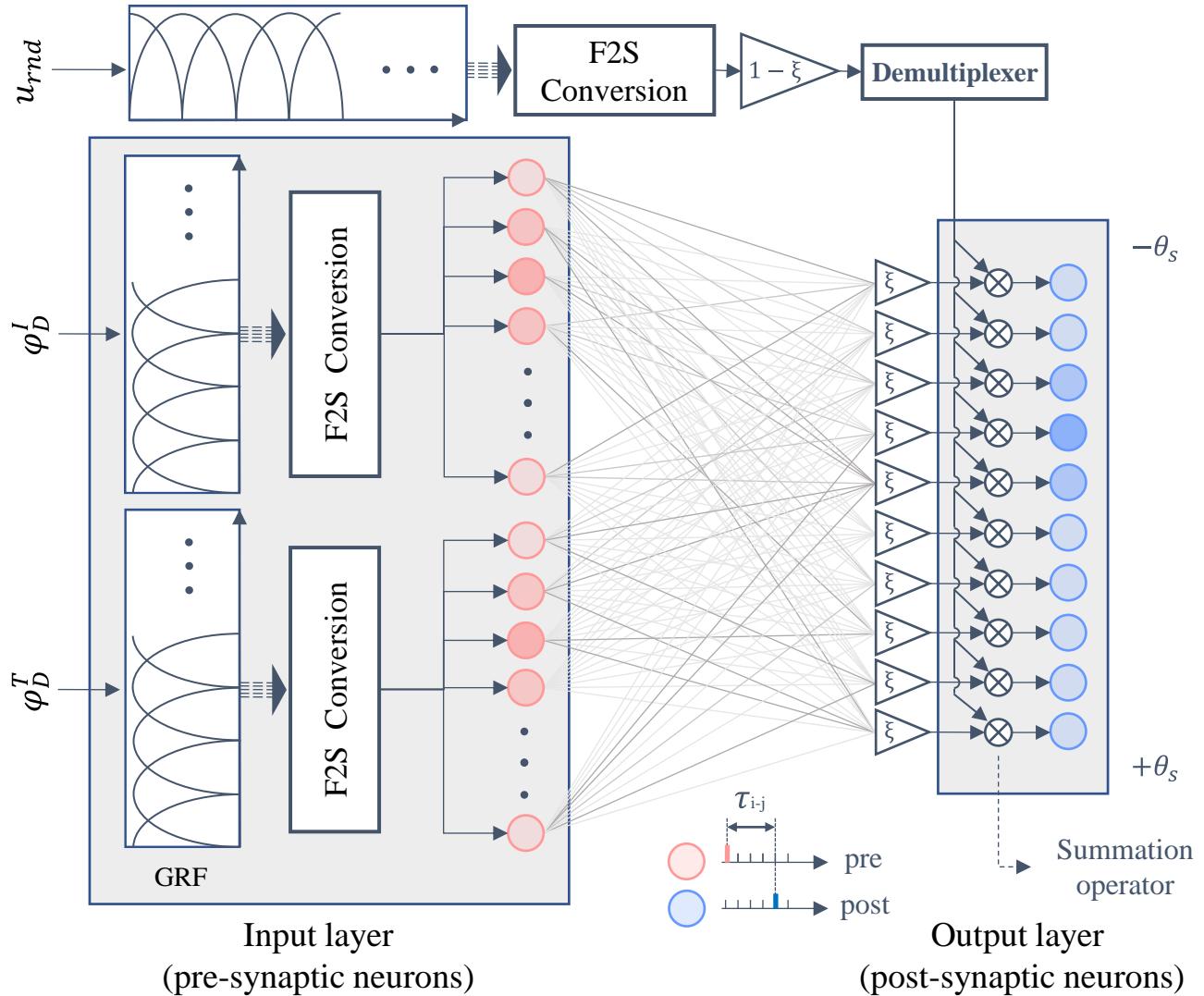


Figure 2.5: Network structure and encoding process for the input layer (Defender). Each neuron is associated with a membership function in GRF. The GRF encodes an input State (S^t) at each time step. There is both a training phase when ($\xi = 0$) and an operating phase when ($\xi = 1$). The “F2S Conversion” block converts fuzzy membership values to currents for the neurons according to (2.12).

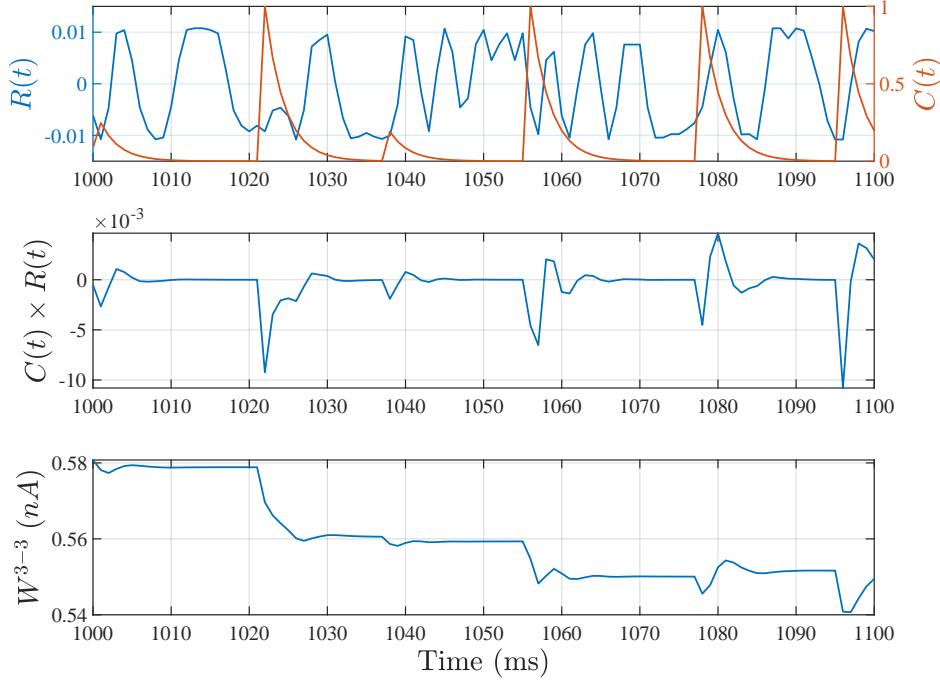


Figure 2.6: Reward, eligibility trace, and weight change during the simulation for W^{3-3} . The $C(t)R(t)$ changes the synaptic weight by considering activation strength and reward value.

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

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

or

$$I_\sigma = \frac{\tau_m (V_{th} - V_{res})}{\Delta t R_m} FMV_\sigma + \frac{V_{th} - E_l}{R_m} \quad (2.12)$$

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

Algorithm 1 Weight training algorithm

```

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

```

The output of the SNN is the steering angle (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,

$$I_{out} = \xi I_{syn} + (1 - \xi) I_{rnd} \quad (2.13)$$

where ξ 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 ξ is 0 during the training, the I_{syn} does not affect the SNN's output (Equation 2.13).

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

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

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

2.5 Results

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

Table 2.1: Parameter values for LIF neuron model [86]

Parameter	Value	Description
R_m	40 M Ω	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

Table 2.2: 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

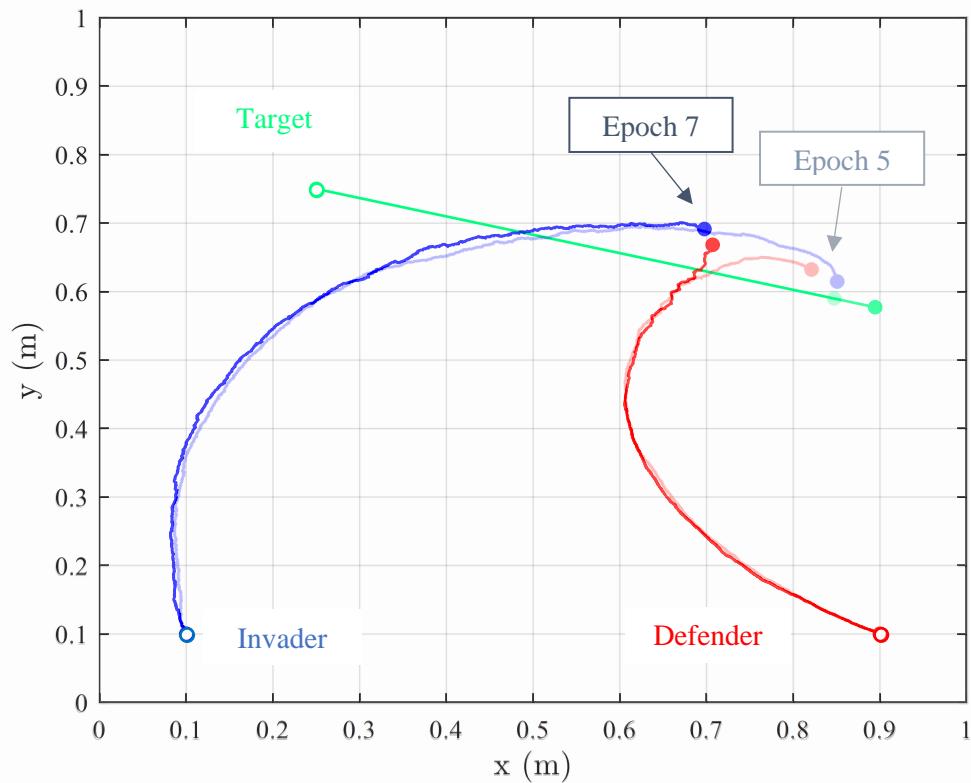


Figure 2.7: SNN's performance during training. Hollow circles show the initial positions.

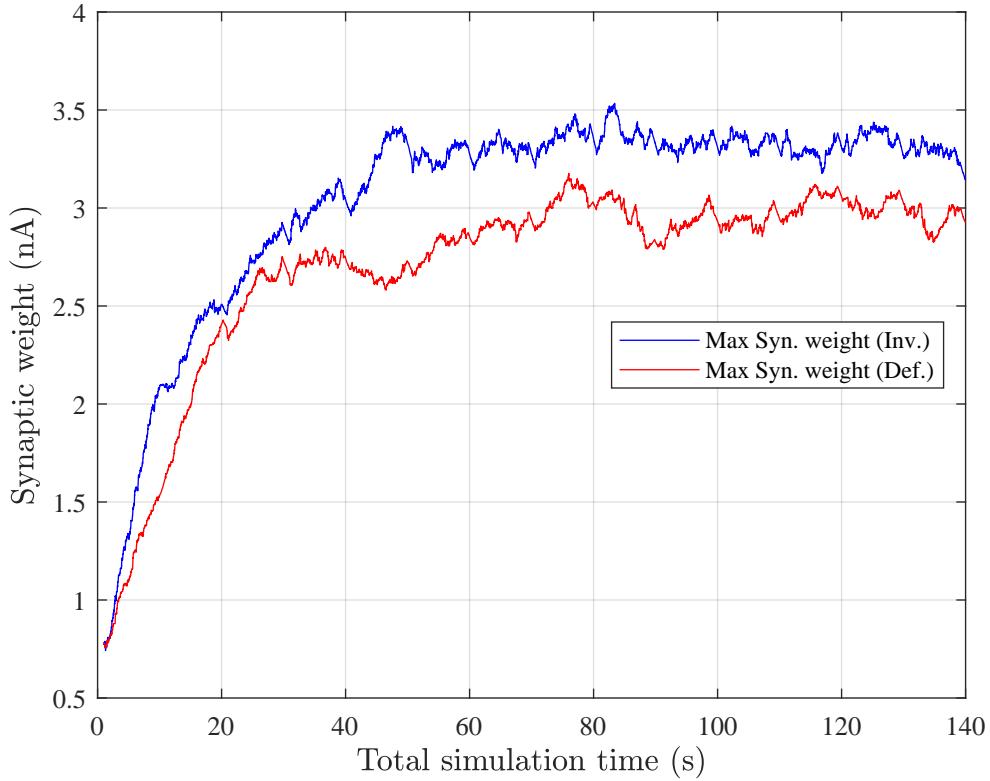


Figure 2.8: Changes in synaptic weights during training. Only Maximum synaptic weights for both agents are shown.

After several simulations, the number of input neurons for invader and defender networks was set to 20. Both agents have 10 neurons in their output layer. The encoding resolution can be enhanced by increasing the number of neurons, although this results in a higher number of synaptic connections. An optimization algorithm can be employed to determine the optimal neuron number in the SNN. Half of the input neurons for the invader are for the 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 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 $-\pi/4 < \theta_s < \pi/4$ for invader and defender, and the σ is set to 1.25 . The parameter values for R-STDP, shown in Table 2.2, are set through several simulations.

The reward coefficients are set manually. The τ_s in Table 2.2 defines the decaying rate of C in (2.1) that determines the R-STDP sensitivity to prior firings. According to (2.2), higher 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 [87].

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

2.5.1 Simulation without noise

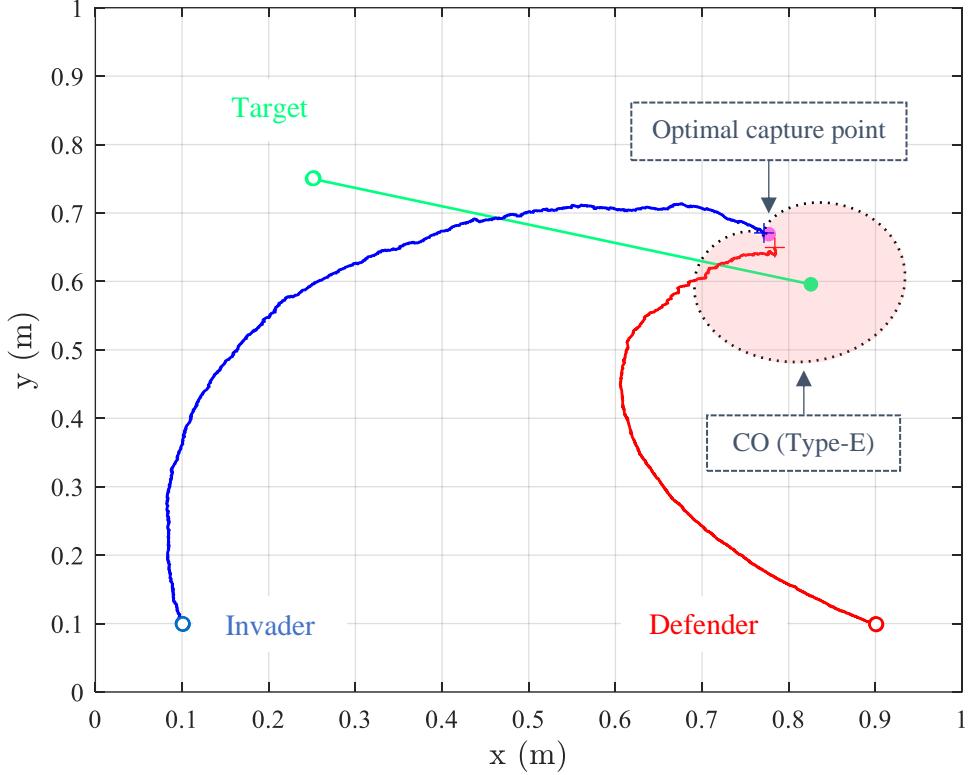


Figure 2.9: SNN’s performance after training. The highlighted region shows the defender’s dominant region. The purple dot is the optimal capture point. The CO Type-E shows the reachable regions of the Invader and the Defender.

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

According to Figure 2.9, the invader’s SNN can find the optimal [82] capture point for the invader, while the defender’s SNN can protect the moving target against a superior invader. It should be noted that this solution is obtained without a global reference frame. This

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

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

2.5.2 Simulation with noise

In Figure 2.10, we observe the performance of two methods, namely the SNN and the Cartesian Oval (CO) method, in the presence of noise. The CO method computes the capture location by explicitly constructing the boundary that separates the reachable regions of the attacker and the defender under constant-speed motion. This boundary is defined as a CO, which represents the locus of points for which the time-to-go of both players is equal. Let (x_A, y_A) denote the attacker position, d the distance between the attacker and defender, ρ the initial distance between the attacker and the target, and $\gamma = v_D/v_A$ the defender-to-attacker speed ratio. The capture boundary is parameterized as

$$x = x_A + r(\phi) \cos(\lambda + \phi + \pi), \quad y = y_A + r(\phi) \sin(\lambda + \phi + \pi), \quad (2.14)$$

where λ is the line-of-sight angle from the attacker to the defender and ϕ is a free angular parameter. The radial function $r(\phi)$ is given by

$$r(\phi) = \frac{\gamma\rho + d \cos \phi \pm \sqrt{(\gamma\rho + d \cos \phi)^2 - (1 - \gamma^2)(d^2 - \rho^2)}}{1 - \gamma^2}, \quad (2.15)$$

with ϕ constrained to the interval $\phi \in [-\phi_l, \phi_l]$, where

$$\phi_l = \arccos\left(\frac{\sqrt{(1-\gamma^2)(d^2-\rho^2)} - \gamma\rho}{d}\right). \quad (2.16)$$

The capture location is determined by evaluating this parametric curve and selecting the point on the oval that satisfies the game termination condition, i.e., equal arrival time of the attacker and defender to the target. Since the oval is constructed directly from instantaneous position measurements, any perturbation in the measured states affects the shape and position of the oval, making the resulting capture point sensitive to measurement noise.

The noise in this experiment is introduced as white Gaussian noise, characterized by a mean of zero and a variance of 0.01. Both SNN and CO receive position data that has been corrupted by this noise.

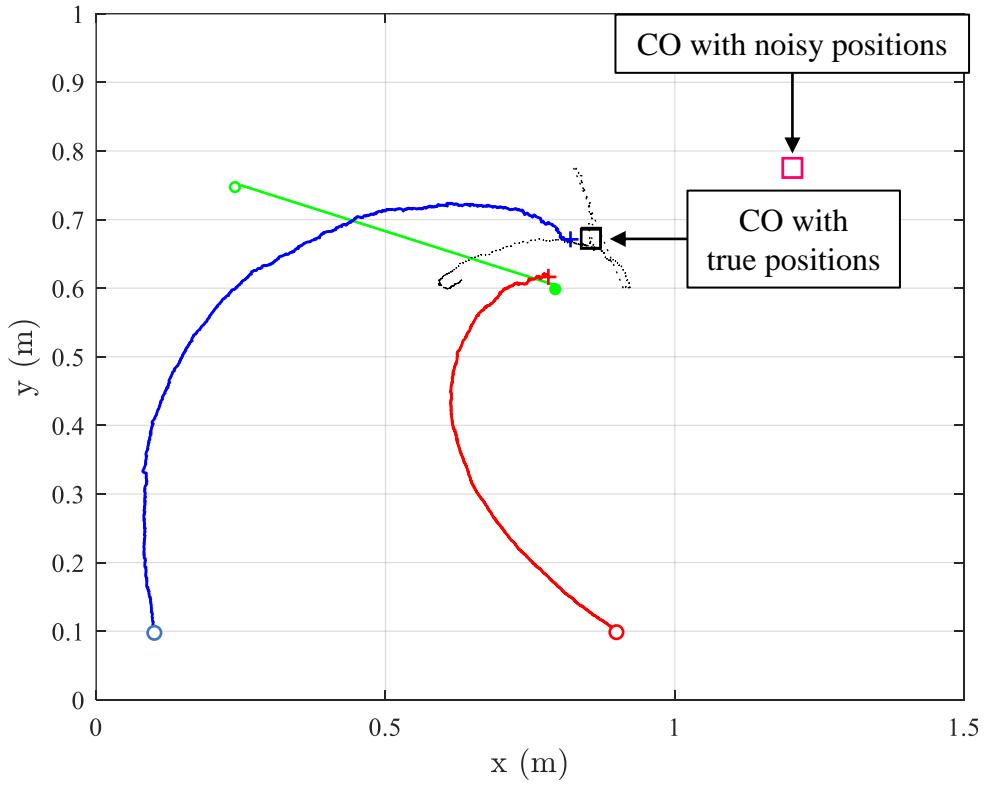


Figure 2.10: SNN’s performance in noisy conditions. A white Gaussian noise with a variance of 0.01 is added to the measured inputs.

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

2.6 Conclusion

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

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

Chapter 3

Integration of R-STDP and Federated Learning

3.1 Consensus Flying Problem

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

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

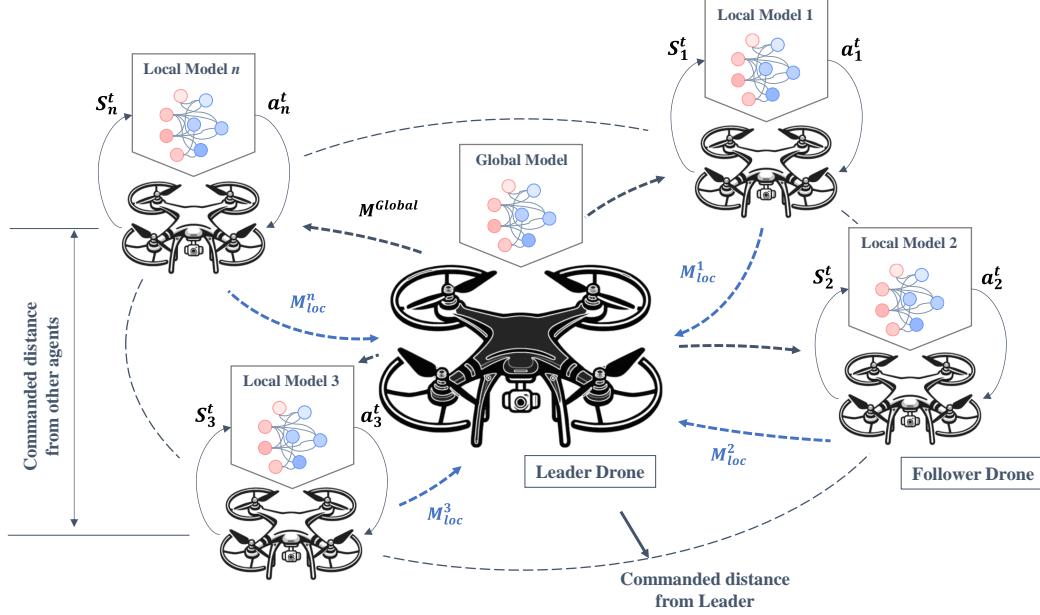


Figure 3.1: The central server (the leader) and the surrounding follower agents (white drones). The follower agents learn to fly in a formation to maintain the commanded distance. The Local models trained individually by follower agents are sent to the leader. The leader aggregates the models and sends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

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

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

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

3.2 Proposed Method

3.2.1 Network Structure

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

Every input sub-layer is divided into two parts. The first part represents distances greater than the commanded distance, while the second part corresponds to distances smaller than the commanded value. Within each part, the LOS angle is encoded using Gaussian membership functions. The difference between the current distance and the commanded

distance is treated as an error signal. This error is transformed into an amplitude using the hyperbolic tangent function so that it is bounded between 0 and 1. An error of zero produces zero amplitude, while large errors asymptotically lead to unit amplitude. Accordingly, the fuzzy encoding function for the input layer is defined as

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

where ζ and σ denote the center and standard deviation of the Gaussian membership functions, respectively. The variable r_i represents the distance to the corresponding agent, ϕ_i is the LOS angle, and μ_I is the resulting membership degree. The variable r acts as a placeholder and represents either r_{cmd} or R_{cmd} , depending on whether the interaction is between followers or between a follower and the leader.

The firing strengths produced by the fuzzy encoders are converted into input currents for the spiking neurons using the neuron model dynamics. The fuzzy-to-spiking conversion is given by [88]

$$I_{sub-layer} = \frac{\tau_m (V_{th} - V_{res})}{\Delta t R_m} \mu_I(\phi_i, r_i) + \frac{V_{th} - E_l}{R_m}, \quad (3.2)$$

where τ_m is the membrane time constant, R_m is the membrane resistance, Δt is the simulation time step, V_{th} and V_{res} denote the firing threshold and reset potentials, respectively, and E_l is the leak reversal potential.

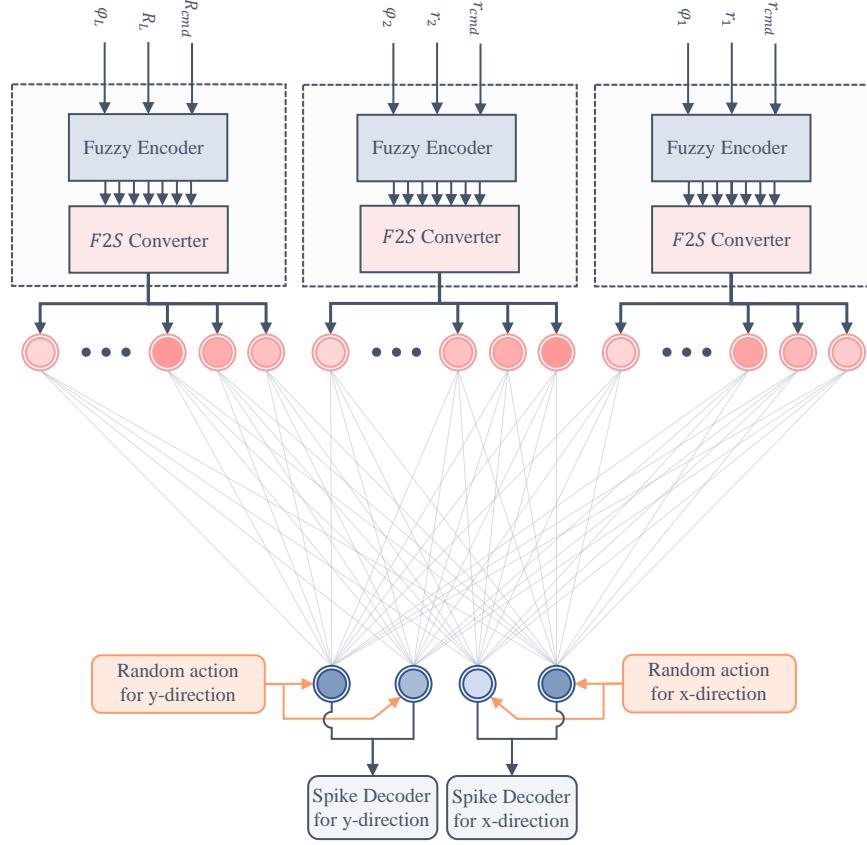


Figure 3.2: SNN structure with encoding and decoding layers. Each input sub-layer consists of a fuzzy encoder followed by the fuzzy-to-spiking current conversion defined in (3.2). During the training phase, the output layer receives input only from the random action selector, which is replaced by synaptic weight inputs during the testing phase.

For completeness, an alternative linear mapping between fuzzy membership values and input current can also be employed,

$$I_{\text{sub-layer}} = (I^{\max} - I^{\min}) \mu_I(\phi_i, r_i) + I^{\min}, \quad (3.3)$$

where I^{\max} and I^{\min} denote the maximum and minimum allowable input currents defined in (1.5) and (1.9). This formulation provides a simplified current scaling while preserving the same encoding structure.

The output layer has two sub-layers, and each sub-layer has two neurons. The first

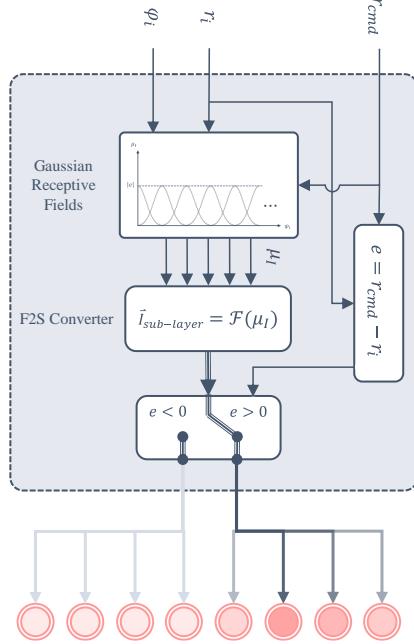


Figure 3.3: The fuzzy encoding principle for the input sub-layer.

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 (3.2) 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 and 1 as follows,

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

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

where Δx and Δy are selected actions, and ΔX_{max} and ΔY_{max} are maximum steps (displacements) in X and Y directions. 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 $f(t)$ as the activity of the output neurons that control the movement in the x and y-directions:

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

The equation for decoding this activity can be expressed as:

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

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

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

where ΔT is the time window that the network uses to update the weights.

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

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

In this equation, Π_t represents the random action at time t (shown as random actions in X and Y-directions in Figure 3.2), γ is a correlation coefficient that controls the influence of the previous action, and Υ_t is a random number drawn from a standard distribution (e.g.,

Gaussian) at time t . This recursive formulation ensures that the action at any given time t is a weighted blend of the previous action and a new random input, functioning as a first-order filter to produce colored noise.

3.2.2 Training algorithm

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

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

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

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

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

One of the challenges in R-STDP is the unbounded growth or decay of synaptic weights, which can impede stable and effective learning in neural networks. The following section

introduces a novel method focused on learning rate and weight stabilization to address this challenge and enhance the algorithm's applicability. This proposed method, designed to regulate synaptic weight changes, ensures a balanced and controlled learning process. It innovatively incorporates an adaptive decay rate technique designed to maintain stability in synaptic weight adjustments, thereby significantly improving the performance and reliability of R-STDP in SNNs.

3.2.3 Weight Stabilization using Reward-Modulated Competitive Synaptic Equilibrium (RCSE)

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

Traditional methods like L1 regularization and weight decay employ a constant decay rate, which can slow the network's responsiveness to changes in rewards. Alternatively, a more advanced approach, the Bienenstock, Cooper, and Munro (BCM) method dynamically adjusts both a threshold and a decay rate in response to input variations [89]. However, this method does not provide a control mechanism for the fuzzy inputs. In this chapter, a method called Reward-Modulated Competitive Synaptic Equilibrium (RCSE) is introduced to control the unbounded growth of synaptic weights. The proposed method preserves the gradual evolution of synaptic strengths arising from differences in firing strength, as generated by fuzzy membership functions. In addition, the network can adapt to changing rewards (caused by environmental changes) by dynamically adjusting the maximum allowable synap-

tic weight.

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

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

where \odot denotes the Hadamard product, $\boldsymbol{\alpha}$ is the learning-rate matrix, and $\boldsymbol{\Theta}$ is the decay-rate matrix. In contrast to the previously proposed method in Section 2.4, the proposed RCSE method does not require an explicit separation between training and testing phases, nor does it rely on manually enabling or disabling learning.

In the RCSE framework, the network continuously receives reward signals at all times. During learning, the interaction between the R-STDP term and the adaptive decay mechanism drives the synaptic weights toward an equilibrium that is a function of the rewards. Once this equilibrium is reached, the synaptic weights do not become fixed. Instead, they exhibit small bounded oscillations around the converged value due to the competitive balance between the learning-rate and decay-rate terms.

This oscillatory behavior plays a key functional role. Because the learning-rate matrix $\boldsymbol{\alpha}$ is defined as a function of the maximum synaptic weight, it naturally transitions between active ($\alpha \approx 1$) and inactive ($\alpha \approx 0$) states as the weights fluctuate around the equilibrium threshold. As a result, the network remains sensitive to changes in the reward signal even after convergence. When the reward structure changes, the learning rate can automatically reactivate without requiring external intervention or monitoring of the synaptic weights.

Therefore, in the RCSE method, the distinction between training and testing phases emerges implicitly from the synaptic dynamics rather than being imposed manually. The adaptive learning-rate and decay-rate mechanisms jointly regulate when learning is suppressed and when it is re-enabled. This design allows the network to maintain stability after convergence while still enabling rapid adaptation to reward changes, eliminating the need for explicit phase switching or heuristic control logic.

Let us define \mathcal{S} as the set of input and output neurons that fired at time t in one of the

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

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

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

The network contains a variety of reward functions, each with its own maximum and minimum values. The highest reward value in a specific area of the network sets the limit for the synaptic weight in that area. The synaptic weight limit is linked to the ratio of the local maximum reward (\mathcal{R}_{max}^S) to the global maximum reward (\mathcal{R}_{max}^G). As a result, the network section with the highest local maximum reward ($\mathcal{R}_{max}^S = \mathcal{R}_{max}^G$) attains the maximum allowable synaptic weights because $\frac{\mathcal{R}_{max}^S}{\mathcal{R}_{max}^G} = 1$, while sections with lower local maximum rewards reach only a proportional fraction of the maximum weight. Adjusting the learning rate introduces a competitive mechanism among synaptic connections. Each synaptic weight is allowed to grow only while it remains below a reward-dependent threshold. The maximum synaptic weight within an active set determines whether learning continues or is suppressed for neighboring connections. As a result, synaptic growth is regulated based on both the reward magnitude and the current distribution of synaptic weights in the network. This competitive behavior prevents unbounded weight growth while preserving relative differences between synapses that arise from variations in firing strength.

A significant challenge in learning algorithms is their capacity to adapt to changes in rewards (e.g., the roles of objects within the mission may change, resulting in the leader

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

In this chapter, the decay rate is defined as an explicit function of the maximum synaptic weight among the neurons belonging to the active set \mathcal{S} . The purpose of this design is to regulate synaptic growth based on the network's current state rather than using a fixed decay value. The maximum synaptic weight $|W_{max}^{\mathcal{S}}(t)|$ serves as an indicator of how close the network section \mathcal{S} is to its reward-dependent equilibrium.

When the maximum synaptic weight approaches the threshold value $\Psi^{\mathcal{S}}$, the network is considered to be near convergence for the corresponding reward condition. At this point, it is important that learning does not completely stop. If the learning rate were forced to zero, the synaptic weights would become frozen, and the network would lose its ability to respond to future changes in the environment (reward). Therefore, the decay mechanism is designed such that the learning rate remains above zero when $|W_{max}^{\mathcal{S}}(t)| = \Psi^{\mathcal{S}}$, allowing small weight adjustments to continue.

At the same time, synaptic updates must remain bounded. The learning rate must not exceed the maximum allowable rate of weight change, which is determined by the product $\mathcal{A}\mathcal{R}_{max}^G$. This constraint ensures stability and prevents excessive synaptic growth across the network. If the reward coefficients change after the convergence phase, the threshold $\Psi^{\mathcal{S}}$ may shift, causing the existing maximum synaptic weight to reach a new equilibrium value.

To handle this situation, the decay rate is increased as a function of the deviation between $|W_{max}^{\mathcal{S}}(t)|$ and $\Psi^{\mathcal{S}}$. Specifically, the decay rate is set to a small value $\mathcal{A}/\lambda \times \mathcal{R}max^G$ when $|W_{max}^{\mathcal{S}}(t)| = \Psi^{\mathcal{S}}$, ensuring gentle regulation near equilibrium. As the maximum synaptic weight increases beyond this threshold, the decay rate grows progressively and reaches $\lambda\mathcal{A} \times$

\mathcal{R}_{\max}^G when $|W_{\max}^S(t)| = 2\Psi^S$. The parameter λ controls how aggressively the decay term responds to weights that exceed the equilibrium region.

This adaptive decay mechanism allows the network to stabilize synaptic weights around a reward-dependent operating point while preserving sensitivity to reward changes. As a result, the network can both maintain convergence and rapidly re-adjust its synaptic structure when the learning environment changes.

The decay rate is constructed using a SoftPlus function of the form

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

where $\eta > 0$ is a scaling factor and $\beta > 0$ controls the curvature. A higher β makes the SoftPlus function approach a step function, making it closer to the binary behavior. Conversely, a smaller β makes the function smoother and more gradual. The shift by Ψ^S ensures that the transition occurs around the reward-dependent equilibrium threshold.

Let's define $x(t) = |W_{\max}^S(t)| - \Psi^S$. Two boundary conditions are imposed to determine (η, β) . First, when the maximum synaptic weight reaches the equilibrium threshold, i.e., $|W_{\max}^S(t)| = \Psi^S$ (equivalently $x = 0$), the decay rate is required to be small,

$$\Theta^S(t) \Big|_{x=0} = \frac{\mathcal{A}}{\lambda} \mathcal{R}_{\max}^G. \quad (3.14)$$

Second, when the weight exceeds the equilibrium region and reaches $|W_{\max}^S(t)| = 2\Psi^S$ (equivalently $x = \Psi^S$), the decay rate is required to be large,

$$\Theta^S(t) \Big|_{x=\Psi^S} = \lambda \mathcal{A} \mathcal{R}_{\max}^G. \quad (3.15)$$

Substituting $x = 0$ into (3.13) yields

$$\Theta^S(t) \Big|_{x=0} = \frac{\eta}{\beta} \log(1 + \exp(0)) = \frac{\eta}{\beta} \log(2). \quad (3.16)$$

Combining (3.14) and (3.16) gives the first relationship,

$$\frac{\eta}{\beta} \log(2) = \frac{\mathcal{A}}{\lambda} \mathcal{R}_{\max}^G. \quad (3.17)$$

Next, substituting $x = \Psi^S$ into (3.13) yields

$$\Theta^S(t) \Big|_{x=\Psi^S} = \frac{\eta}{\beta} \log(1 + \exp(\beta\Psi^S)). \quad (3.18)$$

Dividing (3.18) by (3.16) and using the imposed values in (3.14)–(3.15) gives

$$\frac{\log(1 + \exp(\beta\Psi^S))}{\log(2)} = \frac{\lambda\mathcal{A}\mathcal{R}_{\max}^G}{(\mathcal{A}/\lambda)\mathcal{R}_{\max}^G} = \lambda^2. \quad (3.19)$$

Therefore,

$$\log(1 + \exp(\beta\Psi^S)) = \lambda^2 \log(2) = \log(2^{\lambda^2}), \quad (3.20)$$

which implies

$$1 + \exp(\beta\Psi^S) = 2^{\lambda^2} \Rightarrow \exp(\beta\Psi^S) = 2^{\lambda^2} - 1. \quad (3.21)$$

Taking the natural logarithm yields

$$\beta = \frac{1}{\Psi^S} \ln(2^{\lambda^2} - 1). \quad (3.22)$$

Finally, substituting (3.22) into (3.17) gives

$$\eta = \frac{\mathcal{A}\mathcal{R}_{\max}^G}{\lambda \log(2)} \beta = \frac{\mathcal{A}\mathcal{R}_{\max}^G}{\lambda \Psi^S \log(2)} \ln(2^{\lambda^2} - 1). \quad (3.23)$$

Substituting (3.22)–(3.23) into (3.13) yields (3.24),

$$\Theta^S = \left(\frac{\mathcal{A}\mathcal{R}_{\max}^G}{\lambda \log(2)} \right) \log \left(1 + \exp \left[\left(\frac{\ln(2^{\lambda^2} - 1)}{\Psi^S} \right) (|W_{\max}^S(t)| - \Psi^S) \right] \right) \quad (3.24)$$

The choice of setting the decay rate to $\lambda\mathcal{A} \times \mathcal{R}_{\max}^G$ when $|W_{\max}^S(t)| = 2\Psi^S$ is based on the feature of reward coefficients. Specifically, when the reward coefficients in (3.9) and (3.10)

increase, leading to new condition where \mathcal{R}_{max}^S or \mathcal{R}_{max}^G change, the $|W_{max}^S(t)|$ is allowed to increase. Conversely, a decrease in the reward coefficient, resulting in $|W_{max}^S(t)| > \Psi^S$, necessitates a higher decay rate to reduce the $|W_{max}^S(t)|$ back to Ψ^S .

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

Numerical example for the RCSE algorithm

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

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

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

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

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

Let's assume that the maximum reward for the neurons that fired is $\mathcal{R}_{max}^S = 0.5$ and maximum global reward is $\mathcal{R}_{max}^G = 1$ ($\Psi^S = 0.5/1 \times 15.5 = 7.75$), so the maximum change rate for this set \mathcal{S} is $\mathcal{A}\mathcal{R}_{max}^S = 0.5$ while the maximum change rate in the network is $\mathcal{A}\mathcal{R}_{max}^G = 1$. For $W_{max}^S(t) = 7$, the decay rate Θ^S can be calculated as follows,

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

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

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

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

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

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

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

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

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

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

Case 3: When Ψ^S increases from 7.75 to 15.5 (Reward Change)

Assuming a reward change causes Ψ^S to increase to 15.5. The importance of an object in the output can be controlled by adjusting its reward coefficient. Increasing the importance of an object (e.g., neighboring agent or leader) involves setting its reward coefficient to the maximum value ($\mathcal{R}_{max}^S = \mathcal{R}_{max}^G = 1$). Considering $W_{max}^S(t) = 7.8$, the decay rate Θ^S is recalculated as follows,

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

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

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

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

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

Case 4: When Ψ^S decreases from 15.5 to 7.75 (Reward Change)

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

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

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

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

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

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

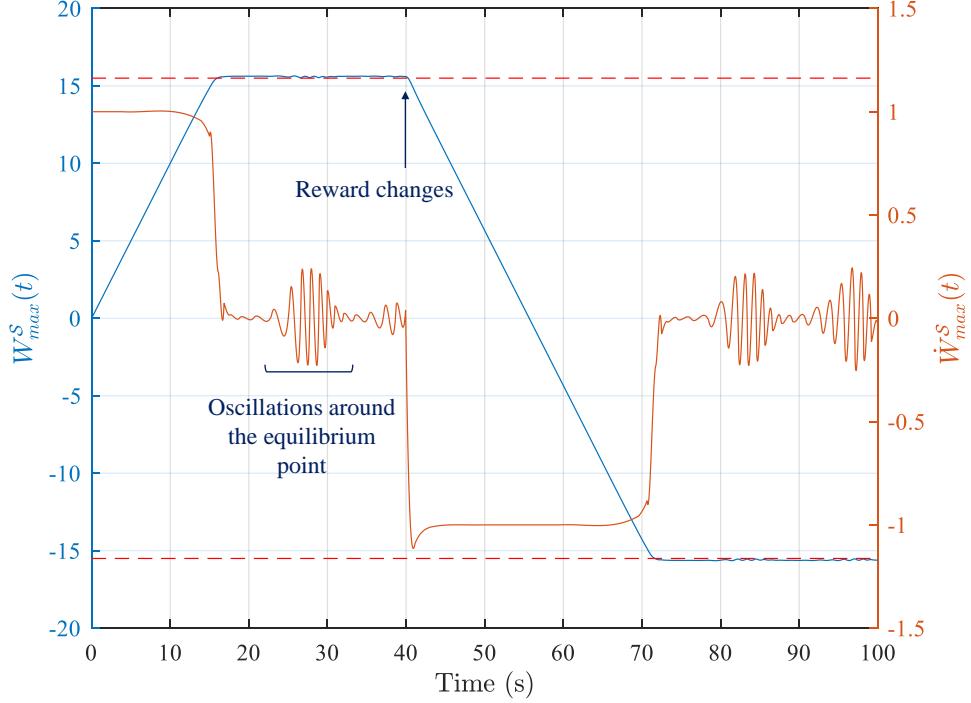


Figure 3.4: Synaptic weight change for $\lambda = 5$, $\Psi^S = 15.5$, and $\mathcal{AR}_{max}^G = 1$.

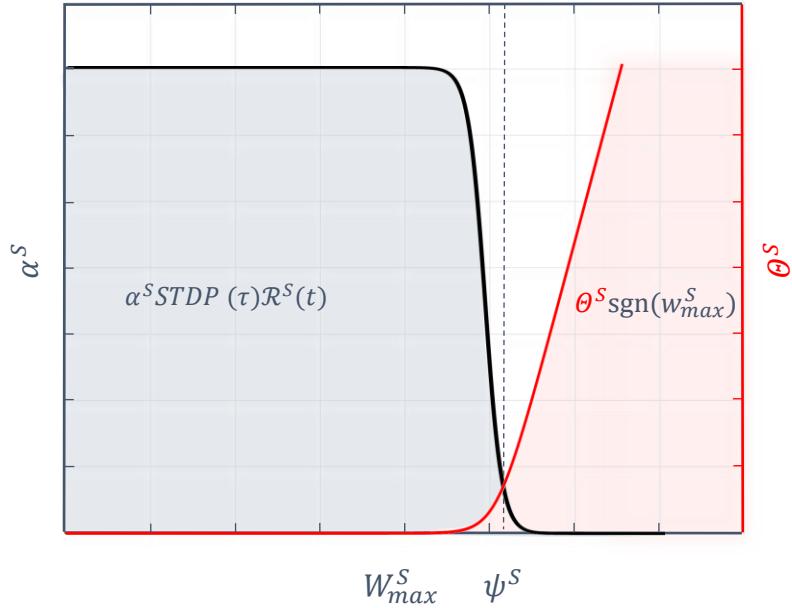


Figure 3.5: Reward-based learning rate and decay rate functions. In the blue region (active learning rate), the reward adjusts the weights, and in the red region (active decay rate), the RCSE method controls synaptic growth.

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

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

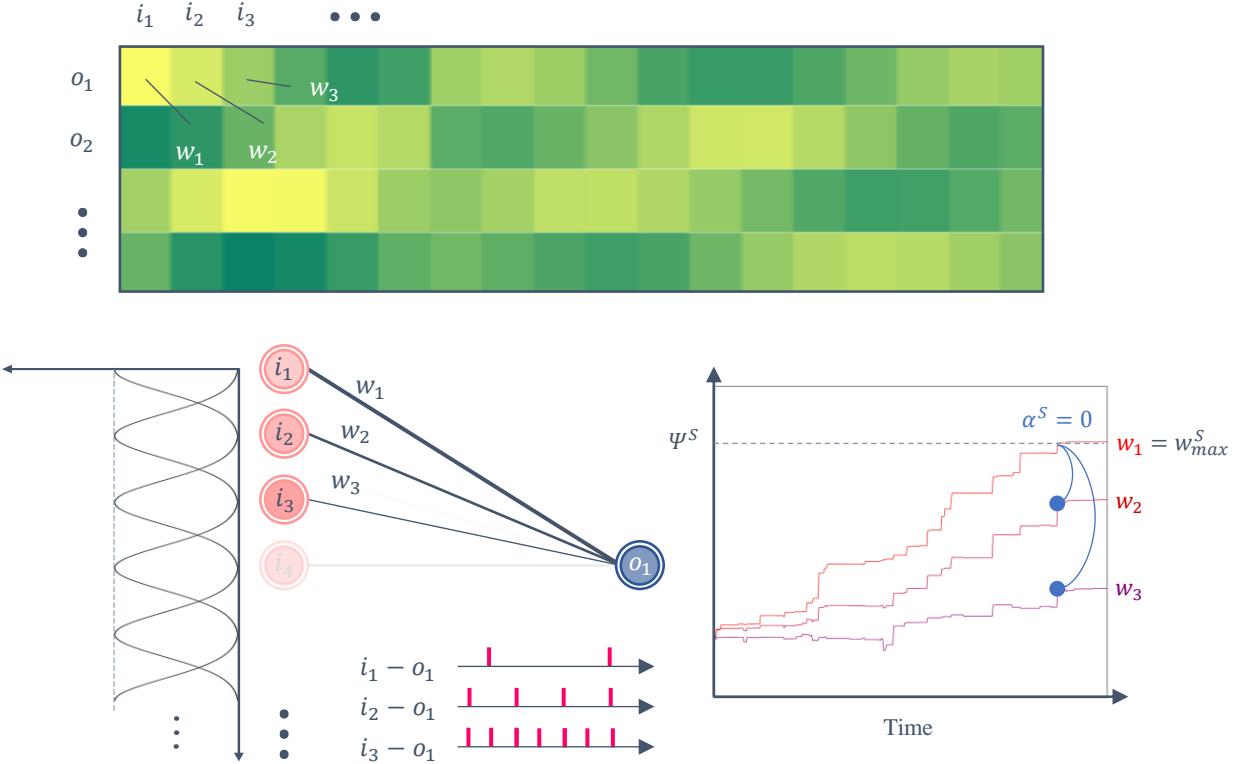


Figure 3.6: RCSE working principle in inhibiting the adjacent synaptic connections. The heatmap shows the synaptic weight matrix. Neurons have different firing strengths due to the difference in fuzzy membership values, which affects the increase or decrease rate and shapes the patterns in the synaptic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

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

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

3.2.4 Federated Learning for Consensus Flying

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

This approach allows clients to upload their local model updates at different times rather than synchronously. Such a method is particularly beneficial in reducing the negative impacts of device heterogeneity, which can include varying computational capacities and network connectivity among devices [90]. In traditional FL setups, delays caused by poor network signals or unexpected client crashes can significantly prolong the time the server takes to receive updates from all clients. By adopting asynchronous aggregation, the server processes and aggregates models as they are received without synchronizing with all clients. This strategy accelerates the training process, making FL more efficient and adaptable to diverse client conditions.

Our proposed FL model aggregation algorithm aims to establish an efficient and event-triggered system for global and local model publishing. This system relies on the similarity between consecutive global and local models and publishes updates only when significant changes are detected, thus avoiding redundant updates and improving overall efficiency. Unlike the uniform model updates in FedAvg [91,92], 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 [93], adding a layer of context sensitivity to our approach.

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

$$\bar{\mathbf{W}}_k(t) = \frac{1}{I_k^{max}} [\mathbf{W}_k(t)] \quad (3.25)$$

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

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

$$\bar{\mathbf{W}}_G(t) = \frac{\sum_{k=1}^N \omega_k \cdot \bar{\mathbf{W}}_k(t)}{\sum_{k=1}^N \omega_k} \quad (3.26)$$

where $\bar{\mathbf{W}}_G(t)$ 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,

$$\omega_k = \frac{1}{\sqrt{mn}} \|\bar{\mathbf{W}}_k(t)\|_F \exp \left(-\frac{t - T_k}{\tau_{cs}} \right) \quad (t \geq T_k) \quad (3.27)$$

where the term $\|\cdot\|_F$ is the Frobenius norm, and m and n are the dimensions of the matrix $\bar{\mathbf{W}}_k(t)$, used for normalizing the Frobenius norm. T_k indicates the time at which agent k last transmitted its local model to the central server, and 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. In the proposed method, the reward-modulated synaptic dynamics in (3.11) remain active until the synaptic weights reach a competitive equilibrium, i.e., until

the weight variations become negligible due to the coupled effect of the learning-rate gate (3.12) and the decay mechanism (3.24). This steady regime is referred to as the testing phase, during which the weights remain effectively constant under unchanged reward condition (no change in the environment). Nevertheless, because the RCSE mechanism maintains bounded oscillations around the equilibrium threshold Ψ^S , the learning-rate gate can intermittently return to $\alpha^S(t) \approx 1$ when $|W_{\max}^S(t)|$ drops below Ψ^S . Therefore, if the reward structure changes after convergence, the network can resume adaptation immediately when $\alpha^S(t)$ becomes active again, without requiring reinitialization.

Throughout the training phase, each agent calculates the Euclidean distance between the most recent global model published by the central server and its current synaptic weight matrix, as follows,

$$\mathcal{D}_a(\overline{\mathbf{W}}_k(t), \overline{\mathbf{W}}_G(T_{cs})) = \frac{1}{2\sqrt{mn}} \sqrt{\sum_{i=1}^m \sum_{j=1}^n (a_{ij} - b_{ij})^2} \quad (3.28)$$

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 $\overline{\mathbf{W}}_k(t)$ and the last model that it sent to the server at time T_k ($\overline{\mathbf{W}}_k(T_k)$) until it receives a new model update from the central server.

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

$$\mathcal{D}_G(\overline{\mathbf{W}}_G(t), \overline{\mathbf{W}}_G(T_{cs})) = \frac{1}{2\sqrt{mn}} \sqrt{\sum_{i=1}^m \sum_{j=1}^n (a_{ij} - b_{ij})^2} \quad (t \geq T_{cs}) \quad (3.29)$$

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

$$\begin{aligned}\dot{\mathbf{W}}_k(t) = & (1 - \delta(t - T_{cs})) [\boldsymbol{\alpha} \odot \mathbf{STDP}(\tau) \odot \mathcal{R}(t) - \boldsymbol{\Theta} \odot \text{sgn}(\mathbf{W}_k(t))] \\ & + \delta(t - T_{cs}) I_k^{\max} (\overline{\mathbf{W}}_G(t) - \overline{\mathbf{W}}_k(t))\end{aligned}\quad (3.30)$$

where δ is the Dirac delta function. When $t = T_{cs}$, the first term of the equation becomes zero, and the second term becomes active, allowing the agent to replace its local model with the global model from the central server.

Equation (3.30) describes the complete synaptic weight dynamics when the proposed RCSE learning rule is integrated with the federated learning update mechanism. The equation consists of two distinct components that are activated at different time instants and serve different purposes in the learning process.

The first term,

$$(1 - \delta(t - T_{cs})) [\boldsymbol{\alpha} \odot \mathbf{STDP}(\tau) \odot \mathcal{R}(t) - \boldsymbol{\Theta} \odot \text{sgn}(\mathbf{W}_k(t))] ,$$

governs the continuous-time local learning dynamics at the agent level. When $t \neq T_{cs}$, the Dirac delta function evaluates to zero, and the synaptic weights evolve according to reward-modulated STDP regulated by the RCSE mechanism. In this case, weight adaptation is driven by the interaction of spike timing and reward through $\mathbf{STDP}(\tau) \odot \mathcal{R}(t)$, while excessive growth is counteracted by the decay term $\boldsymbol{\Theta} \odot \text{sgn}(\mathbf{W}_k(t))$. The learning-rate matrix $\boldsymbol{\alpha}$ selectively enables or suppresses learning for each active synaptic set based on the competitive equilibrium defined by Ψ^S .

The second term,

$$\delta(t - T_{cs}) I_k^{\max} (\overline{\mathbf{W}}_G(t) - \overline{\mathbf{W}}_k(t)) ,$$

models the instantaneous federated synchronization event that occurs when the central server publishes a new global model at time T_{cs} . At this instant, the Dirac delta function becomes active, suppressing the local learning dynamics and applying a discrete correction that pulls the local synaptic weight matrix toward the global model.

The factor I_k^{max} appears because the global model $\overline{\mathbf{W}}_G(t)$ and the local model $\overline{\mathbf{W}}_k(t)$ are defined in normalized form within the interval $[-1, 1]$. Multiplying the difference by I_k^{max} rescales the federated update back to the physical synaptic weight domain of agent k , ensuring dimensional consistency with the continuous-time learning dynamics. This scaling guarantees that the magnitude of the federated correction is compatible with the agent's allowable synaptic weight range and preserves proportionality across agents with different maximum synaptic capacities.

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

Algorithm 2 High-Level Algorithm for the Proposed FL Algorithm

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

```

1: Initialize the agents and Central Server with default parameters for model publication
   threshold, Euclidean distance, and model publish status
2: Initialize the Global Model on the Central Server
3: if  $t$  is greater than 0 then
4:   Normalize synaptic weights of local models using (3.25)
5:   Aggregate models from all agents at the Central Server using (3.26)
6:   Calculate the  $\mathcal{D}_G$  between the current and previous global models on the Central
   Server using (3.29)
7:   if  $\mathcal{D}_G >$  the Central Server's threshold then
8:     Publish the global model
9:     set  $T_{cs} = t$ 
10:    end if
11:    for each Agent in the network do
12:      if Central Server publishes a new global model then
13:        Update the local model of the Agent with the global model using (3.30)
14:      else
15:        Agents evaluate their local models against the latest global model ( $\mathcal{D}_a$ ) using
           (3.28)
16:        if  $\mathcal{D}_a >$  the Agent's threshold then
17:          Send the model to the Central Server
18:          set  $T_k = t$ 
19:        end if
20:      end if
21:    end for
22:  end if

```

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

3.3 Results and Discussion

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

3.3.1 Simulation without FL

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

Table 3.1: Parameter values for LIF neuron model [86]

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 (section 3.2.3).

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

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

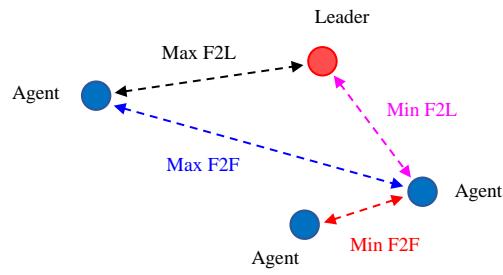


Figure 3.7: Measured distances used for evaluating swarm flight performance and collision detection.

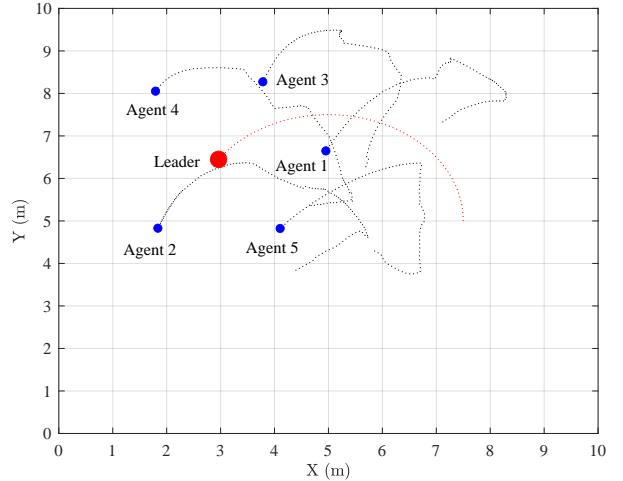


Figure 3.8: Agents' trajectory during the test phase

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

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

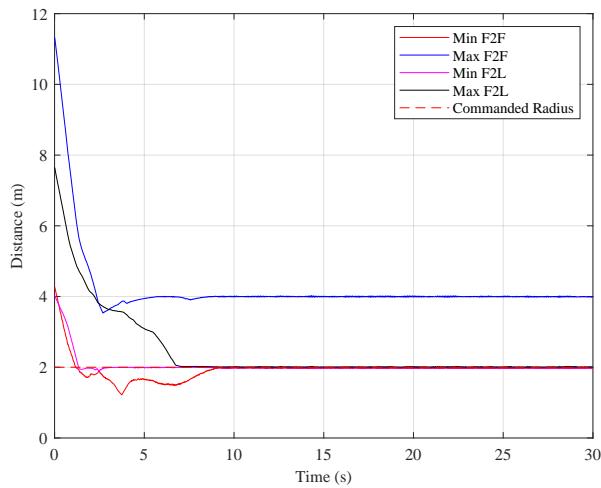


Figure 3.9: Variation of distances within the swarm during the test phase

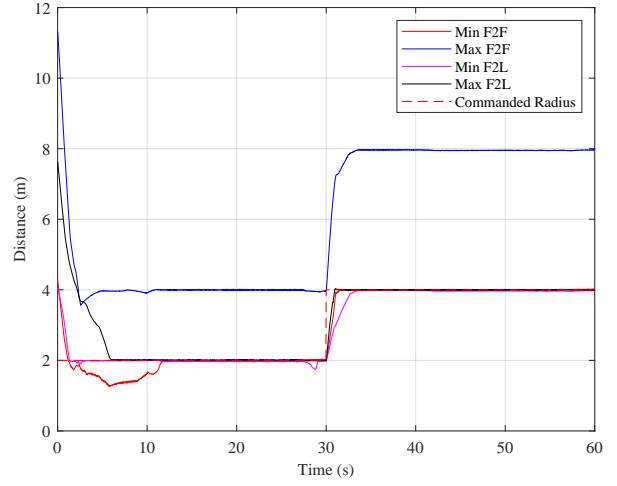


Figure 3.10: Adaptive response to commanded distance adjustments - reconfiguration during the test phase

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

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

$$\mathcal{R}_{Fi}^L(t) = -\mathcal{C}_F^L [r_{Fi}^L(t-1) - r_{Fi}^L(t)] \quad (3.31)$$

The original encoding function (3.1) for the input layer is recalled here for clarity:

$$\mu_I(\phi_i, r_i) = |\tanh(r - r_i)| \cdot \exp\left(-\frac{(\phi_i - \zeta)^2}{2\sigma^2}\right), \quad (3.32)$$

where the hyperbolic tangent term scales the angular membership by the magnitude of the distance error ($r - r_i$), and the Gaussian term encodes the LOS angle.

The $\tanh(\cdot)$ factor in (3.32) serves as an amplitude modulation that depends on the difference between the current distance and a commanded distance r . This term is meaningful only when a desired reference distance exists, as it maps the distance error into a bounded gain that increases with deviation from the command and vanishes when the error is zero.

When the leader is redefined as an obstacle, no commanded distance is associated with it. In this case, the distance variable r is undefined for the obstacle encoder, and the distance error term ($r - r_i$) cannot be formed. Mathematically, this removes the distance-dependent modulation from the encoding process. Consequently, the membership function reduces to its angular component, yielding

$$\mu_I(\phi_i) = \exp\left(-\frac{(\phi_i - \zeta)^2}{2\sigma^2}\right). \quad (3.33)$$

Equation (3.33) can therefore be interpreted as a special case of (3.32), corresponding to the absence of a distance reference. In this case, the encoding preserves directional (LOS) information while intentionally discarding distance-based scaling, which is consistent with obstacle avoidance behavior where only angular awareness, rather than distance regulation toward a commanded value, is required.

The simulation proceeded for an additional 1200 seconds, during which the synaptic weights were adjusted in accordance with the new reward function given by (3.31). In this case, because the leader is now an obstacle, so then $\alpha^S(t)$ in Figure 3.5 switches to 1 and weight adaptation starts again.

The results of the reward change (change in environment) are shown in Figures 3.11 and 3.12, which indicates that the agents quickly reduced their initial distance to the commanded

distance of 2 m. Simultaneously, the minimum distance from the obstacle, which was the leader, increased over time, indicating that the agents adapted their behavior to maintain a greater distance from the obstacle. Figure 3.11 shows the trajectory of each agent after the reward change.

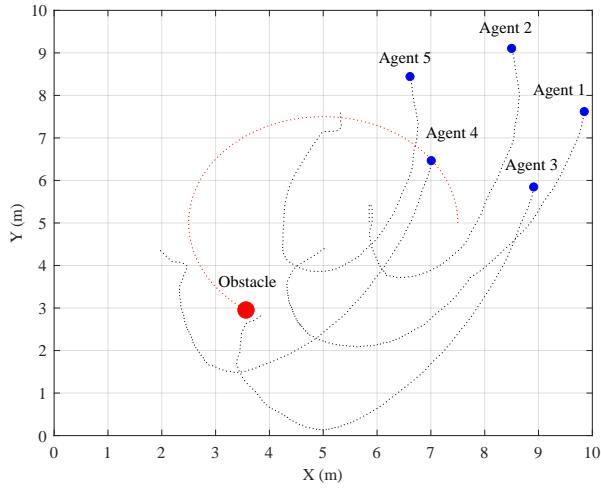


Figure 3.11: Trajectory adaptations of following agents in response to reward change for the leader during the test phase.

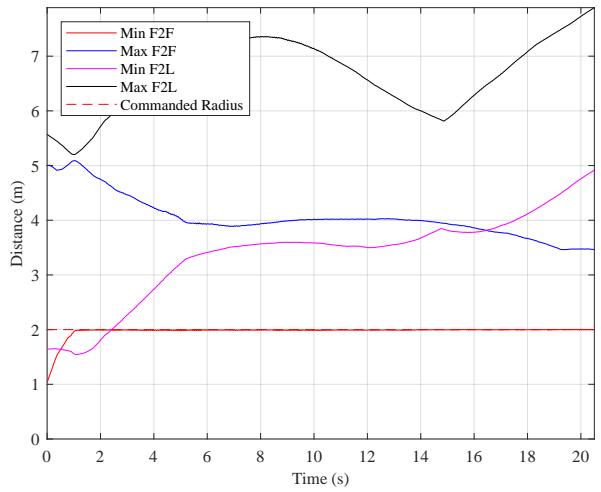


Figure 3.12: Variations in distances after reward changes and Leader becomes Obstacle - test phase.

Each agent in the swarm detects two neighboring agents and the leader, acquiring the Line-of-Sight angle and distance. Their neural network consists of three sub-layers in the input layer: two for neighboring followers (F_1 and F_2) and one for the leader (L), with inputs encoded using Gaussian Receptive Fields and fuzzy membership functions. The network uses the difference between the current and commanded distances within the swarm (r_{cmd}) and between each follower and the leader (R_{cmd}) as input to stimulate the input neurons. Each input channel is divided into two regions: those with distances greater than the commanded value and those within the commanded value. Within each region, the LOS angle is encoded using fuzzy membership functions. Errors in distances are transformed into amplitude values using the $tanh$ function, bounded between 0 and 1. Each agent's neural network includes 72

input neurons, with 24 neurons for each of the three objects that it detects. These objects are two neighboring agents and the leader. The output layer has 4 neurons representing positive and negative Δx and Δy movements.

In order to better understand the effect of reward change (change in environment) on the SNN, the synaptic weights matrix before and after reward change has been illustrated in Figures 3.13 and 3.14, the vertical axis shows the output neurons. The first output neuron is for negative displacement in the x-direction, while the second output neuron is dedicated to positive displacement in the x-direction. Similarly, the third output neuron corresponds to negative displacement in the y-direction and the fourth output neuron to positive displacement in the y-direction. The horizontal axis shows the input neurons. The neuron IDs from 1 to 24 are for the first sub-layer dedicated to the neighboring follower. The network has two sub-layers for the neighboring follower agents, but only one is shown since they are similar in the case of synaptic weight values. The neuron numbers from 25 to 48 are for the sub-layer dedicated to the leader. The RCSE method aims to maintain the synaptic weight matrix gradient while adapting to changes in the reward signal (Figures 3.13 and 3.14).

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

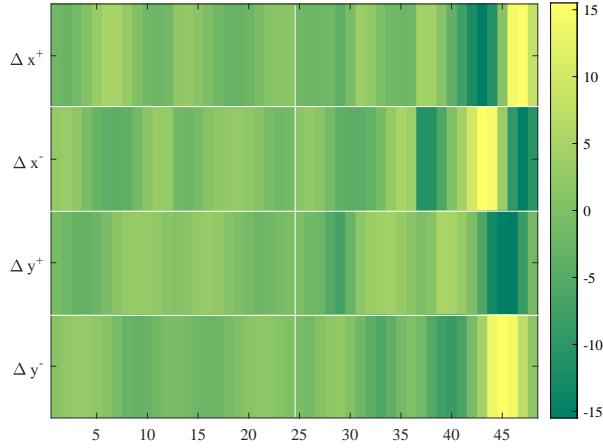


Figure 3.13: Synaptic Weights before Reward change in RCSE method.

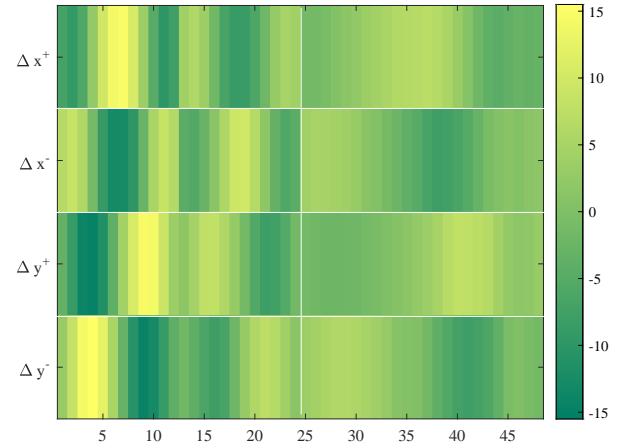


Figure 3.14: Synaptic Weights after Reward change in RCSE method.

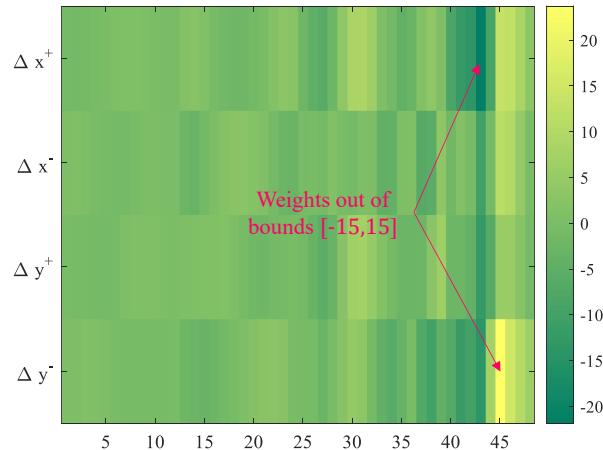


Figure 3.15: Synaptic Weights before Reward change in Multiplicative Synaptic Normalization method.

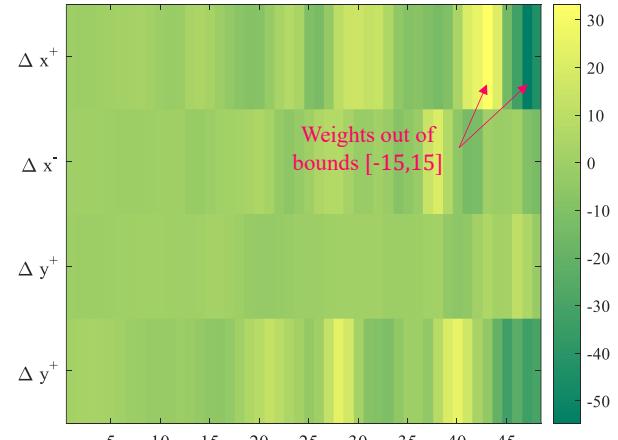


Figure 3.16: Synaptic Weights after Reward change in Multiplicative Synaptic Normalization method.

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

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

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

Figures 3.15 and 3.16 show the performance of the Multiplicative Synaptic Normalization (MSN) method. According to the results, this method cannot control the weight growth because of the coexistence of the excitatory and inhibitory (positive and negative synaptic weights). This method adjusts the weights when the summation of the input synaptic weights to an output neuron exceeds the maximum synaptic weight. In this case, the condition is never satisfied because positive and negative synaptic weights cancel each other, which is why we see synaptic weight values around 30 or -50 that are not within the I^{min} and I^{max} range, defined in (1.5) and (1.9). According to Figure 3.16, when the reward changes for the Leader, the MSN cannot adjust the weights because it does not have the reward-modulated decay rate mechanism.

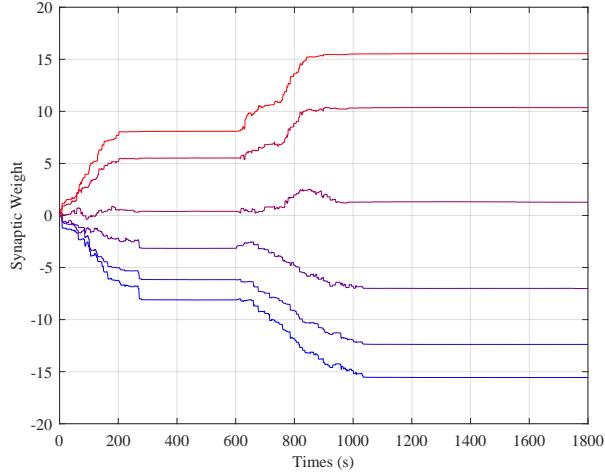


Figure 3.17: Synaptic weights increase after reward change in RCSE method.

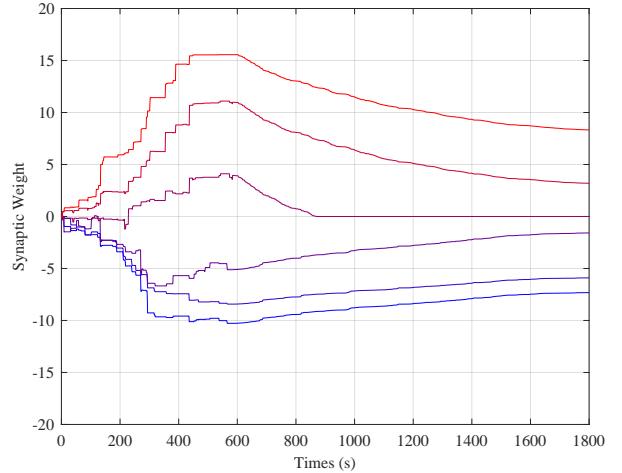


Figure 3.18: Synaptic weights decrease after reward change in RCSE method.

Figures 3.17 and 3.18 show the synaptic weights after the reward change (change in environment). In this case, since the reward coefficients are changed, the η and β values in (3.24) are changed for the represented sub-layers, and the proposed method has enabled the R-STDP algorithm to adjust the weights based on the new situation in the environment.

3.3.2 Simulation with Federated Learning and Reward-modulated Competitive Synaptic Equilibrium

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 ((3.28) and (3.29)) was 0.0005 and 0.00051, respectively. The reason for choosing the server's threshold higher than the agents' is that as soon as the first agent sends its model to the server, the Euclidean distance between the current and previously published model by the server reaches 0.0005, and the server distributes the model

immediately. Therefore, the serve's threshold is set higher than the agents' threshold, so it waits for the other agents to send their models.

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

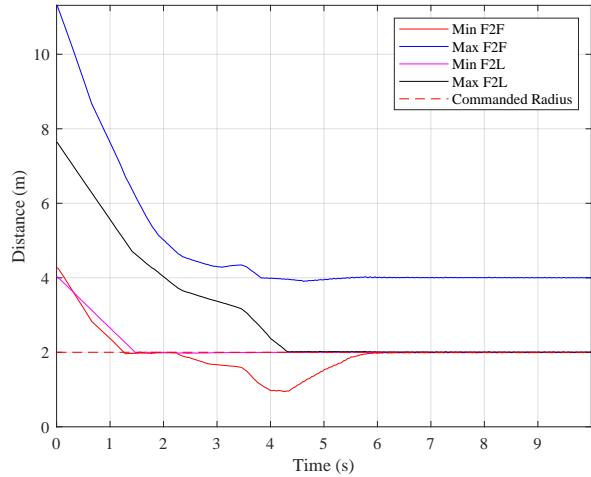


Figure 3.19: Distances during the test phase before reward change in the proposed event-triggered FL method.

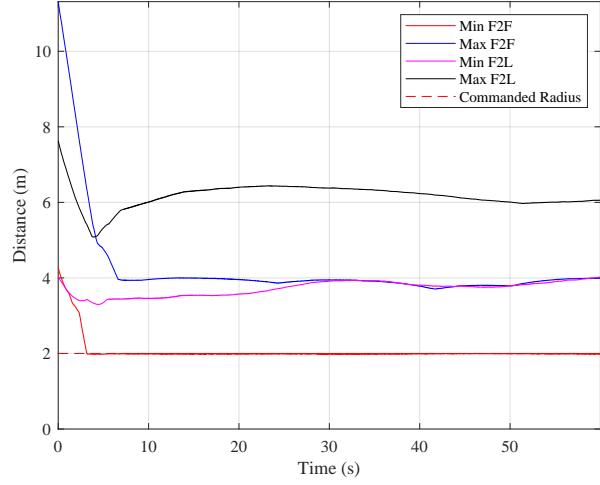


Figure 3.20: Distances during the test phase after reward change in the proposed event-triggered FL method.

The norm of the synaptic weights can represent the changes in the synaptic weights due to the change in the environment during the training process, which can be used to adjust the learning process in SNNs. In the proposed event-triggered FL method, agents communicate with the Central Server (leader) during training. According to Figure 3.22, the aggregation step time is small at the beginning of the training and increases as the SNN

models converge to the final solution. The rate of change of the norm of the synaptic weights (change in Frobenius norm showed in (3.27)) determines the communication sample time of the aggregation process, which results in small aggregation intervals when the change rate is high and larger intervals when it reduces.

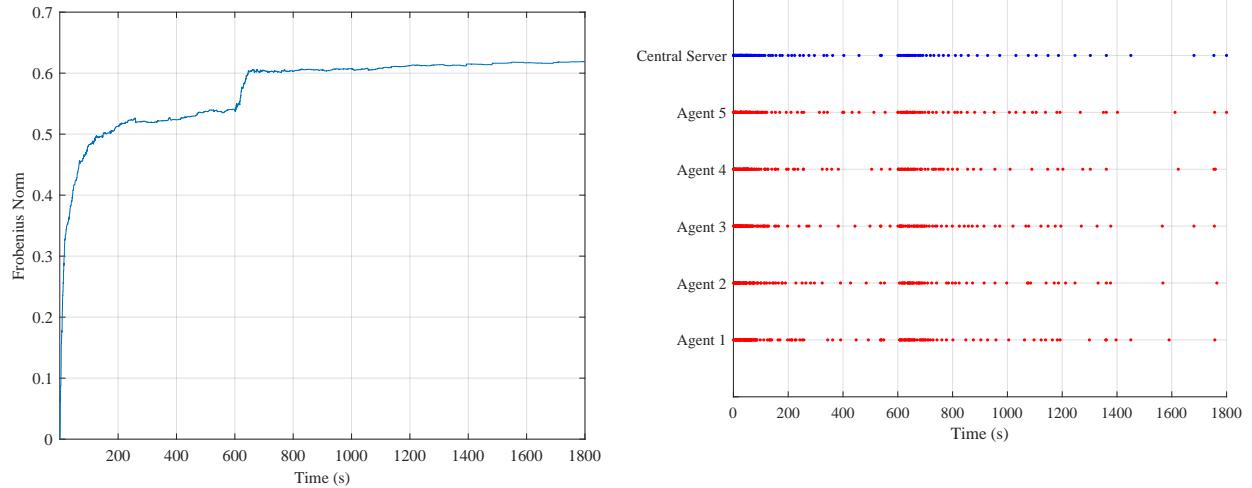


Figure 3.21: Frobenius norm of the Agent 1's weighs during the learning phase. The reward changes for the Leader after 600 s.

Initially, the aggregation frequency is high, but it decreases as the change in synaptic weights approaches zero. This reduction occurs because the learning rate matrix in (3.12) also approaches zero. The Sigmoid function is specifically designed to output 0 when the maximum synaptic weight exceeds the threshold and to output 1 when it is below the threshold. 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 (change in synaptic weights and the Frobenius norm shown in (3.27)), and the agents adjust the associated synaptic weight.

It should be noted that in the test phase, the synaptic weights oscillate around the

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

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

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

converged values. When they are lower than the converged value, the learning rate is 1; otherwise, it is 0. The decay rate pulls them away from the converged value, and the reward pulls them back toward it. The oscillation arises from the hard threshold mechanism, defined by the Sigmoid function. Therefore, if the reward changes (due to environmental changes), synaptic weights can adjust quickly because the learning rate returns to 1.

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

3.4 Conclusion

In this chapter, we presented a comprehensive approach addressing the challenges of uncontrolled growth in synaptic weights (shown in Figures 3.15 and 3.16) and the limited responsiveness of R-STDP to real-time changes within SNNs. Our proposed solution integrates the RCSE method with a dynamic aggregation interval in FL and significantly reduces learning time while improving performance.

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

weight saturation, the RCSE method facilitates a balanced approach to weight adjustment, preventing network saturation and promoting continuous learning adaptability.

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

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

Moreover, the dynamic aggregation interval effectively reduces communication overhead between the agents and the central server, particularly after model converges and the algorithm switches from the training mode to the operational mode. This reduction is critical when communication bandwidth is limited or costly. This approach is particularly advantageous in 5G networks, where efficient bandwidth use can enhance the overall throughput and reduce latency in real-time applications. Moreover, the adaptive use of communication resources aligns with the scalable and flexible infrastructure of 5G, optimizing network performance even during peak demand periods.

Chapter 4

Vision-Based Multi-Agent Docking with Deep Spiking Neural Networks

Autonomous docking of multiple agents in close-proximity missions, such as on-orbit servicing, satellite rendezvous, and swarm payload alignment, poses unique challenges in precision, robustness, and energy efficiency. In these missions, a team of robots must cooperate without centralized control, perceive relative pose information under stringent bandwidth and latency constraints, and execute coordinated maneuvers despite dynamic uncertainties and limited onboard resources. To address these challenges, this chapter introduces a fully decentralized, vision-based docking framework for swarm robots, combining DVS event streams, Active LED Markers (ALMs), and a Deep SNN trained with R-STDP. The sparse, asynchronous output of DVS cameras and the event-driven nature of spiking neural networks contribute to low-latency and energy-efficient control, which supports the development of responsive and resource-aware systems for proximity docking [94].

Prior approaches are often limited by their reliance on centralized coordination, dense frame-based vision pipelines (e.g., frame-based cameras), and rigid control architectures that struggle to scale to multi-agent docking under energy and bandwidth constraints. These limitations hinder their applicability to fully autonomous, decentralized swarm missions in dynamic proximity environments. Building on these insights and addressing the limitations

of prior methods, this chapter contributes the following advancements for reliable, efficient, and scalable autonomous docking in multi-agent proximity missions.

This work presents a biologically interpretable and scalable neuromorphic architecture that achieves end-to-end spiking control for cooperative docking through event-based perception and distributed learning. The main novelties of the proposed system are summarized as follows:

- 1. Entropy-based convolution kernel for DVS data:** A new convolution operator is introduced for processing ternary DVS events $\{-1, 0, +1\}$, where conventional kernels lose information due to polarity sparsity. The proposed variable entropy-based kernel computes spatial importance weights from local event entropy, enabling robust feature extraction from asynchronous visual inputs without frame reconstruction.
- 2. Short-term memory via bistable neurons:** A population of bistable Izhikevich neurons is designed to provide a dynamical short-term memory mechanism based on intrinsic oscillatory behavior. Rather than tracking the phase of an external periodic signal, each neuron exhibits an internal oscillatory attractor that can be activated by transient DVS events. Once the neuron transitions into this oscillatory state, it maintains sustained spiking activity even in the absence of further input. Event-triggered inhibitory signals act as a reset mechanism, returning the neuron to its resting equilibrium and terminating the oscillatory activity. This bistable excitation–inhibition structure enables short-term temporal memory on the order of tens of milliseconds, preserving recent event information while ensuring temporal continuity in event-driven perception when visual motion temporarily ceases.
- 3. Attention-guided object detection through phase-encoded ALM inhibition:** The first hidden layer includes neurons that emit phase-encoded inhibitory signals synchronized with a physical ALM. This mechanism uses ALMs to gate attention toward agents in the DVS field of view, suppressing background noise and enhancing agent detection in sparse event streams.

4. **Event-triggered policy switching for mission phases:** The second hidden layer incorporates specialized neuron repositories that trigger automatic transitions between mission phases. These transitions are governed by spiking events rather than explicit commands, enabling self-regulated and context-dependent policy switching.
5. **Multi-scale synaptic system for visual–motor coordination:** The architecture employs heterogeneous synaptic time constants ($\tau_C, \tau_{\text{STDP}}$) across layers, producing a temporal hierarchy where fast synapses capture rapid DVS dynamics and slow synapses stabilize actuator control. This multi-scale structure allows coherent perception–action coupling entirely within the spiking domain.

The remainder of this chapter is organized as follows. The Preliminaries section outlines the docking mission problem with swarm robots, describes the event-based camera simulation setup, details the dynamic models of the payload and agents, and explains the neuron model employed in our network. The Proposed Neuromorphic Framework section presents the architecture of the deep SNN, including the input layer design with entropy-based adaptive pooling, the encoding configuration, the hidden layers for visual processing and mission phase switching, and the output layer for PWM signal generation. This section also covers the training of the deep SNN using R-STDP, detailing the reward functions for different hidden layers and the synaptic weight update rules. The Results section evaluates the proposed framework, followed by the Conclusion.

4.1 Preliminaries

4.1.1 Problem definition: Docking Missions with Swarm Robots

Swarm robotics, inspired by the collective behaviors observed in natural systems such as ant colonies, bee swarms, and flocks of birds, represents a promising paradigm for creating intelligent robotic systems capable of performing complex, decentralized tasks. These systems are typically characterized by a number of relatively simple robots that collaborate

and coordinate their actions without relying on centralized control. This decentralized approach offers several advantages, including robustness to individual robot failures, flexibility in adapting to changing environments, and scalability to large numbers of agents [95].

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

In the specific application of this study, the swarm agents are tasked with pushing a payload towards a Soft Capture System (SCS) to achieve a precise docking alignment. Each agent is equipped with thrusters, allowing them to move in 2D space, and a DVS mounted on board provides them with visual feedback about the environment and proximal agents. Additionally, a DVS camera is mounted at the center of the SCS, providing a global event-based view of the payload and agents during docking.

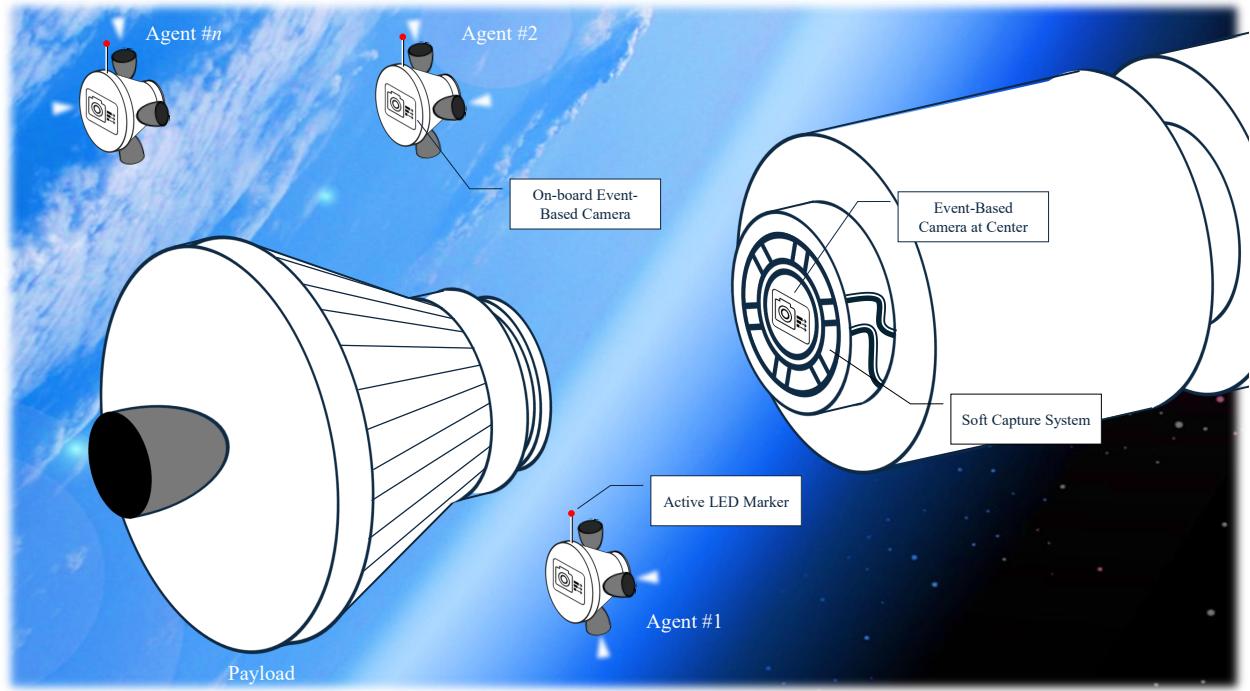


Figure 4.1: 3D schematic of the multi-agent docking mission environment. Multiple agents work collaboratively to push a central payload towards a SCS for docking alignment.

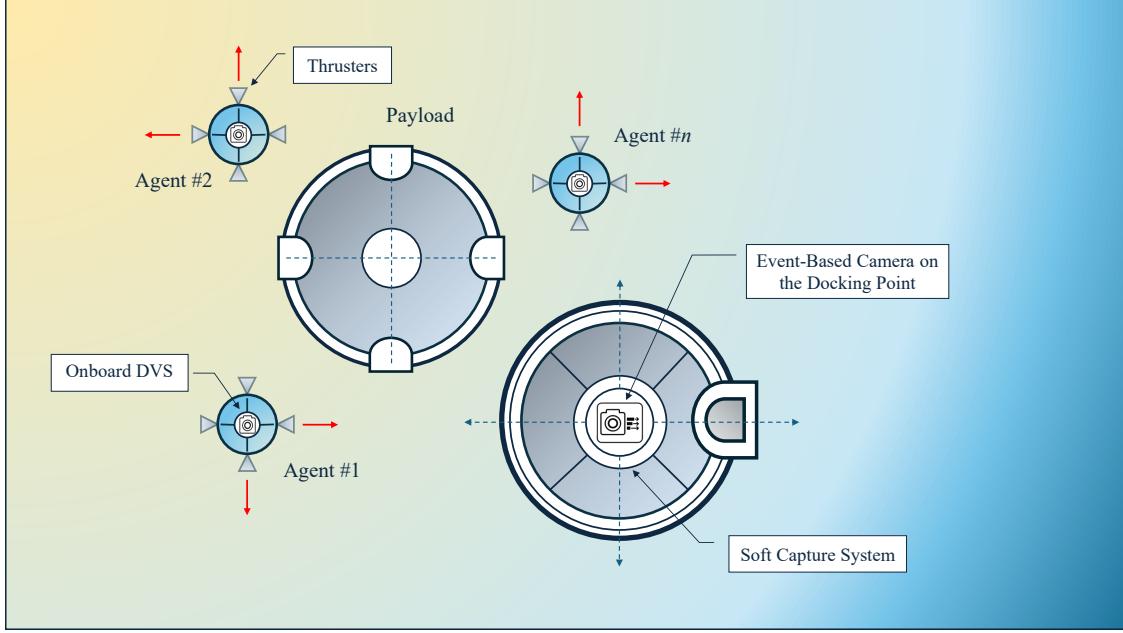


Figure 4.2: 2D representation of robots performing docking mission. Each agent pushes the payload towards the SCS to align them together. The SCS and agents are equipped with a DVS that streams high-frequency data for the proximity mission.

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

In this environment, ALMs are installed exclusively on the agents to enhance their visibility within the event-based vision system. Each agent only sees its neighboring agents via DVS events and does not directly measure their positions. The payload has a considerably larger surface area; as such, it naturally generates sufficient brightness changes to trigger events in the DVS sensor without the need for additional illumination. In contrast, the agents are compact and would otherwise produce sparse event activity; thus, their ALMs are used to amplify their visual signatures and ensure reliable detection by the vision system.

This configuration simplifies the temporal structure of the event stream while still enabling accurate agent identification and segmentation within the spatiotemporal data.

When the agents are doing rendezvous maneuver and have not made contact with the payload, they rely solely on their onboard DVS for environment perception. This allows them to capture events corresponding to nearby agents and the payload. Once an agent makes physical contact with the payload, its onboard force contact sensor activates and the ALM signal is turned off. The agents' onboard DVS data is then replaced by the SCS DVS, providing a more comprehensive and centralized perspective of the payload and the surrounding agents.

4.1.2 Event-Based Camera Simulation

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

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

For each DVS view, a brightness map is generated at every simulation time step. Let X_w and Y_w represent the 2D coordinate meshgrids over the DVS image plane in world units. The initial brightness grid $E(x, y)$ is set to a constant background value ζ_{bg} across all pixels. For each object in the scene, we determine which pixels of the DVS grid fall inside its circular outline. The brightness of these pixels is then calculated using Lambertian reflectance, which varies with the object's orientation and its distance to each light source.

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

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

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

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

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

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

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

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

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

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

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

Once the brightness map $E(x, y)$ is generated, it is passed to the event generation module

that models the behavior of the DVS sensor. To emulate the logarithmic response of biological photoreceptors, the brightness is first clamped from below to a minimum threshold ζ_{\min} (i.e., any value below ζ_{\min} is raised to ζ_{\min}) and transformed into logarithmic intensity [94]:

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

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

The temporal contrast is computed as the difference between the current and previously stored logarithmic intensity and integrated into a membrane potential variable $U(x, y)$ for each pixel [96]:

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

$$U_t(x, y) = U_{t-1}(x, y) + \Delta\mathcal{L}(x, y). \quad (4.10)$$

The membrane potential is bounded from below at $-\vartheta_{\text{on}}$ to prevent excessive accumulation of tiny negative values in dark regions, thereby improving numerical stability. In practice, a real DVS stops integrating once intensity becomes constant, but in simulation, small numerical biases can accumulate over time. To prevent this artificial drift from repeatedly triggering OFF events in static dark regions, we bound the potential to a lower limit instead of allowing it to integrate indefinitely.

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

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

Otherwise, no event is generated and $e(x, y) = 0$. These updates are performed for both agent-mounted and SCS-mounted DVS views independently, using their respective brightness maps and potential histories. The $\mathcal{L}_{\text{current}}$ map is stored as $\mathcal{L}_{\text{prev}}$ for the next time step. This detailed simulation allows each agent to perceive a localized view of the environment while

the SCS maintains a global perspective.

4.1.3 Dynamic Model of the Payload and Agents

Each agent in the swarm is modeled as a point mass of mass m and radius R . The dynamic state of each agent includes its position, velocity, thrust-based control forces, and interaction forces arising from *agent–agent* collisions. A central payload object with its own dynamics is influenced by contacts with the agents via a time-limited attachment rule (described below).

The state of agent i at time t is defined as $\mathbf{p}_i(t) = \begin{bmatrix} X_i(t) \\ Y_i(t) \end{bmatrix}$ and $\mathbf{V}_i(t) = \begin{bmatrix} V_{x_i}(t) \\ V_{y_i}(t) \end{bmatrix}$, and the control thrusts along the X and Y axes are derived from spike activity in the output layer of a SNN. Agent–agent interactions are modeled using a spring–damper system. Let agent a with position \mathbf{p}_a , velocity \mathbf{V}_a , and radius R_a interact with agent b with position \mathbf{p}_b , velocity \mathbf{V}_b , and radius R_b . The collision force is

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

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

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

$$\mathbf{F}_{ab}^{\text{coll}} = k \Delta_{ab} \mathbf{n}_{ab} + c \max(-(\mathbf{V}_{ab}^{\text{rel}} \cdot \mathbf{n}_{ab}), 0) \mathbf{n}_{ab}, \quad (4.15)$$

where k is the stiffness and $c = 2\zeta\sqrt{k m_{\text{eff}}}$ is a near-critically damped coefficient with reduced mass $m_{\text{eff}} = \frac{m_a m_b}{m_a + m_b}$ (damping acts only on approach). The force on a is $\mathbf{F}_{ab}^{\text{coll}}$ and on b is $-\mathbf{F}_{ab}^{\text{coll}}$.

The total force on a free agent a is the sum of its thrust and agent–agent collisions:

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

and its motion follows Newton's second law

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

Agent–payload interactions are not modeled via a spring–damper system; instead, first contact triggers a rigid attachment for a fixed duration $\tau_{\text{attach}} = 10$ s. Let $\mathbf{p}_P, \mathbf{V}_P, R_P, m_P$ be the payload pose, velocity, radius, and mass. Attachment is triggered when

$$\Delta_{iP}(t) = R_i + R_P - \|\mathbf{p}_i(t) - \mathbf{p}_P(t)\| > 0. \quad (4.18)$$

While attached, the agent's pose is locked to a fixed offset $\boldsymbol{\delta}_i$ on the payload surface and shares its velocity:

$$\mathbf{p}_i(t) = \mathbf{p}_P(t) + \boldsymbol{\delta}_i, \quad \mathbf{V}_i(t) = \mathbf{V}_P(t), \quad t \in [t_0, t_0 + \tau_{\text{attach}}]. \quad (4.19)$$

During attachment, the agent's thrust is routed to the payload, and the payload integrates with the *effective mass* of itself plus all currently attached agents:

$$\mathcal{Z}(t) := \{ i \mid i \text{ attached at } t \}, \quad (4.20)$$

$$m_{\text{eff}}(t) = m_P + \sum_{i \in \mathcal{Z}(t)} m_i, \quad (4.21)$$

$$\mathbf{F}_P^{\text{total}}(t) = \sum_{i \in \mathcal{Z}(t)} \mathbf{F}_i^{\text{thrust}}(t), \quad (4.22)$$

$$\frac{d^2 \mathbf{p}_P(t)}{dt^2} = \frac{\mathbf{F}_P^{\text{total}}(t)}{m_{\text{eff}}(t)}. \quad (4.23)$$

After τ_{attach} , the agent detaches and resumes free-agent dynamics.

These equations describe the continuous-time dynamics of the agents and the payload with agent–agent collision handling, attachment-mediated agent–payload coupling, thrust routing to the payload during attachment, and mass aggregation for the translational motion of the combined system (center-of-mass dynamics) over the attachment interval. Rotational

dynamics of the payload are neglected in this simplified model.

The agent actuation in the simulation follows a thruster model, in which each nozzle exhibits a Minimum–Impulse Bit (MIB) of 50 ms [97]. A MIB is the thruster’s minimum non-zero on-time (i.e., the smallest pulse duration the thruster can produce). Consequently, thrust updates are issued at a fixed actuation interval $T_{\text{mib}} = 50$ ms, while the physics are integrated with a smaller step $\Delta t > 0$. Let $k \in \mathbb{N}$ index the MIB windows and define $t \in [kT_{\text{mib}}, (k+1)T_{\text{mib}})$. The thrust decoded from the output layer of the SNN, $\mathbf{u}_{\text{snn}}(t) \in \mathbb{R}^2$, is zero-order-held over each window and quantized by the thruster on-time constraint:

$$\mathbf{u}_{\text{thr}}(t) = F_{\max} \mathbf{s}_k, \quad \mathbf{s}_k \in \{-1, 0, +1\}^2 \text{ (X- and Y-components)}, \quad t \in [kT_{\text{mib}}, (k+1)T_{\text{mib}}), \quad (4.24)$$

where the sign vector \mathbf{s}_k is inferred from the spike counts in the last decision window. For example,

$$s_{k,x} = \text{sign}(N_{x^-}(k) - N_{x^+}(k)), \quad s_{k,y} = \text{sign}(N_{y^-}(k) - N_{y^+}(k)), \quad \text{sign}(0) = 0.$$

Here $N_{x^-}(k)$ and $N_{x^+}(k)$ denote the spike counts of the output neurons encoding the negative and positive X thrust directions, respectively, during MIB window k (and analogously N_{y^-}, N_{y^+} for the Y axis). The translational dynamics of a free agent of mass m are

$$\dot{\mathbf{p}}(t) = \mathbf{V}(t), \quad m \dot{\mathbf{V}}(t) = \mathbf{u}_{\text{thr}}(t) + \mathbf{f}_{\text{coll}}(t), \quad (4.25)$$

where $\mathbf{f}_{\text{coll}}(t)$ denotes the spring–damper collision force modeled previously. The thruster MIB fixes the pulse width. As a result, the thrust command is applied in discrete time intervals, while the system dynamics remain continuous. Whenever $\mathbf{s}_k \neq \mathbf{0}$, a constant thrust of magnitude F_{\max} is held over the entire interval $[kT_{\text{mib}}, (k+1)T_{\text{mib}})$. Within that interval, no change in thrust direction or magnitude is possible.

4.1.4 Neuron model

Biologically inspired spiking neuron models often exhibit rich dynamical behaviors such as excitability, bursting, and bistability. The well-known Izhikevich model [17] strikes a balance between biophysical realism and computational efficiency, using only two ordinary differential equations plus a reset condition. The standard Izhikevich model is given by

$$\begin{cases} \frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I(t), \\ \frac{du}{dt} = a(bv - u), \end{cases} \quad (4.26)$$

with a reset rule:

$$\text{if } v \geq 30 \text{ mV, then } \begin{cases} v \leftarrow c, \\ u \leftarrow u + d. \end{cases} \quad (4.27)$$

where a and \mathbf{B} are the neuron parameters, v is the membrane potential (in mV), u is a slow recovery variable, $I(t)$ is the external input current, and c and d define how the system is reset after a spike.

4.2 Proposed Neuromorphic Framework

4.2.1 Deep SNN Structure

The neural network implemented in this study consists of four layers: an entropy-pooled input layer that processes sensory information from a DVS, the first hidden layer that learns visual perception and attention through ALM repositories (Payload Attention Inhibitor (PAI) and Neighboring Agent Attention Inhibitor (NAAI)) for Payload Attention Repository (PAR) and Neighboring Agent Attention Repository (NAAR), the second hidden layer that governs the mission phases through regulatory and control repositories (Docking Phase Inhibitor (DPI), Rendezvous Phase Inhibitor (RPI), Rendezvous Phase Repository (RPR), and Docking Phase Repository (DPR)), and an output layer responsible for generating motor

control commands. Each of these layers employs a distinct parameterization of the Izhikevich neuron model to achieve its respective function within the network. The term dominator refers to a dynamical winner-take-all condition enforced through lateral inhibition at the population level.

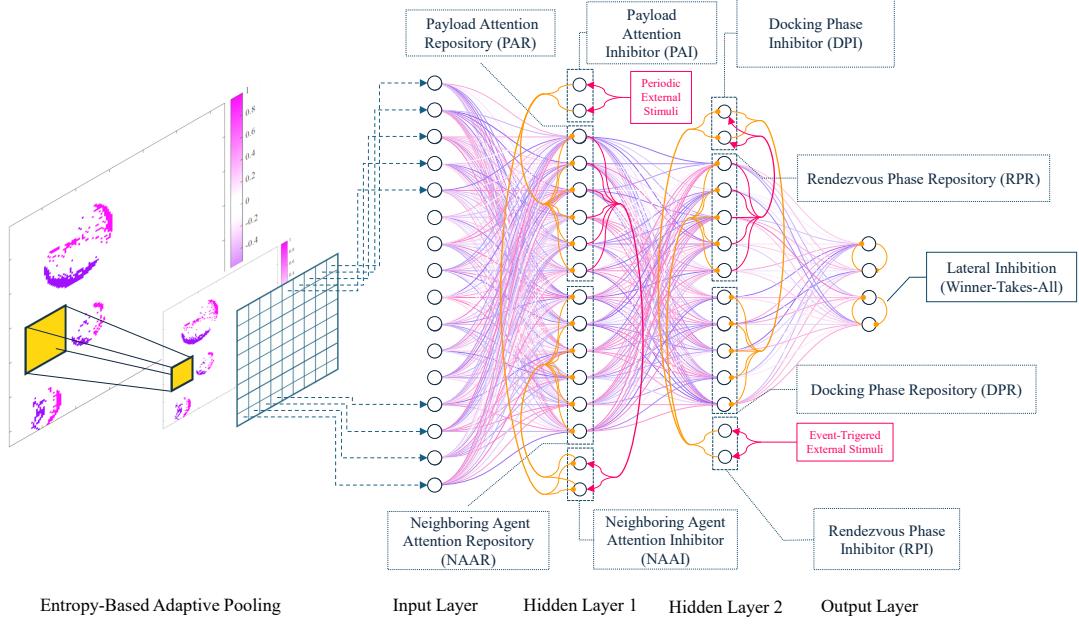


Figure 4.3: Deep SNN architecture with entropy-based adaptive pooling, multi-repository hidden layers, and biologically-inspired inhibitory and excitatory synapses. Purple, red, and blue arrows denote excitatory and yellow arrows denote inhibitory pathways.

In this study, we design a deep SNN architecture tailored for swarm robotics applications requiring perception, decision-making, and control under strict energy and computational constraints. The network consists of four main stages. The first is an entropy-based adaptive pooling input layer that compresses DVS event data into a 32×32 map while preserving temporal contrast information. This layer provides a compact and information-rich representation of dynamic visual stimuli, forming the input to subsequent processing layers.

The second stage is the first hidden layer, which is organized into multiple repositories of

neurons to separate and process attention-related signals. Specifically, it includes repositories for both payload and neighboring-agent attention (PAR and NAAR), as well as PAI and NAAI repositories that regulate the excitatory and inhibitory interactions between these attention pathways. This structure allows the network to perform selective attention and inhibition based on the relative activation of these neuron groups.

The third stage is the second hidden layer, which is divided into four functionally distinct repositories: the DPI, RPI, RPR, and DPR. These submodules are composed of bistable and Class 2 Izhikevich neurons that govern phase-gated transitions between rendezvous and docking behaviors, ensuring smooth and stable policy switching across mission stages.

Finally, the output layer consists of four spiking neurons corresponding to the thrust directions x^- , x^+ , y^- , and y^+ . These neurons generate directional control commands based on the collective spiking activity, which is refined through R-STDP. Each layer in the network employs layer-specific rewards to adapt its synaptic weights autonomously. This hierarchical organization enables the agents to perform phase-dependent policy switching, attention-driven visual perception, and decentralized coordination during both rendezvous and collaborative docking.

Input Layer

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

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

$$Pr(q) = \frac{1}{|\Omega|} \sum_{(i,j) \in \Omega} \mathbb{I}(E_{i,j} = q), \quad \text{for } q \in \{-1, 0, +1\}. \quad (4.28)$$

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

$$H(\Omega) = - \sum_{q \in \{-1, 0, +1\}} Pr(q) \log_2 Pr(q), \quad (4.29)$$

excluding terms where $Pr(q) = 0$. This entropy quantifies the diversity of event types in the region, with $H(\Omega) = 0$ indicating uniform activity and $H_{\max} = \log_2(3) \approx 1.585$ indicating maximum variability. The factor 3 in ($H_{\max} = \log_2(3)$) comes from the three possible event categories (-1,0,+1); the maximum entropy is reached when these categories are equally likely (probability (1/3) each, i.e., (1/3+1/3+1/3)).

The entropy value is then mapped to a pooling kernel size $\rho \in [\rho_{\min}, \rho_{\max}]$ using a linear transformation that adaptively controls the local compression ratio:

$$\rho(H) = \rho_{\max} - (\rho_{\max} - \rho_{\min}) \cdot \frac{H(\Omega)}{H_{\max}}. \quad (4.30)$$

Low-entropy (homogeneous) regions are pooled using a larger kernel ρ_{\max} , while high-entropy (heterogeneous) regions are pooled with finer granularity. Within each subregion, a sign-preserving selection is applied by retaining the event with the highest absolute intensity, maintaining polarity information. The resulting values form a downsampled event map $E' \in \{-1, 0, +1\}^{n' \times n'}$, typically $n' = 32$, which serves as the input to the bistable neurons of the first hidden layer.

Bistability-based input encoding and inhibitory calibration

The input layer is composed of bistable spiking neurons implemented using the Izhikevich model in a parameter regime where a stable resting equilibrium coexists with a self-sustained spiking attractor [98]. This bistability enables each neuron to retain a memory of recent excitation without requiring recurrent connections. An excitatory input event can switch the neuron from rest into the spiking attractor, where it remains active until an explicit

inhibitory input returns it to quiescence. In this sense, the neuron maintains an internal dynamical state that persists after the removal of the input, providing short-term temporal memory of event polarity.

At steady state, setting $\dot{v} = \dot{u} = 0$ in the Izhikevich model yields

$$u = bv, \quad (4.31)$$

$$0.04v^2 + (5 - b)v + (140 + I) = 0, \quad (4.32)$$

with equilibrium points determined by the discriminant

$$\mathbf{D} = (5 - b)^2 - 4(0.04)(140 + I). \quad (4.33)$$

When $\mathbf{D} > 0$, the system admits two real equilibria corresponding to a stable resting state and an unstable equilibrium associated with the onset of a spiking limit cycle, defining the bistable regime. The critical inhibitory current required to collapse the oscillatory attractor is obtained by enforcing tangency of the v -nullcline,

$$\frac{\partial}{\partial v}(0.04v^2 + 5v + 140 - u + I) = 0 \Rightarrow v^* = -62.5, \quad u^* = b v^* = -93.75, \quad (4.34)$$

which yields $I_{\text{crit}} = -77.5$. Based on this analysis, each input neuron receives current according to the preprocessed DVS event E'_i ,

$$I_i(t) = \begin{cases} 100, & E'_i = +1, \\ -77.5, & E'_i = -1, \\ -65, & \text{otherwise,} \end{cases} \quad (4.35)$$

where excitatory ON events induce a transition into the spiking attractor, OFF events actively reset the neuron to rest, and the baseline current maintains quiescence. This bistability-based encoding allows recently active visual features to remain represented for

tens of milliseconds even in the absence of new events, ensuring temporal continuity in sparse, event-driven perception.

Hidden Layers

First Hidden Layer: Attention and Perception of DVS Events

The first hidden layer of the SNN is designed to perform early-stage visual feature extraction, dimensionality reduction, and activity-coded spatial representation of the environment while incorporating ALM-based attention and gating mechanisms. This layer contains four repositories of neurons that collectively process the DVS input to distinguish between the payload and proximal agents while encoding these features in a activity-coded spatial representation guided by reward function (section 4.2.2).

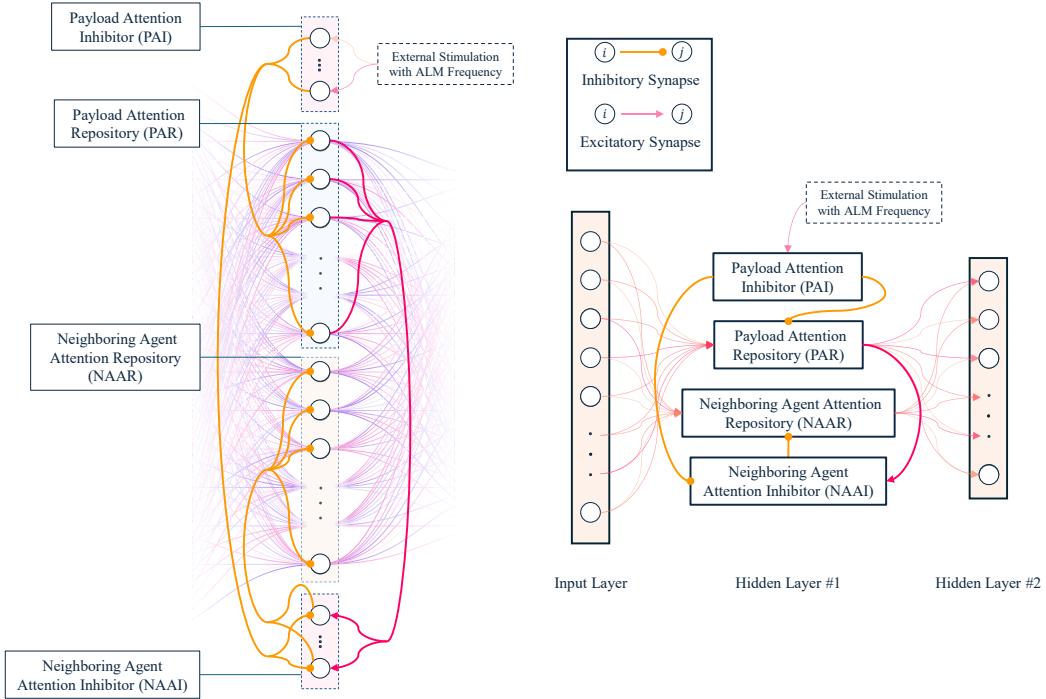


Figure 4.4: Structured First Hidden Layer Architecture with payload, and Agent Perception Repositories for Visual Perception.

All neurons in this layer employ the Izhikevich model with Class 2 excitability for smooth frequency modulation, while the NAAI neurons specifically use a bistable configuration to maintain persistent spiking during activation intervals. This bistability enables autonomous inhibition without continuous external input: once NAAI neurons are activated, they continue firing until the inhibitory reset signal is received. Each bistable neuron is initialized with consistent membrane and recovery variables to guarantee readiness for event-triggered activation. The ALMs are mounted on each agent transmit light periodically at a frequency designed for phase-shifted activation patterns. The ALMs are configured to blink with a period 20 ms, remaining active for 10 ms.

Functionally, the first hidden layer separates the payload position channel and the proximal agents' position channel, representing them in a activity-coded spatial representation (firing rates indicate the degree of activity associated with that spatial region) using the spiking activity of its neurons. This activity-coded spatial representation is achieved by mapping the processed DVS input onto the neurons while allowing the network to maintain partial activation across spatial regions, enabling smooth transitions between detected object configurations. Additionally, the first hidden layer performs dimensionality reduction by projecting the high-resolution DVS input onto two separate repositories (one for the payload and one for proximal agents) each represented on an 16×16 grid.

The spiking patterns generated within this layer create unique states in the network, allowing the system to distinguish different object configurations in the environment. During operation, when the agents' ALMs are active, the corresponding ALM neurons in the first hidden layer fire and inhibit the payload detection repository. This inhibition ensures that during the weight update phase of the network, only the connections associated with the agent detection repository are updated. When the agents' ALMs turn off, the payload detection neurons fire, and the connections associated with the payload detection repository are updated in the subsequent weight update phase.

Through this mechanism, the first hidden layer dynamically manages the flow of information based on the ALM signals, ensuring clear separation between agent and payload

representations, maintaining activity-coded spatial representation, reducing input dimensionality, and creating structured, distinguishable states for downstream processing in the SNN.

Second Hidden Layer: Mission Phase Control and Synaptic Pathway Switching

The second hidden layer governs mission-phase control and synaptic pathway switching by utilizing structured neuron repositories that enable the network to autonomously transition between rendezvous and docking behaviors while learning phase-dependent control policies through spiking activity driven by external sensory feedback.

Within this layer, the Rendezvous Phase Repository (RPR), Docking Phase Repository (DPR), Docking Phase Inhibitor (DPI), and Rendezvous Phase Inhibitor (RPI) repositories are interconnected to enforce phase-gated learning. The RPR and DPR repositories act as primary output conduits through which synaptic connections to the motor control layer are routed, enabling phase-specific R-STDP-based updates only in the currently active pathway. Inactive pathways remain unchanged, preserving previously learned control policies. The RPI and DPI repositories regulate these transitions through event-based mutual inhibition and excitatory control.

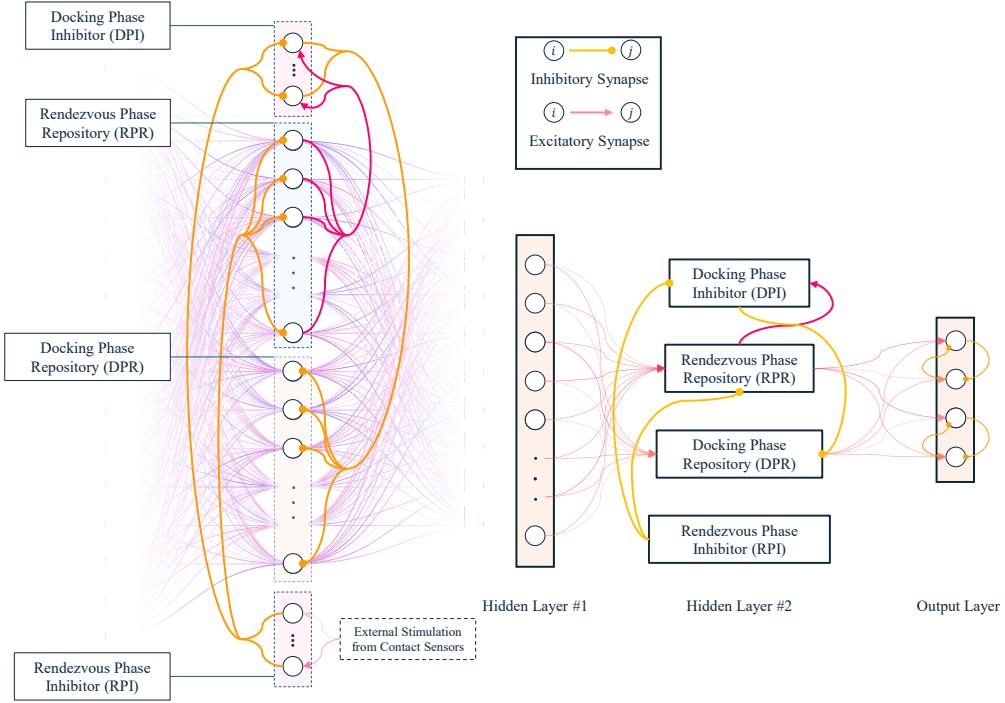


Figure 4.5: Structured Second Hidden Layer architecture showing the Rendezvous Phase Repository, Docking Phase Repository, Rendezvous Phase Inhibitor, and Docking Phase Inhibitor repositories interconnected through inhibitory (yellow) and excitatory (purple) synapses. External stimulation from contact sensors triggers mission-phase transitions.

The neurons within the second hidden layer employ models capable of producing persistent spiking according to the input (Class 2 model), and rapid state transitions required for dynamic mission control. The RPR and DPR neurons are configured to exhibit regular, sustained spiking, enabling them to produce stable outputs during their respective active phases. In contrast, the DPI neurons exhibit bistable firing patterns, enabling them to maintain persistent activity while providing controlled inhibition during phase transitions. The RPI neurons use a bistable neuron model, designed to generate rapid and transient spiking patterns that effectively trigger phase switches by inhibiting the active repositories when mission conditions change.

During rendezvous phase, the agent has not contacted the payload, the RPR neurons

maintain persistent spiking activity. In this phase, the active learning pathway extends from the input layer to the visual processing layer, through the RPR, to the output layer, where synaptic connections within the RPR repository are updated according to the R-STDP rule. Upon payload contact, the RPI neurons are activated, signaling the transition to the docking phase. The RPI neurons inhibit both RPR and DPI neurons. This inhibition silences RPR and DPI activity, releasing the DPR neurons from inhibition and allowing them to fire, thereby enabling the network to transition to the docking phase. During docking, the active learning pathway becomes the input layer to the visual processing layer, to the DPR, and continues to the output layer, allowing the network to learn the docking policy explicitly through the active DPR connections.

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

Output layer, lateral inhibition, and MIB-synchronized learning dynamics

The output layer of the SNN consists of four “bistable” Izhikevich neurons. These neurons correspond to the four directional thrust commands: negative X-axis, positive X-axis, negative Y-axis, and positive Y-axis, indexed respectively as $\{X^-, X^+, Y^-, Y^+\}$. Each neuron receives synaptic input from the preceding hidden layer and produces spike trains, while the effective actuation and reward update interval are synchronized with the MIB of the thruster, $T_{mib} = 50$ ms.

To ensure that only one neuron within each axis dominates during a 50 ms actuation window, a pairwise lateral inhibition mechanism is introduced between opposing directional neurons ($X^- - X^+$ and $Y^- - Y^+$). When one neuron in a pair fires, it instantaneously suppresses its counterpart via a strong inhibitory synapse, effectively enforcing a winner-take-all behavior along each control axis. This mechanism ensures that the neuron representing the

intended direction fires at a significantly higher rate, while its opposite neuron produces only a few residual spikes.

Such lateral inhibition serves two essential functions. First, it sharpens directional selectivity, guaranteeing that only one thrust command per axis is active at any given time, preventing thrust conflicts. Second, it improves the fidelity of reward assignment under R-STDP. Without this inhibition, both neurons in a pair might fire at comparable rates, causing the reward signal to propagate to the non-contributing neuron. With lateral inhibition, the suppressed neuron produces very few spikes during the MIB window and therefore receives negligible eligibility trace updates, ensuring that the reward is applied only to the neuron responsible for the chosen action. Figure 4.6 shows how lateral inhibition is implemented between opposing direction neurons in the output layer to enforce winner-take-all dynamics within each control axis. When X^+ fires, it inhibits X^- , and vice versa; similarly for the Y axis neurons.

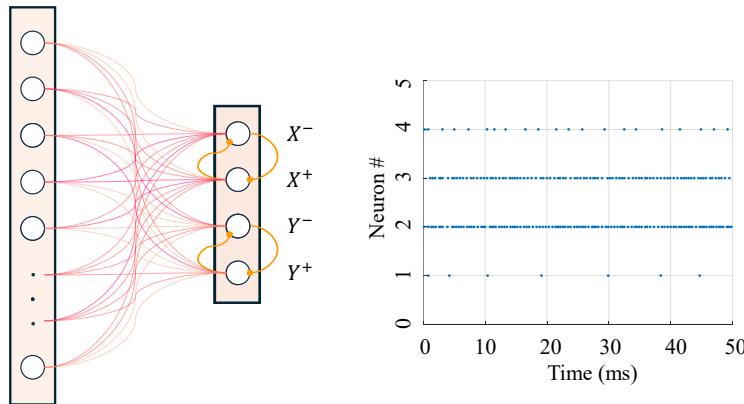


Figure 4.6: Output Layer Architecture with Lateral Inhibition between Opposing Direction Neurons to Enforce Winner-Take-All Dynamics within Each Control Axis.

A distinctive property of the output layer compared with the upstream layers is the adaptation of its synaptic time constants to match the actuation cycle of the physical system. The eligibility trace $\mathcal{C}(t)$, which integrates pre-post spike correlations prior to reward arrival, follows an exponential decay whose time constant is set to $\tau_c = 100$ ms. This value corresponds to twice the MIB duration, ensuring that the effect of an action persists long

enough for its thrust consequence to manifest before the reward is delivered.

The reward is computed every $T_{\text{mib}} = 50$ ms, synchronously with the control update interval. Since the physical effect of an action is evaluated one cycle later (approximately 100 ms after the initiating spike), the R-STDP rule correctly associates each synaptic modification with its corresponding behavioral outcome. This MIB-synchronized design enables each layer to operate at its appropriate temporal resolution that is shorter (e.g., 10 ms) for sensory and visual processing layers that handle high-rate DVS events, and longer (e.g., 50–100 ms) for the thrust layers constrained by actuation dynamics.

At each control window, the total number of spikes emitted by each output neuron within the 50 ms interval is accumulated to form the PWM duty ratio:

$$\text{duty}_i = \frac{1}{T_{\text{mib}}} \sum_{t \in [kT_{\text{mib}}, (k+1)T_{\text{mib}})} f_i(t), \quad (4.36)$$

where $f_i(t) \in \{0, 1\}$ denotes the instantaneous spiking indicator. Although the thrust decoding mechanism resembles pulse-width modulation (PWM), it is quantized to discrete actuation windows synchronized with the thruster’s minimum impulse bit ($T_{\text{mib}} = 50$ ms). In this formulation, each window acts as the effective PWM sample time, within which the neuron’s firing activity determines the commanded thrust direction and magnitude. The resulting firing ratios are then mapped to the corresponding thrust directions:

$$F_X^{\text{thrust}} = F_{\max} \text{ sign}(\text{duty}_{X+} - \text{duty}_{X-}), \quad (4.37)$$

$$F_Y^{\text{thrust}} = F_{\max} \text{ sign}(\text{duty}_{Y+} - \text{duty}_{Y-}), \quad (4.38)$$

where F_{\max} is the maximum available thrust. Although lateral inhibition enforces a winner-take-all regime, inhibition is not ideal; the suppressed neuron may emit sparse residual spikes within a control window, and the thrust direction is therefore robustly determined by the sign of the dominant neuron’s firing excess.

Each neuron’s activity thus directly determines the discrete thrust command applied to the corresponding micro-thruster, while the difference between opposing neurons defines the

net thrust along each axis. If both neurons in a pair fire simultaneously, their contributions cancel, producing zero net force and preventing control conflicts (simultaneous activity of thrusters in opposite directions).

4.2.2 Training Deep SNN with R-STDP

Reward Function for the First Hidden Layer

The reward function in the SNN provides a structured feedback mechanism that directly guides synaptic plasticity. Its role is to compare the firing rates of the hidden layer with the event-based sensory input and to generate a modulation signal that drives weight updates. In this way, the reward function ensures that the hidden layer gradually aligns its perception (activity patterns) with the external sensory events. At each reward computation step, the input layer activity is calculated as the time-averaged firing activity:

$$\mathbf{A}^{\text{input}}(i) = \frac{1}{T} \sum_{\tau=t-T}^t f_i^{\text{input}}(\tau), \quad (4.39)$$

where $f_i^{\text{input}}(\tau) \in \{0, 1\}$ indicates whether neuron i generated a spike at time τ . Each entry $\mathbf{A}^{\text{input}}(i)$ is the time-averaged firing rate of neuron i over the window (T). For example, the fraction of time steps in that window when neuron i fired; the matrix collects these per-neuron values. Specifically,

$$f_i(\tau) = \begin{cases} 1, & \text{if } v_i(\tau) \geq v_{\text{th}}, \\ 0, & \text{otherwise,} \end{cases} \quad (4.40)$$

where $v_i(\tau)$ is the membrane potential of neuron i at time τ and v_{th} is the firing threshold. Figure 4.7 illustrates how the activity matrix is computed from the time-averaged spiking activity of a layer n . Each element represents the fraction of time steps in which neuron i fired within the update window T .

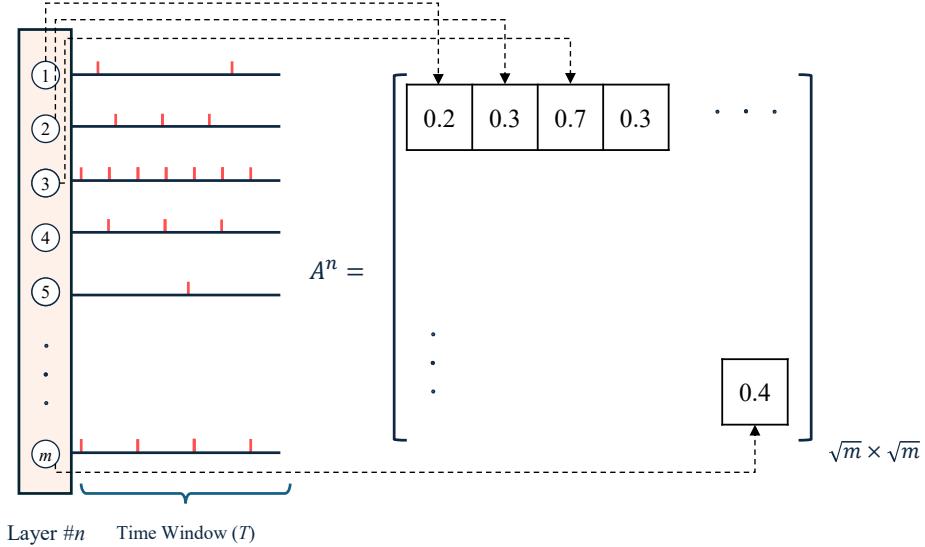


Figure 4.7: Computation of the activity matrix (A) from the time-averaged spiking activity.

Similarly to Figure 4.7, for the first hidden layer \mathbf{A}^{H1} , the payload detector and agent detector repositories compute:

$$\mathbf{A}^{H1}(i) = \frac{1}{T} \sum_{\tau=t-T}^t f_i^{H1}(\tau). \quad (4.41)$$

The activity matrix shown in Figure 4.7 is calculated separately for each repository (one matrix for the payload detector and one for the agent detector based on their number of neurons).

To compute the attention map, the input layer activity matrix $\mathbf{A}^{\text{input}}$ is compared with the positions represented in the first hidden layer using a Gaussian-weighted mapping. Each neuron is permanently assigned to one grid cell. Therefore, every neuron has an associated spatial coordinate that is defined by its index. For a first hidden-layer neuron indexed by q , whose receptive-field center is located at (x_q, y_q) , and the input-layer indexed by k , located at (x_k, y_k) , the squared Euclidean distance is given by:

$$d_{kq}^2 = (x_k - x_q)^2 + (y_k - y_q)^2. \quad (4.42)$$

Here, x_k and y_k are not the physical positions estimated by the agents. Instead, they are fixed, normalized coordinates based on the grid indices of the input and first hidden layer activity maps. These coordinates are static, predefined, and known in advance from the network topology. The attention weight assigned is:

$$\gamma_{kq} = \exp\left(-\frac{d_{kq}^2}{2\sigma_A^2}\right), \quad (4.43)$$

where $\sigma_A = [0.05, 0.1]$ is a scale parameter controlling the sensitivity. The coefficient γ_{kq} is a Gaussian spatial weighting term that quantifies how strongly the activity of input neuron k influences hidden-layer neuron q based on their relative positions in the fixed activity grids, enforcing spatial locality such that each hidden neuron primarily attends to nearby input layer activity while smoothly attenuating contributions from distant regions.

The modulation signal $r^{H1}(q)$, which serves as the reward for the first hidden layer neuron q , is computed directly by comparing the attention-weighted input layer activity with the current hidden layer activity:

$$\mathcal{A}^{\text{Attention}}(q) = \frac{\sum_k \gamma_{kq} \cdot \mathbf{A}^{\text{input}}(k)}{\sum_{k=1}^K \gamma_{kq} + \epsilon}, \quad (4.44)$$

$$r^{H1}(q) = \mathbf{A}^{H1}(q) \times (\mathcal{A}^{\text{Attention}}(q) - \mathbf{A}^{H1}(q)), \quad (4.45)$$

This formulation emphasizes high-activity regions aligned with the ALM areas and directly provides the per-neuron learning signal. When the hidden layer activity \mathbf{A}^{H1} matches the attention-driven signal, the $r^{H1}(q) \rightarrow 0$, stopping the weight updates for that neuron during the iteration. The reward $r^{H1}(q)$ reinforces first hidden-layer neurons whose activity is consistent with the currently attended object, such that when the ALM is on the attention is concentrated on neighboring agent regions, and when the ALM is off the higher event density generated by the payload shifts attention toward payload-related regions, enabling dynamic, attention-driven specialization of spatial representations.

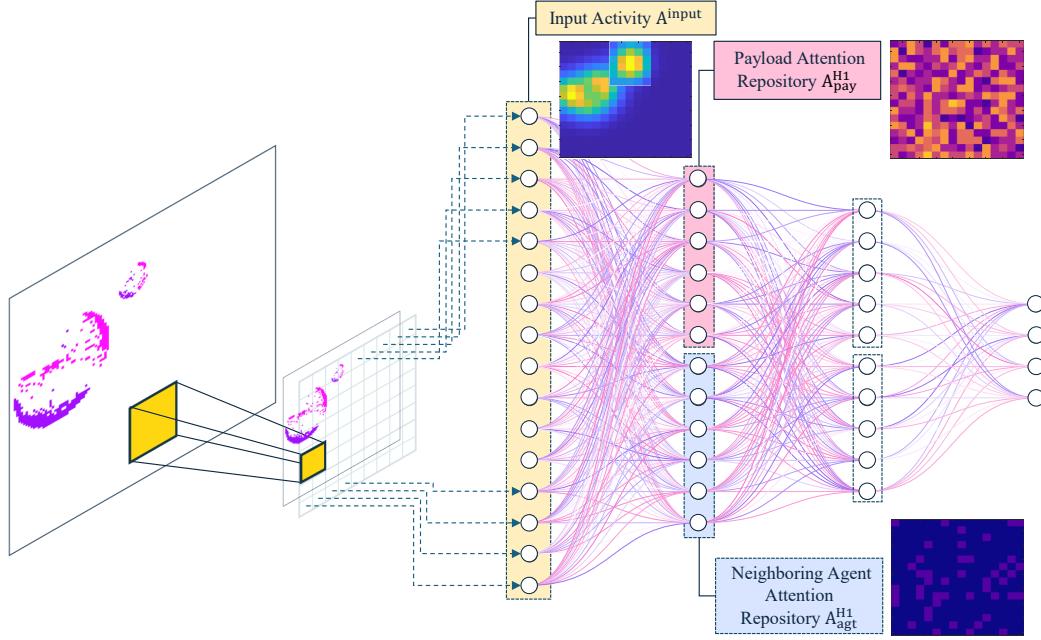


Figure 4.8: Input layer activity ($\mathbf{A}^{\text{input}}$), Payload Attention Activity ($\mathbf{A}_{\text{pay}}^{\text{H1}}$), and Neighboring Agent Attention Repository ($\mathbf{A}_{\text{agt}}^{\text{H1}}$) computation pipeline when the agent’s ALM is turned off and most of the events are generated by the payload because of its larger weight in comparison with the agent. This feature puts more attention on the payload position.

Figure 4.8 illustrates the computation pipeline for the activity matrices in the input layer and first hidden layer when the agent’s ALM is turned off, focusing on payload events.

According to Figure 4.9, at each time step, the activity of both the payload detector neurons and the agent detector neurons is compared with the attention map. A lateral inhibition mechanism ensures that only one repository is active at a time. As a result, the inhibited neurons do not generate spikes, and there is no active synaptic path from the input neurons to these inhibited neurons during that interval. Consequently, according to (4.45), the reward signal for any inhibited repository is effectively multiplied by the zero activity of that repository ($\mathbf{A}^{\text{H1}} = 0$), and no weight updates occur for those synapses. This selective gating ensures that only the active repository receives a weight update.

Figure 4.10 shows how the reward function pipeline works for the agent detection process in the first hidden layer. The reward signal provides the value of $r^{\text{H1}}(q)$ for each neuron,

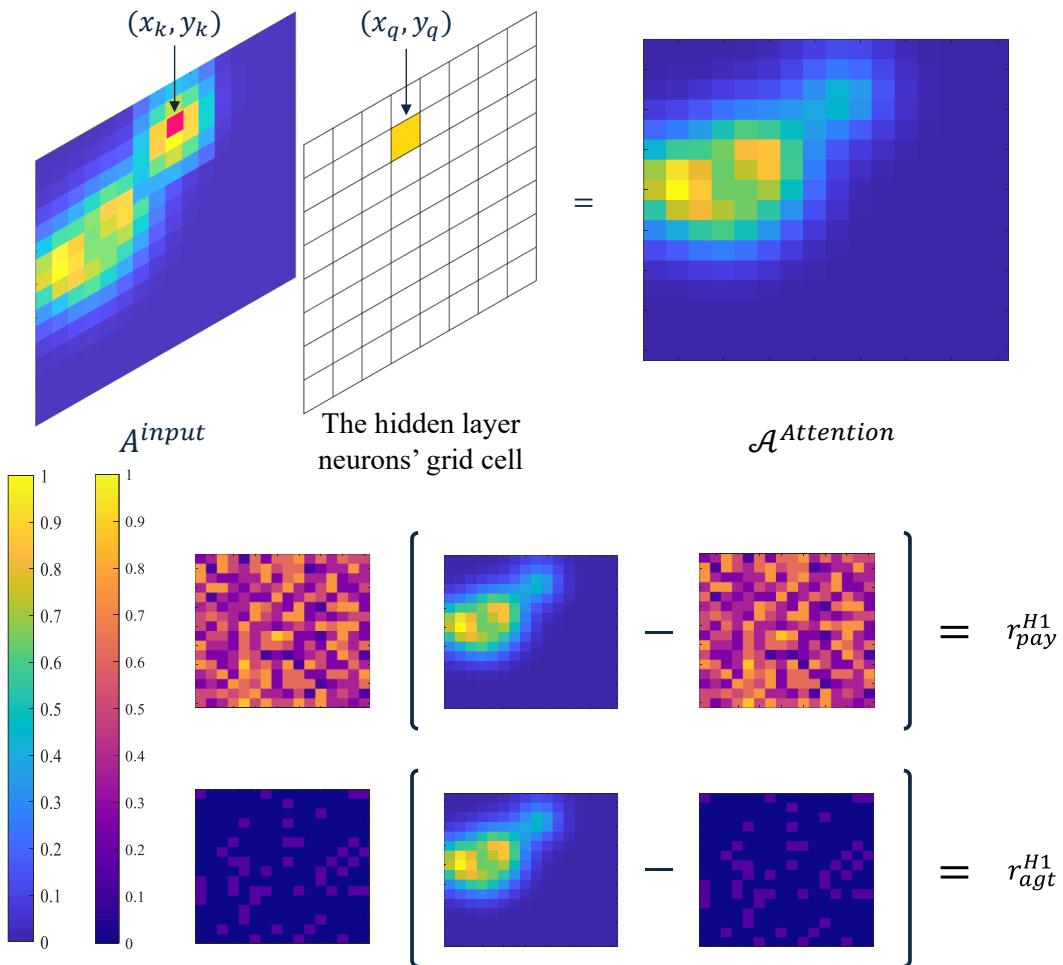


Figure 4.9: Computation of the Attention Mechanism for Reward Generation in the First Hidden Layer. The attention map ($\mathcal{A}^{Attention}$) is computed by comparing the input layer activity (A^{input}) with the spatial positions of the first hidden layer neurons using a Gaussian-weighted mapping. The resulting attention map is then used to calculate the reward signal (r^{H1}) for each neuron in the first hidden layer by comparing it with the hidden layer activity (\mathbf{A}^{H1}).

corresponding to each pixel in the representation.

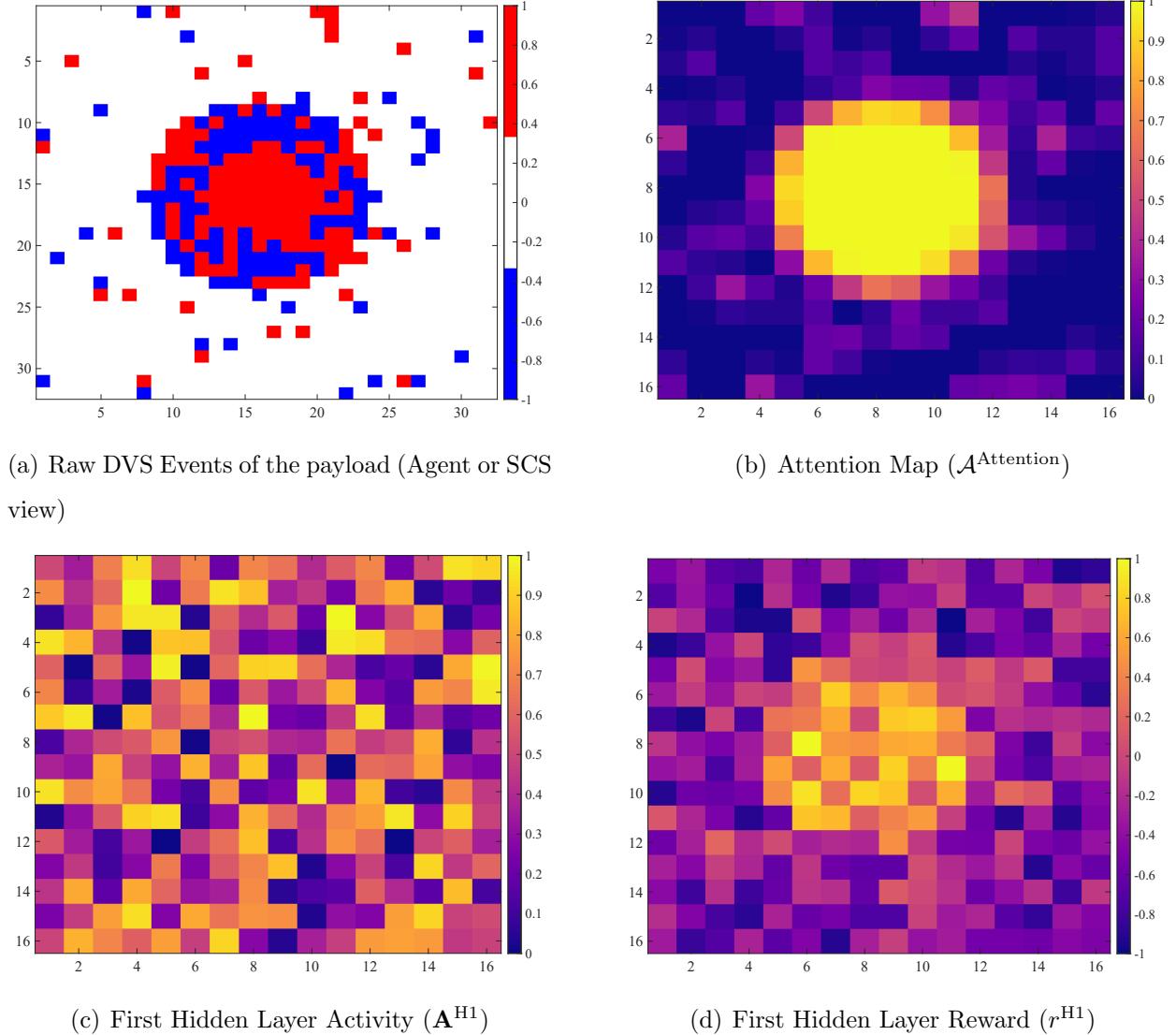


Figure 4.10: Reward function pipeline for the payload. The pipeline illustrates the raw DVS events, the attention-based map, the normalized hidden layer activity during the initial training phase, and the reward signal calculated as the difference between the attention map and the hidden layer activity ($r^{H1}(q)$).

Each grid cell in Figure 4.10(d) shows the reward value assigned to the corresponding neuron in the payload detector repository of the first hidden layer. The spatial arrangement of these neurons follows the same layout as illustrated in Figure 4.7, where each cell represents

a specific neuron's position within the repository grid.

Figure 4.11 shows an example of the reward function pipeline for the agent detector repository in the first hidden layer.

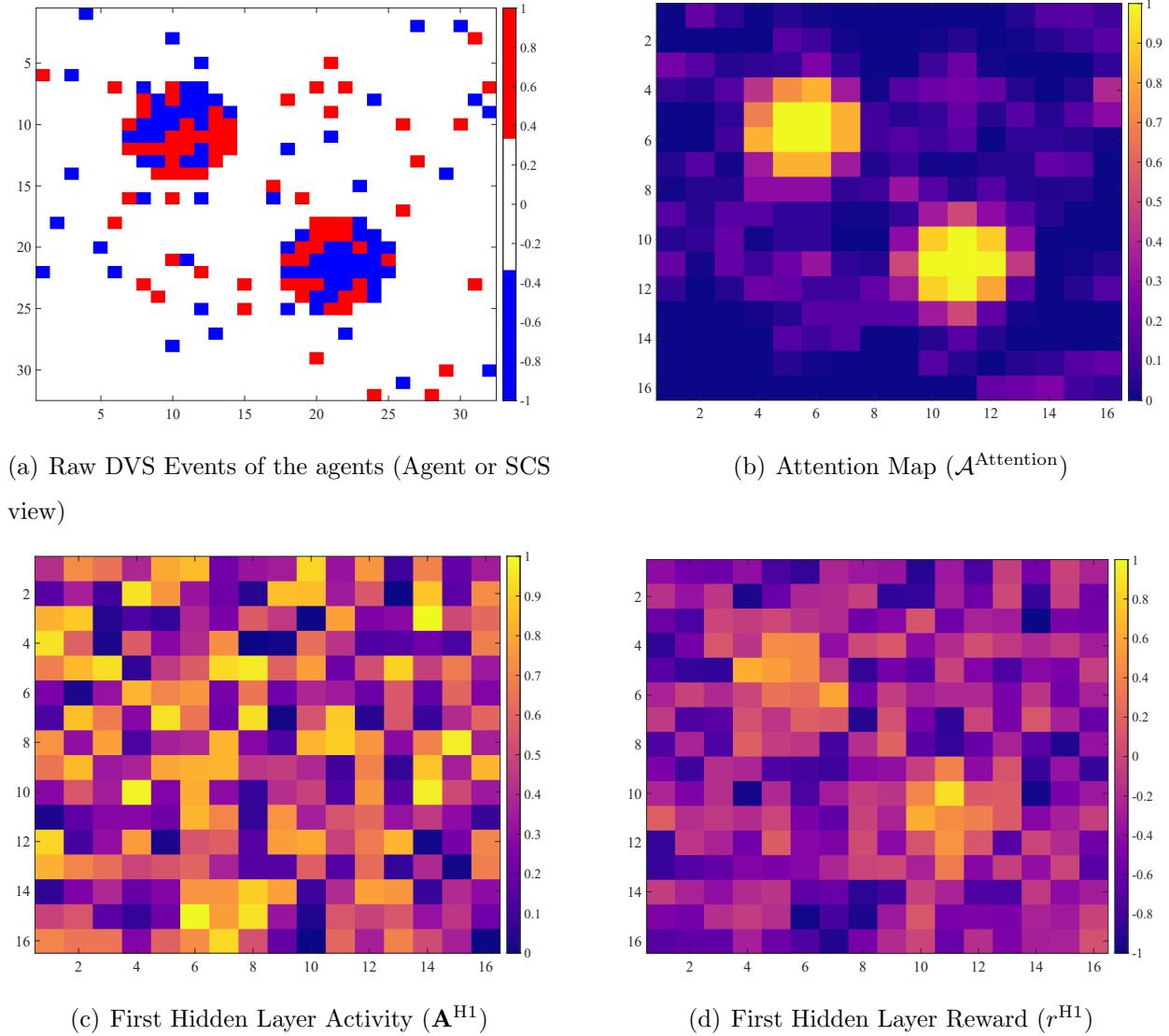


Figure 4.11: Reward function pipeline for the agents. This pipeline demonstrates the raw DVS events for multiple agents, the attention map, the hidden layer's normalized activity, and the modulation signal computed for reward.

This design enables the first hidden layer to adaptively learn distinct environmental configurations (the spatial positions of the payload and nearby agents), using event-based

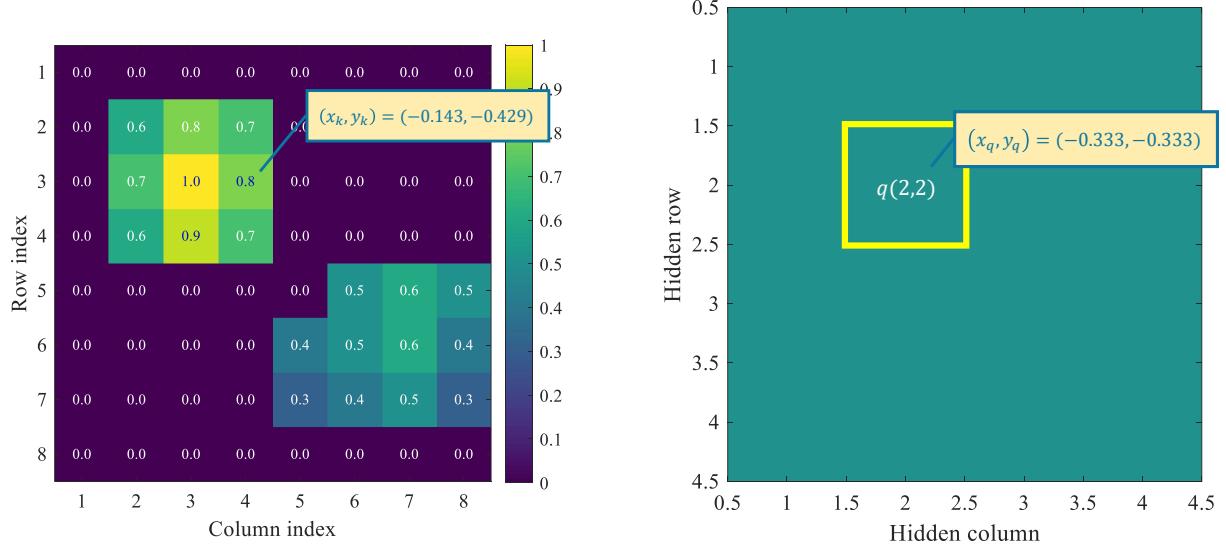
attention and ensures stable convergence in layered SNN learning.

Numerical example (explicit Gaussian attention with d_{kq}^2 and γ_{kq}) Use an 8×8 input grid and a 4×4 first-hidden-layer grid (same as Figure 4.7). Coordinates follow the code:

$$x_k, y_k \in \{-1, -0.714, -0.429, -0.143, 0.143, 0.429, 0.714, 1\}, \quad x_q, y_q \in \{-1, -0.333, 0.333, 1\}.$$

Set the attention scale to $\sigma_A = 0.1$ and $\epsilon = 0.001$. Let the time-averaged input layer activity (illustrated in Figure 4.7) be

$$\mathbf{A}^{\text{input}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.6 & 0.8 & 0.7 & 0 & 0 & 0 & 0 \\ 0 & 0.7 & 1.0 & 0.8 & 0 & 0 & 0 & 0 \\ 0 & 0.6 & 0.9 & 0.7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 & 0.6 & 0.5 \\ 0 & 0 & 0 & 0 & 0.4 & 0.5 & 0.6 & 0.4 \\ 0 & 0 & 0 & 0 & 0.3 & 0.4 & 0.5 & 0.3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$



(a) Input layer activity matrix $\mathbf{A}^{\text{input}}$ (shown above) for numerical example of Gaussian attention computation.

(b) Hidden-layer centers of attention (queries) for numerical example of Gaussian attention computation.

Figure 4.12: Numerical example setup for Gaussian attention computation in the first hidden layer. (a) shows the input layer activity matrix representing time-averaged firing rates from the input layer (4.39). (b) illustrates the spatial arrangement of hidden-layer neurons, each serving as a query point ($q(2, 2)$) for attention calculation.

In Figure 4.12, no physical positions are estimated and no coordinate transformation is performed. Fixed, normalized coordinates are assigned to grid indices so that each hidden-layer neuron can compute Gaussian distances to input cells. The 8×8 input grid and the 4×4 hidden grid share the same normalized $[-1, 1] \times [-1, 1]$ coordinate space but at different resolutions. Each input pixel index is mapped to a fixed coordinate (x_k, y_k) , and each hidden neuron q is mapped to the center of its corresponding coarse grid cell (x_q, y_q) . Panel (a) shows input activity on the fine grid, while panel (b) shows a hidden neuron acting as a query that softly aggregates nearby input activity using a Gaussian kernel based purely on grid distance, not physical geometry.

Separable Gaussian weights along axes. For a x_q in the first hidden layer, define the

8-vector (one entry for each of the eight input columns)

$$g_x(k | x_q) = \exp\left(-\frac{(x_k - x_q)^2}{2\sigma_A^2}\right), \quad 2\sigma_A^2 = 0.02.$$

Define the analogous 8-vector for the vertical axis, $g_y(r | y_q)$, which assigns a weight to each of the eight input rows centered at y_q . The full 8×8 weight map for neuron q is the outer product

$$\gamma_q(r, c) = g_y(r | y_q) g_x(c | x_q),$$

The attention at q is the normalized weighted average

$$\mathcal{A}^{\text{Attention}}(q) = \frac{\sum_{r,c} \gamma_{kq} \mathbf{A}^{\text{input}}(r, c)}{\sum_{r,c} \gamma_{kq} + \epsilon}.$$

Example 1: hidden neuron $q = (2, 2)$ at $(x_q, y_q) = (-0.333, -0.333)$. Compute the axis weights:

$$g_x(k | x_q) = \exp\left(-\frac{(x_k - x_q)^2}{0.02}\right).$$

$$x_k - x_q = -0.714 - (-0.333) = -0.381, \quad (x_k - x_q)^2 = 0.145161,$$

$$g_x(2 | -0.333) = \exp\left(-\frac{0.145161}{0.02}\right) = \exp(-7.25805) = 7.0448 \times 10^{-4}.$$

$$x_k - x_q = -0.429 - (-0.333) = -0.096, \quad (x_k - x_q)^2 = 0.009216,$$

$$g_x(3 | -0.333) = \exp\left(-\frac{0.009216}{0.02}\right) = \exp(-0.4608) = 0.63078.$$

$$x_k - x_q = -0.143 - (-0.333) = 0.190, \quad (x_k - x_q)^2 = 0.0361,$$

$$g_x(4 | -0.333) = \exp\left(-\frac{0.0361}{0.02}\right) = \exp(-1.805) = 0.16447.$$

$$x_k - x_q = 0.143 - (-0.333) = 0.476, \quad (x_k - x_q)^2 = 0.226576,$$

$$g_x(5 | -0.333) = \exp\left(-\frac{0.226576}{0.02}\right) = \exp(-11.3288) = 1.2022 \times 10^{-5} \approx 0.0000 \text{ (to 4 decimals)}.$$

$$g_x(\cdot | -0.333) = [0.0000, 0.0007, 0.6308, 0.1645, 0.0000, 0.0000, 0.0000, 0.0000],$$

$$g_y(\cdot | -0.333) = [0.0000, 0.0007, 0.6308, 0.1645, 0.0000, 0.0000, 0.0000, 0.0000].$$

Form $\gamma_q = g_y g_x^\top$. Non-negligible entries are the 2×2 block on rows 3:4, columns 3:4 (Figure 4.13(a)):

$$\gamma_q(3, 3) = 0.6308 \cdot 0.6308 = 0.398, \quad \gamma_q(3, 4) = 0.6308 \cdot 0.1645 = 0.104,$$

$$\gamma_q(4, 3) = 0.1645 \cdot 0.6308 = 0.104, \quad \gamma_q(4, 4) = 0.1645 \cdot 0.1645 = 0.027.$$

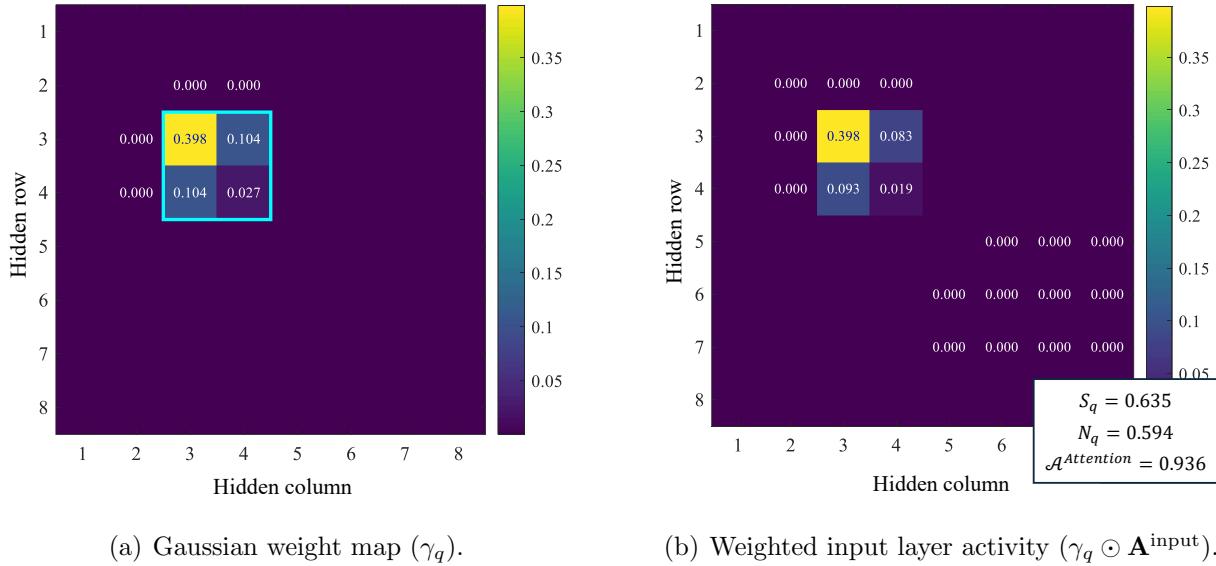


Figure 4.13: Attention weight computation for hidden neuron $q = (2, 2)$ at position $(x_q, y_q) = (-0.333, -0.333)$. (a) shows the Gaussian weight map γ_q centered at the hidden neuron's location. (b) illustrates the element-wise product of the weight map with the input layer activity matrix, highlighting the contributions to the attention calculation.

The denominator (with ϵ) is

$$S_q = \sum_{r,c} \gamma_q(r, c) + \epsilon = (0.398 + 0.104 + 0.104 + 0.027) + 0.001 = 0.635.$$

The numerator uses the corresponding entries from $\mathbf{A}^{\text{input}}$:

$$N_q = 0.404 \cdot 1.000 + 0.104 \cdot 0.800 + 0.104 \cdot 0.900 + 0.027 \cdot 0.700 = 0.594.$$

Hence

$$\mathcal{A}^{\text{Attention}}(2, 2) = \frac{N_q}{S_q} = \frac{0.594}{0.635} = 0.936.$$

Example 2: hidden neuron $q = (2, 1)$ at $(x_q, y_q) = (-1, -0.333)$. Here

$$g_x(\cdot | -1) = [1.000, 0.017, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000]$$

$$g_y(\cdot | -0.333) = [0.000, 0.001, 0.636, 0.163, 0.000, 0.000, 0.000, 0.000].$$

Non-negligible weights sit on rows 3:4 and columns 1:2:

$$\gamma_q(3, 1) = 0.636 \cdot 1.000 = 0.636, , \quad \gamma_q(3, 2) = 0.636 \cdot 0.017 = 0.011,$$

$$\gamma_q(4, 1) = 0.163 \cdot 1.000 = 0.163, , \quad \gamma_q(4, 2) = 0.163 \cdot 0.017 = 0.003.$$

The denominator is $S_q = 0.636 + 0.011 + 0.163 + 0.003 + 0.001 = 0.814$. The numerator uses $\mathbf{A}^{\text{input}}(3, 2) = 0.7$ and $\mathbf{A}^{\text{input}}(4, 2) = 0.6$:

$$N_q = 0.011 \cdot 0.7 + 0.003 \cdot 0.6 = 0.009.$$

Thus

$$\mathcal{A}^{\text{Attention}}(2, 1) = \frac{0.009}{0.814} = 0.012.$$

Example 3: hidden neuron $q = (3, 3)$ at $(x_q, y_q) = (0.333, 0.333)$. Axis weights are centered on columns 5:6 and rows 5:6:

$$g_x(\cdot | 0.333) = [0, 0, 0, 0.163, 0.636, 0.000, 0, 0] \quad (\text{nonzeros at col 5:6 are } 0.163, 0.636),$$

$$g_y(\cdot | 0.333) = [0, 0, 0, 0.163, 0.636, 0, 0, 0] \quad (\text{nonzeros at row 5:6 are } 0.163, 0.636).$$

The four main weights are

$$\gamma_q(5, 5) = 0.163 \cdot 0.163 = 0.027, \quad \gamma_q(5, 6) = 0.163 \cdot 0.636 = 0.104,$$

$$\gamma_q(6, 5) = 0.636 \cdot 0.163 = 0.104, \quad \gamma_q(6, 6) = 0.636 \cdot 0.636 = 0.404.$$

With $S_q = 0.404 + 0.104 + 0.104 + 0.027 + 0.001 = 0.640$ and

$$N_q = 0.027 \cdot 0.000 + 0.104 \cdot 0.500 + 0.104 \cdot 0.400 + 0.404 \cdot 0.500 = 0.296,$$

one obtains

$$\mathcal{A}^{\text{Attention}}(3, 3) = \frac{0.296}{0.640} = 0.463.$$

Example 4: hidden neuron $q = (3, 4)$ at $(x_q, y_q) = (1, 0.333)$. Now g_x sits on columns 7:8 with weights (0.017, 1.000) and g_y on rows 5:6 with (0.163, 0.636). The four main weights are

$$\gamma_q(5, 7) = 0.163 \cdot 0.017 = 0.003, \quad \gamma_q(5, 8) = 0.163 \cdot 1.000 = 0.163,$$

$$\gamma_q(6, 7) = 0.636 \cdot 0.017 = 0.011, \quad \gamma_q(6, 8) = 0.636 \cdot 1.000 = 0.636,$$

so $S_q = 0.814$. With

$$N_q = 0.003 \cdot 0.600 + 0.163 \cdot 0.500 + 0.011 \cdot 0.600 + 0.636 \cdot 0.400 = 0.344,$$

we get

$$\mathcal{A}^{\text{Attention}}(3, 4) = \frac{0.344}{0.814} = 0.423.$$

Assembling the 4×4 attention map. Repeating the above computation for all $q \in \{1, \dots, 4\}^2$ produces the attention values

$$\mathcal{A}^{\text{Attention}} = \begin{bmatrix} 0.000 & 0.014 & 0.000 & 0.000 \\ 0.012 & 0.936 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.463 & 0.423 \\ 0.000 & 0.000 & 0.006 & 0.005 \end{bmatrix}.$$

These values correspond to the normalized Gaussian attention centered at the four-by-four hidden-layer coordinates and reflect the degree to which each hidden neuron “queries” the spatio-temporal activity of the input grid.

Hidden-layer activities. Let the current hidden activities for the payload and agent

detectors be

$$\mathbf{A}_{\text{payload}}^{\text{H1}} = \begin{bmatrix} 0.050 & 0.100 & 0.000 & 0.000 \\ 0.100 & 0.200 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.100 & 0.200 \\ 0.000 & 0.000 & 0.050 & 0.100 \end{bmatrix}, \quad \mathbf{A}_{\text{agent}}^{\text{H1}} = \begin{bmatrix} 0.000 & 0.020 & 0.000 & 0.000 \\ 0.010 & 0.010 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.010 & 0.010 \\ 0.000 & 0.000 & 0.020 & 0.020 \end{bmatrix}.$$

The activity matrices presented above are defined for the scenario when the ALM is turned off, which allows the first hidden layer to focus on detecting the payload. The small, nonzero values in the agent activity matrix occur because inhibition is not always perfect. Occasionally, some agent detector neurons (or payload detector neurons when the ALM is activated) may spike once or twice within the time window due to residual excitation or incomplete suppression.

Reward computation. Using the multiplicative reward definition $r^{\text{H1}}(q) = \mathbf{A}^{\text{H1}}(q)(\mathcal{A}_q^{\text{Attention}} - \mathbf{A}^{\text{H1}}(q))$, the elementwise rewards for the payload detector become

$$r_{\text{payload}}^{\text{H1}} = \begin{bmatrix} -0.0025 & -0.0086 & 0 & 0 \\ -0.0088 & 0.1472 & 0 & 0 \\ 0 & 0 & 0.0363 & 0.0446 \\ 0 & 0 & -0.0022 & -0.0095 \end{bmatrix}.$$

With the updated agent-detector activity matrix, the corresponding reward values are

$$r_{\text{agent}}^{\text{H1}} = \begin{bmatrix} 0 & -0.00012 & 0 & 0 \\ 0.00002 & 0.00926 & 0 & 0 \\ 0 & 0 & 0.00453 & 0.00413 \\ 0 & 0 & -0.00028 & -0.00030 \end{bmatrix}.$$

The combined reward vector is obtained by concatenating the row-wise vectorizations of

the two matrices,

$$r^{H1} = \begin{bmatrix} \text{rowvec}(r_{\text{payload}}^{H1}) \\ \text{rowvec}(r_{\text{agent}}^{H1}) \end{bmatrix} \in \mathbb{R}^{32 \times 1}.$$

Positive entries indicate that the attention-driven input estimate exceeds the neuron's present activity, leading to increase in synaptic weights under the R-STDP rule. Negative values indicate that the neuron is currently overactive relative to the attended input and should undergo a reduction in synaptic strength. Since only one of the two repositories is active at any time because of lateral inhibition, only the corresponding reward matrix contributes to synaptic updates during each training iteration.

Reward Function for the Second Hidden Layer

Incoming DVS events are represented as an image $E \in \{-1, 0, +1\}^{N_e \times N_e}$, where each pixel encodes a negative, positive, or absent brightness change. The first hidden layer consists of two distinct neuron sections dedicated to payload detection and proximal agent detection, respectively. For each section, the time-averaged spiking activity is aggregated according to the activity-matrix construction rule illustrated in Figure 4.7, resulting in two separate grid-based activity maps, $\mathbf{A}_{\text{pay}}^{H1}$ and $\mathbf{A}_{\text{agt}}^{H1}$, each of size $N_m \times N_m$.

Each grid cell in $\mathbf{A}_{\text{pay}}^{H1}$ and $\mathbf{A}_{\text{agt}}^{H1}$ corresponds to a fixed group of neurons in the respective section of the first hidden layer and represents the local firing-rate activity for that spatial region. These maps provide a spatial representation of the payload and proximal agents, indicating the extent to which the network associates each grid location with the presence of the corresponding object. The first hidden layer also incorporates a lateral inhibition mechanism, which ensures that only one of these two detection maps is active at any given time. When the payload detector neurons are active, the agent detector neurons are suppressed, and vice versa. Consequently, the instantaneous detection state of the network can be expressed as

$$\mathbf{A}_{\text{det}}^{H1} = \mathbf{A}_{\text{pay}}^{H1} + \mathbf{A}_{\text{agt}}^{H1}, \quad (4.46)$$

since at any given moment exactly one of the two terms on the right-hand side of (4.46) is nonzero.

To construct a binary activity-selection mask from the detection map, a Sigmoid function is applied element-wise,

$$\mathbf{B}(i, j) = \frac{1}{1 + \exp[-\beta (\mathbf{A}_{\text{det}}^{\text{H1}}(i, j) - \Upsilon)]}, \quad (4.47)$$

here, Υ denotes a soft threshold, and $\beta \gg 1$ determines the steepness of the transition. This operation converts the detection map into a logical mask that distinguishes active from inactive grid cells. Grid locations with nonzero and sufficiently large values of $\mathbf{A}_{\text{det}}^{\text{H1}}(i, j)$ approach 1, whereas locations with zero or negligible activity remain near 0. Consequently, the resulting mask identifies spatial regions with high neural activity and suppresses regions that do not contribute to the detection process.

The DVS produces events at the sensor resolution $N_e \times N_e$, corresponding to the native pixel grid of the event-based camera. In contrast, the activity maps generated by the entropy-based pooling input layer and the first hidden layer are defined on a coarser spatial grid of size $N_m \times N_m$, where each grid cell aggregates the activity of a fixed block of DVS pixels described in Fig. 4.7. Throughout this work, N_e denotes the event (sensor) resolution, while N_m refers to the pooled neural resolution. To apply the low-resolution activity-selection mask \mathbf{B} to the original DVS event image, the mask must be expanded from a size of $N_m \times N_m$ to the event resolution $N_e \times N_e$. Since the ratio $\kappa = N_e/N_m$ is an integer, this expansion is implemented using nearest-neighbor replication. Each entry of \mathbf{B} is copied into a $\kappa \times \kappa$ block in the high-resolution grid, producing a mask that is spatially aligned with the original DVS events. Formally,

$$\mathcal{E}(\mathbf{B}) = \mathbf{B} \otimes \mathbf{1}_{\kappa \times \kappa}, \quad (\mathcal{E}(\mathbf{B}))_{m,n} = B_{\lceil m/\kappa \rceil, \lceil n/\kappa \rceil}. \quad (4.48)$$

This operation ensures that every pixel (m, n) in the enlarged mask $\mathcal{E}(\mathbf{B})$ inherits the value of the corresponding coarse mask element from \mathbf{B} , producing a properly aligned high event

resolution binary mask. Finally, the upsampled mask is applied to the DVS event frame to isolate only the currently detected object (payload or proximal agent),

$$E_{\text{det}} = E \odot \mathcal{E}(\mathbf{B}), \quad (4.49)$$

where \odot denotes element-wise multiplication. The resulting event image contains only the events emitted from the active target, with all other regions suppressed by the mask.

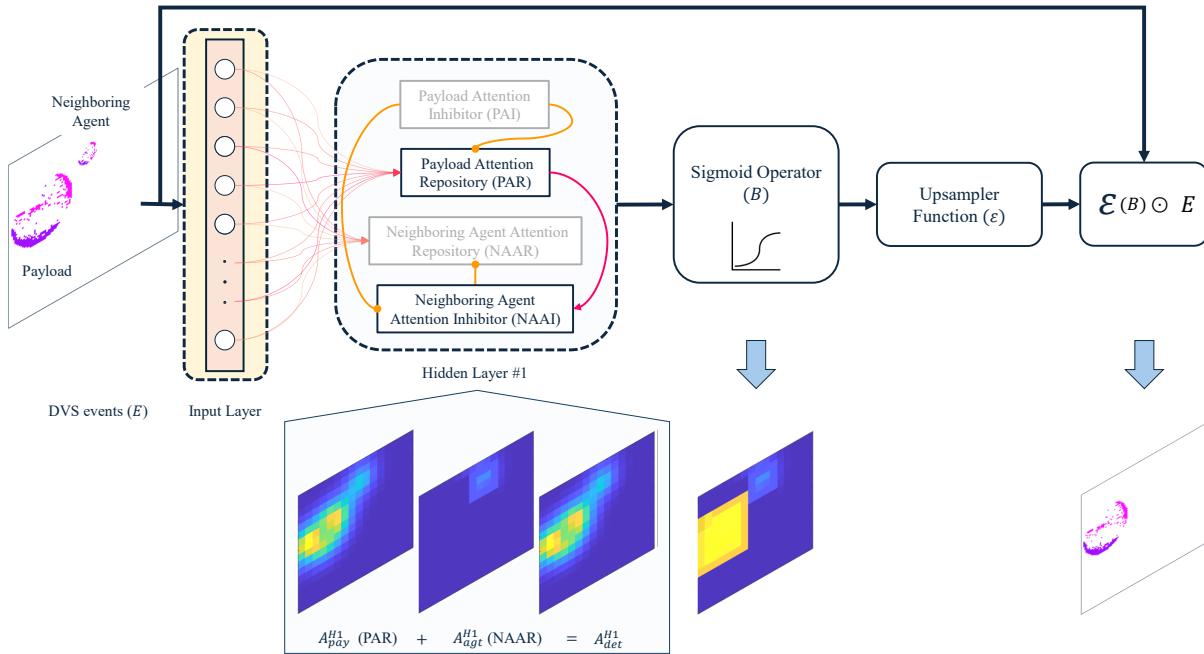


Figure 4.14: The DVS event image is masked by the upsampled binary detection mask derived from the first hidden layer’s output, isolating events corresponding to the active object (payload or agent).

For the second hidden layer, the reward is designed to align synaptic updates with the spatial and temporal focus of the task. The temporal modulation is governed by the agent’s

ALM blinking period T_{ALM} , with oscillation frequency and angular frequency

$$f_{\text{osc}} = \frac{1}{T_{\text{ALM}}}, \quad \omega = 2\pi f_{\text{osc}}. \quad (4.50)$$

The scalar quantity used to detect whether the observed object is moving toward or away from the current center of attention is computed as

$$\mathcal{H} = \frac{1}{Z} \sum_{m=1}^{N_e} \sum_{n=1}^{N_e} E^{\text{det}}(m, n) \mathbf{G}^{(e)}(m, n, t), \quad (4.51)$$

where Z is a normalization constant.

Here, $\mathbf{G}^{(e)}(m, n, t)$ denotes a two-dimensional Gaussian envelope sampled at the DVS resolution $N_e \times N_e$. The kernel $\mathbf{G}^{(e)}$ is defined as

$$\mathbf{G}^{(e)}(m, n, t) = \alpha(t) \exp\left(-\frac{(x_m - x_c(t))^2 + (y_n - y_c(t))^2}{2\sigma(t)^2}\right), \quad (4.52)$$

where (x_m, y_n) are the spatial coordinates of pixel (m, n) . During the rendezvous phase, the Gaussian center $(x_c(t), y_c(t))$ is fixed at the center of the onboard DVS, while during the docking phase it is fixed at the center of the SCS. The parameters $\sigma(t)$ and $\alpha(t)$ control the spatial spread and amplitude of the envelope and are modulated over time according to the ALM blinking frequency. Figure 4.15 shows the change in the Gaussian function's amplitude during time (switching between -1 and +1), where the frequency of the change is defined based on the ALM blinking frequency.

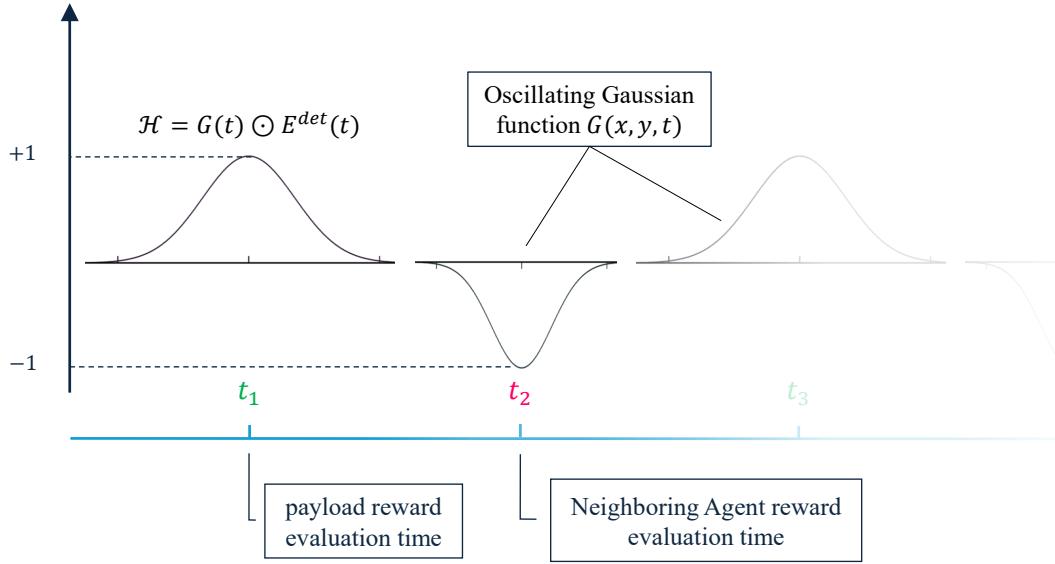
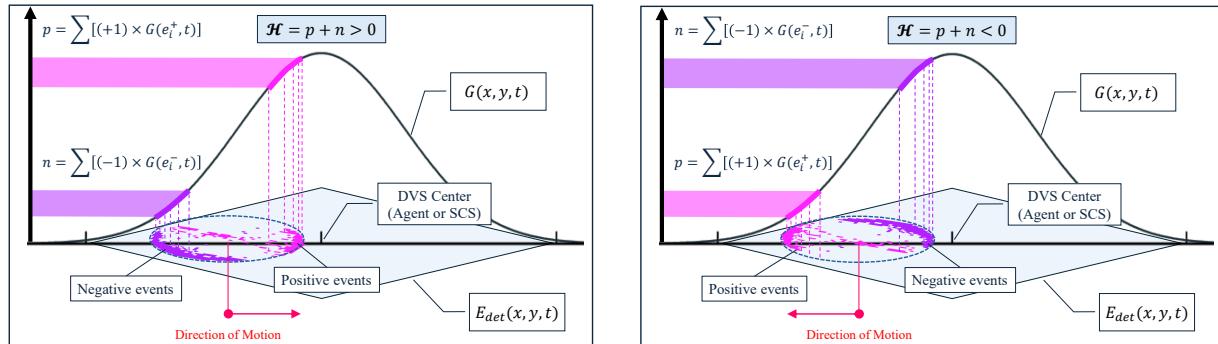


Figure 4.15: Oscillatory behavior of the Gaussian envelope used for computing the motion direction signal \mathcal{H} in the second hidden layer. The amplitude alternates between positive and negative values, shifting focus between the payload and agents based on the active ALM.



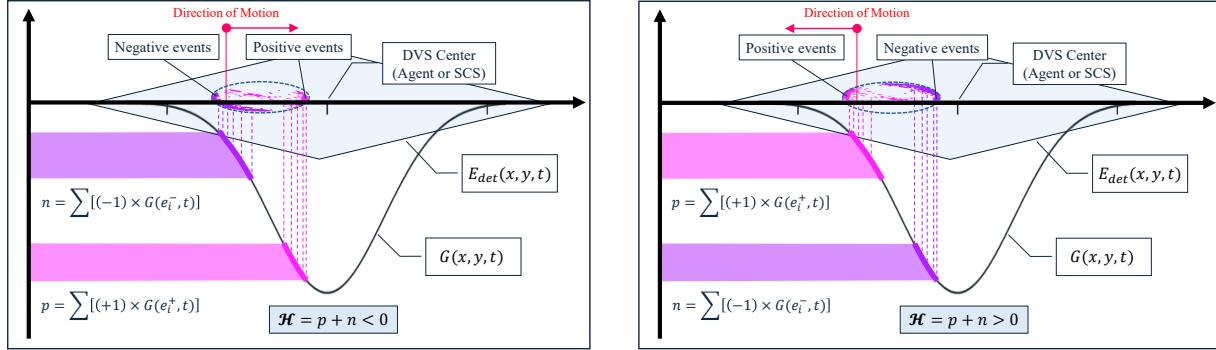
(a) Payload approaching the center results in positive \mathcal{H} value based on the 2D Gaussian function with higher standard deviation.

(b) Payload moving away from the center results in negative \mathcal{H} value based on the 2D Gaussian function with higher standard deviation.

Figure 4.16: Payload motion direction detection using a 2D Gaussian reward function with a higher standard deviation. When the payload moves closer

Figure 4.16 illustrates how the motion direction of the payload is detected using a 2D Gaussian reward function with a higher standard deviation. When the payload moves closer

to the center of the Gaussian envelope ($G(x, y, t)$), it results in a positive \mathcal{H} value, indicating favorable movement towards the target. Conversely, when the payload moves away from the center, it yields a negative \mathcal{H} value, signaling unfavorable movement away from the target.

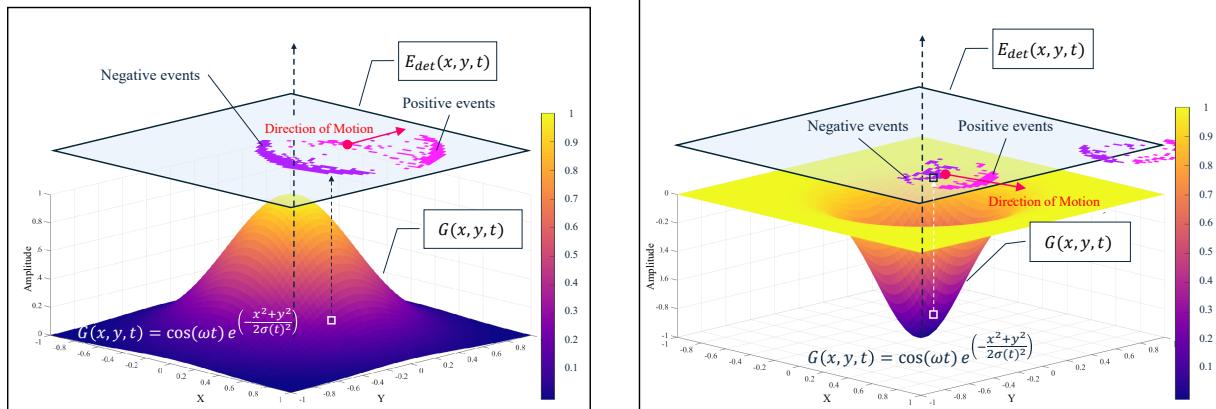


(a) Neighboring agent approaching the center results in negative \mathcal{H} value based on the 2D Gaussian function with lower standard deviation.

(b) Neighboring agent moving away from the center results in positive \mathcal{H} value based on the 2D Gaussian function with lower standard deviation.

Figure 4.17: Neighboring agent motion direction detection using a 2D Gaussian reward function with lower standard deviation.

Figure 4.17 demonstrates how the motion direction of a neighboring agent is detected using a 2D Gaussian reward function with a lower standard deviation. When the neighboring agent moves closer to the center of the Gaussian envelope ($G(x, y, t)$), it results in a negative \mathcal{H} value, indicating unfavorable movement towards the target. Conversely, when the neighboring agent moves away from the center, it yields a positive \mathcal{H} value, signaling favorable movement away from the target.



(a) The oscillatory Gaussian envelope when the ALM is off (payload's moves away from the center results in negative values). (b) The oscillatory Gaussian envelope when the agent ALM is on (Agent's moves away from the center results in positive values).

Figure 4.18: The oscillatory Gaussian envelope used to compute the motion direction signal \mathcal{H} for reward modulation in the second hidden layer. The center can be SCS or the agent's DVS center.

Figure 4.18 illustrates the oscillatory Gaussian envelope used to compute the motion direction signal \mathcal{H} for reward modulation in the second hidden layer. The envelope alternates between focusing on the payload and the agents based on which ALM is active.

Each agent is equipped with a planar inertial measurement unit (IMU) that measures the planar velocity $\mathbf{V} = [V_x, V_y]^\top$. From this measurement, the instantaneous heading and normalized speed are computed as

$$\varphi = \text{atan2}(V_y, V_x), \quad \bar{V} = \frac{\|\mathbf{V}\|}{V_{\max}}, \quad (4.53)$$

where V_{\max} denotes the maximum admissible planar speed.

To obtain a population-based representation of velocity, the velocity space is discretized into an $N_g \times N_g$ grid (as shown in Figure 4.7 for the activity matrix). Each grid cell corresponds to a fixed reference direction (x_i, y_j) , with $x_i, y_j \in [-1, 1]$. These reference

points are expressed in normalized polar coordinates as

$$R_{ij} = \frac{\sqrt{x_i^2 + y_j^2}}{\sqrt{2}}, \quad \Theta_{ij} = \text{atan2}(y_j, x_i). \quad (4.54)$$

For a given velocity direction φ , the angular mismatch between the reference direction Θ_{ij} and the actual heading is computed as follows:

$$\Delta\theta_{ij} = \text{atan2}(\sin(\Theta_{ij} - \varphi), \cos(\Theta_{ij} - \varphi)). \quad (4.55)$$

A Gaussian membership function is used to quantify the similarity between the current velocity and each grid cell in terms of both speed and direction. The radial and angular memberships are defined as

$$\mu_r(i, j) = \exp\left(-\frac{(R_{ij} - \bar{V})^2}{2\sigma_r^2}\right), \quad \mu_\theta(i, j) = \exp\left(-\frac{\Delta\theta_{ij}^2}{2\sigma_\theta^2}\right). \quad (4.56)$$

Their product yields a velocity-aligned activity template,

$$\mathbf{M}_{ij} = \frac{\mu_r(i, j) \mu_\theta(i, j)}{\max_{a,b} \mu_r(a, b) \mu_\theta(a, b)}, \quad (4.57)$$

which represents the desired neural activity pattern associated with the agent's current velocity. An opposite-direction template is obtained by rotating the heading by π ,

$$\widetilde{\mathbf{M}} = \mathbf{M}(\varphi + \pi). \quad (4.58)$$

Let $\mathbf{A} \in [0, 1]$ denote the normalized activity of neurons in the second hidden layer, defined as the number of spikes emitted within a time window ΔT divided by a maximum firing count. The scalar reward signal \mathcal{H} is decomposed into positive and negative components as

$$\mathcal{H}_+ = \frac{\mathcal{H} + |\mathcal{H}|}{2}, \quad \mathcal{H}_- = \frac{|\mathcal{H}| - \mathcal{H}}{2}, \quad (4.59)$$

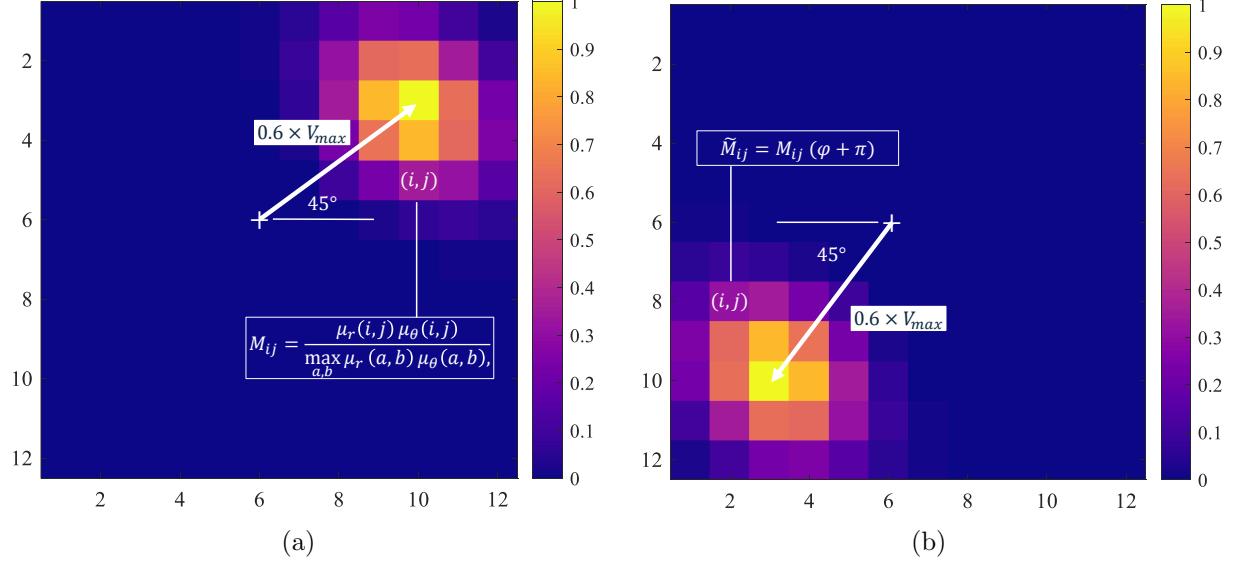


Figure 4.19: Velocity-aligned activity template \mathbf{M} (left) and opposite-direction template $\tilde{\mathbf{M}}$ (right) for an agent moving with heading $\varphi = \pi/4$ and normalized speed $\bar{V} = 0.6$.

with $\mathcal{H}_+ \geq 0$ representing reinforcement and $\mathcal{H}_- \geq 0$ representing punishment.

Using this decomposition, the per-neuron reward assignment is defined as

$$L_i = \mathcal{H}_+ \mathbf{M}_i + \mathcal{H}_- \tilde{\mathbf{M}}_i, \quad (4.60)$$

such that positive reward reinforces neurons aligned with the current velocity direction, while negative reward reinforces neurons encoding the opposite direction.

Finally, the reward-modulated learning signal applied to the second hidden layer is given by

$$r^{H2} = \begin{cases} \mathcal{H}_+(\mathbf{M} - \mathbf{A}) + \mathcal{H}_-(\tilde{\mathbf{M}} - \mathbf{A}), & \text{if } (V_x \neq 0) \vee (V_y \neq 0), \\ 0, & \text{if } V_x = 0 \wedge V_y = 0. \end{cases} \quad (4.61)$$

which is active only when the agent is in motion. According to Figure 4.19, this signal drives the neural activity toward the velocity-consistent template \mathbf{M} under positive reward and toward the opposite template $\tilde{\mathbf{M}}$ under negative reward, while vanishing when $\mathcal{H} = 0$. If the direction of motion is correct based on getting closer to the payload, the reward will be

positive, and the synaptic weights will be adjusted to reinforce this behavior. Conversely, if the motion is toward the neighboring agent, the reward will be negative, leading to synaptic adjustments that discourage this behavior.

Numerical example for the second hidden layer reward computation. We set $N_e = 8$, $N_m = 4$, hence the upsampling factor is $\kappa = N_e/N_m = 2$. Let's assume that at the current instant, the oscillatory envelope has amplitude $\alpha(t) = 1$ (the agent's ALM is off, and the payload is detected). Unless noted, all numbers are rounded to 3 decimals.

Inputs. Let's assume that the DVS data and the firing activity matrix of the payload and neighboring agent detector repositories in the first hidden layer (shown in figure 4.8) are

$$E = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{A}_{pay}^{H1} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0.1 \\ 0 & 0.1 & 0.8 & 0.7 \\ 0 & 0.3 & 0.7 & 0.5 \end{bmatrix}, \quad \mathbf{A}_{agt}^{H1} = \mathbf{0}_{4 \times 4}.$$

Thus $A_{det}^{H1} = \mathbf{A}_{pay}^{H1}$.

Binarization and upsampling. According to (4.47) with threshold $\Upsilon = 0.3$ and steepness $\beta = 10$, the binary mask \mathbf{B} is computed as

$$\mathbf{B}(i, j) = \frac{1}{1 + \exp(-\beta(A_{det}^{H1}(i, j) - \Upsilon))}.$$

For example $\mathbf{B}(3, 3) = \frac{1}{1+\exp(-10(0.8-0.3))} = \frac{1}{1+e^{-5}} = 0.993$. The full matrix is

$$\mathbf{B} = \begin{bmatrix} 0.047 & 0.047 & 0.047 & 0.047 \\ 0.047 & 0.047 & 0.119 & 0.119 \\ 0.047 & 0.119 & 0.993 & 0.982 \\ 0.047 & 0.500 & 0.982 & 0.881 \end{bmatrix}.$$

Nearest-neighbor upsampling by $\kappa = 2$ gives $\mathcal{E}(\mathbf{B}) = \mathbf{B} \otimes \mathbf{1}_{2 \times 2}$:

$$\mathcal{E}(\mathbf{B}) = \begin{bmatrix} 0.047 & 0.047 & 0.047 & 0.047 & 0.047 & 0.047 & 0.047 & 0.047 \\ 0.047 & 0.047 & 0.047 & 0.047 & 0.047 & 0.047 & 0.047 & 0.047 \\ 0.047 & 0.047 & 0.047 & 0.047 & 0.119 & 0.119 & 0.119 & 0.119 \\ 0.047 & 0.047 & 0.047 & 0.047 & 0.119 & 0.119 & 0.119 & 0.119 \\ 0.047 & 0.047 & 0.119 & 0.119 & 0.993 & 0.993 & 0.982 & 0.982 \\ 0.047 & 0.047 & 0.119 & 0.119 & 0.993 & 0.993 & 0.982 & 0.982 \\ 0.047 & 0.047 & 0.500 & 0.500 & 0.982 & 0.982 & 0.881 & 0.881 \\ 0.047 & 0.047 & 0.500 & 0.500 & 0.982 & 0.982 & 0.881 & 0.881 \end{bmatrix}.$$

Masked events.

$$E_{\text{det}} = E \odot \mathcal{E}(\mathbf{B}) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.047 & -0.047 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.047 & -0.047 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.993 & 0.982 & 0 \\ 0 & 0 & 0 & 0 & -0.993 & 0 & 0.982 & 0 \\ 0 & 0 & 0 & 0 & -0.982 & -0.982 & -0.881 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Gaussian kernel and scalar reward. We sample a centered Gaussian kernel $\mathbf{G}^{(e)}$ on the 8×8 grid with coordinates $\{-0.875, -0.625, -0.375, -0.125, 0.125, 0.375, 0.625, 0.875\}$ on

each axis and width $\sigma_g = 0.6$ based on (4.52) with $\alpha(t) = 1$ and center at the origin:

$$\mathbf{G}^{(e)}(m, n) = \exp\left(-\frac{x^2 + y^2}{2\sigma_g^2}\right).$$

This yields

$$\mathbf{G}^{(e)} = \begin{bmatrix} 0.119 & 0.201 & 0.284 & 0.338 & 0.338 & 0.284 & 0.201 & 0.119 \\ 0.201 & 0.338 & 0.478 & 0.569 & 0.569 & 0.478 & 0.338 & 0.201 \\ 0.284 & 0.478 & 0.677 & 0.805 & 0.805 & 0.677 & 0.478 & 0.284 \\ 0.338 & 0.569 & 0.805 & 0.958 & 0.958 & 0.805 & 0.569 & 0.338 \\ 0.338 & 0.569 & 0.805 & 0.958 & 0.958 & 0.805 & 0.569 & 0.338 \\ 0.284 & 0.478 & 0.677 & 0.805 & 0.805 & 0.677 & 0.478 & 0.284 \\ 0.201 & 0.338 & 0.478 & 0.569 & 0.569 & 0.478 & 0.338 & 0.201 \\ 0.119 & 0.201 & 0.284 & 0.338 & 0.338 & 0.284 & 0.201 & 0.119 \end{bmatrix}, \quad Z = \sum_{m,n} \mathbf{G}^{(e)}(m, n) = 29.761.$$

The motion-direction signal is

$$\mathcal{H} = \frac{1}{Z} \sum_{m,n} E_{\text{det}}(m, n) \mathbf{G}^{(e)}(m, n).$$

Only 11 positions are nonzero,

$$\begin{aligned} (2, 2) : 0.047 \cdot 0.338 &= 0.016, & (2, 3) : -0.047 \cdot 0.478 &= -0.023, \\ (3, 2) : 0.047 \cdot 0.478 &= 0.023, & (3, 3) : -0.047 \cdot 0.677 &= -0.032, \\ (5, 6) : 0.993 \cdot 0.805 &= 0.800, & (5, 7) : 0.982 \cdot 0.569 &= 0.559, \\ (6, 5) : -0.993 \cdot 0.805 &= -0.800, & (6, 7) : 0.982 \cdot 0.478 &= 0.469, \\ (7, 5) : -0.982 \cdot 0.569 &= -0.559, & (7, 6) : -0.982 \cdot 0.478 &= -0.470, \\ (7, 7) : -0.881 \cdot 0.338 &= -0.298, \end{aligned}$$

whose sum is -0.314 . Therefore

$$\mathcal{H} = \frac{-0.314}{29.761} = -0.011, \quad \mathcal{H}_+ = 0, \quad \mathcal{H}_- = \frac{|\mathcal{H}| - \mathcal{H}}{2} = 0.011.$$

which means that the payload is moving away from the center of the Gaussian envelope and the agents receive negative reward.

Velocity encoding and memberships. Take $\mathbf{V} = [0.6, 0.2]^\top$ (illustrative), $V_{\max} = 1$. Then

$$\varphi = \text{atan2}(0.2, 0.6) = 0.322 \text{ rad}, \quad \bar{\mathcal{V}} = \min(\|\mathbf{V}\|/V_{\max}, 1) = 0.632.$$

On a 3×3 grid with $(x_i, y_j) \in \{-1, 0, 1\}^2$, define $R_{ij} = \sqrt{x_i^2 + y_j^2}/\sqrt{2}$ and $\Theta_{ij} = \text{atan2}(y_j, x_i)$ (set $\Theta_{2,2} = \varphi$ at the origin). With $\sigma_r = 0.3$, $\sigma_\theta = 0.6$,

$$\mu_r = \begin{bmatrix} 0.472 & 0.970 & 0.472 \\ 0.970 & 0.108 & 0.970 \\ 0.472 & 0.970 & 0.472 \end{bmatrix}, \quad \mu_\theta = \begin{bmatrix} 0.000 & 0.007 & 0.182 \\ 0.000 & 1.000 & 0.866 \\ 0.003 & 0.115 & 0.742 \end{bmatrix}.$$

As an example for the above calculation, the 1.000 entry corresponds to the center cell. With the convention $\Theta_{2,2} = \varphi$ at the origin, the angular mismatch at the center cell is $\Delta\theta_{2,2} = 0$, hence $\mu_\theta(2, 2) = \exp(0) = 1$.

Normalize $\mathbf{M}_{ij} = \mu_r(i, j)\mu_\theta(i, j)/\max_{a,b} \mu_r(a, b)\mu_\theta(a, b)$ and form the opposite-direction map $\widetilde{\mathbf{M}} = \mathbf{M}(\varphi + \pi)$:

$$\mathbf{M} = \begin{bmatrix} 0.000 & 0.008 & 0.102 \\ 0.000 & 0.129 & 1.000 \\ 0.002 & 0.132 & 0.417 \end{bmatrix}, \quad \widetilde{\mathbf{M}} = \begin{bmatrix} 0.417 & 0.132 & 0.002 \\ 1.000 & 0.000 & 0.000 \\ 0.102 & 0.008 & 0.000 \end{bmatrix}.$$

Per-neuron reward and signed credit. Let the normalized second-layer activities be (Figure (4.7))

$$\mathbf{A} = \begin{bmatrix} 0.100 & 0.200 & 0.300 \\ 0.400 & 0.500 & 0.200 \\ 0.100 & 0.300 & 0.600 \end{bmatrix}.$$

Since $\mathcal{H} < 0$,

$$r^{H2} = \mathcal{H}_+(\mathbf{M} - \mathbf{A}) + \mathcal{H}_-(\widetilde{\mathbf{M}} - \mathbf{A}) = 0.011(\widetilde{\mathbf{M}} - \mathbf{A}),$$

so

$$r^{H2} = \begin{bmatrix} 0.003 & -0.001 & -0.003 \\ 0.006 & -0.005 & -0.002 \\ 0.000 & -0.003 & -0.006 \end{bmatrix}.$$

The shape of this reward (matrix form) follows the same rule as Figure 4.7, and each array shows the reward value for each associated neuron.

Interpretation. The masked events and Gaussian envelope yield a small *negative* motion signal (-0.011), so learning pushes activity toward the *opposite* velocity map $\widetilde{\mathbf{M}}$: cells aligned with $\widetilde{\mathbf{M}}$ are encouraged, while currently overactive cells are suppressed by the negative entries of $\dot{\mathbf{W}}$.

Output Layer Reward Computation Based on Directional Error

The output layer consists of four neurons that generate thrust commands along the Cartesian directions $\pm X$ and $\pm Y$. The role of the reward at this layer is to align the generated action with a commanded motion direction provided by the preceding (second hidden) layer.

The second hidden layer encodes motion direction using a population of neurons whose preferred directions are uniformly distributed over the unit circle. Each neuron responds most strongly when the motion direction matches its preferred orientation. Let $\mu_k \in [0, 2\pi)$ denote the preferred direction of the k -th neuron. The activity of each neuron is quantified by a binned spike count $w_k \geq 0$, defined as the total number of spikes emitted within a fixed temporal window of duration $\Delta T = 10$ ms. This temporal binning converts discrete spike trains into a nonnegative scalar activity that can be used for population-level decoding.

The commanded motion direction is extracted from this population activity using a circular mean. Specifically, the population is interpreted as a set of weighted unit vectors oriented along μ_k , and their vector sum is computed as

$$C = \sum_k w_k \cos(\mu_k), \quad S = \sum_k w_k \sin(\mu_k). \quad (4.62)$$

The commanded direction is then given by the angle of the resulting vector,

$$\theta_{\text{cmd}} = \text{atan2}(S, C), \quad (4.63)$$

which corresponds to the circular mean of the population activity and yields a direction.

The output layer consists of four neurons associated with thrust along the negative and positive X and Y directions. Let a_x and a_y denote the net thrust components along the X and Y axes, obtained as the difference between the spike activities of neurons encoding opposite directions and averaged over a 50 ms window. These components define the instantaneous action vector, $\mathbf{a} = (a_x, a_y)$. The direction of this vector represents the action executed by the network. Since only the direction of the action is relevant for reward assignment, explicit normalization of \mathbf{a} is not required; the angular information is extracted directly via

$$\theta_{\text{act}} = \text{atan2}(a_y, a_x). \quad (4.64)$$

To assess directional consistency, the commanded and executed directions are compared through their trigonometric components. The discrepancies along the X and Y axes are defined as

$$\Delta_c = \cos(\theta_{\text{cmd}}) - \cos(\theta_{\text{act}}), \quad \Delta_s = \sin(\theta_{\text{cmd}}) - \sin(\theta_{\text{act}}). \quad (4.65)$$

These quantities measure the mismatch between the desired and actual motion directions in each Cartesian dimension.

Based on these discrepancies, the per-neuron rewards for the output layer are assigned as

$$r_{X^-} = \Delta_c, \quad r_{X^+} = -\Delta_c, \quad r_{Y^-} = \Delta_s, \quad r_{Y^+} = -\Delta_s. \quad (4.66)$$

This reward structure reinforces neurons whose activity contributes to reducing the directional error between the commanded motion (decoded from the second hidden layer) and the executed action, while suppressing neurons that drive motion in the opposite direction.

Figure 4.20 illustrates the process of computing the output layer reward based on the

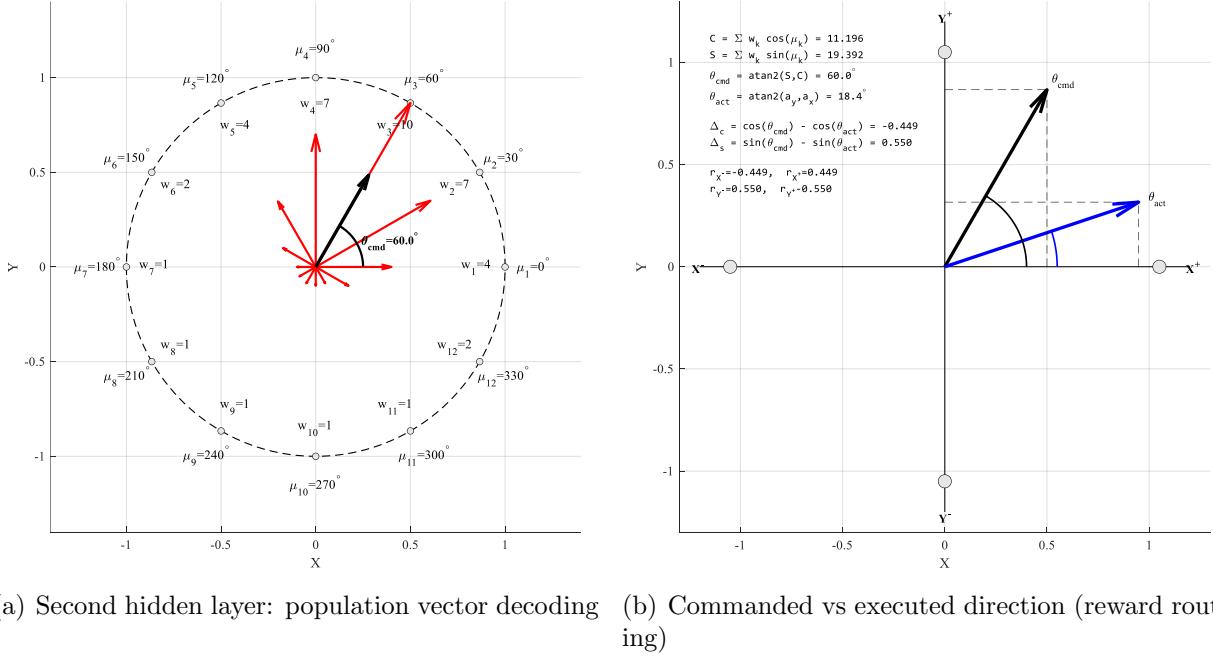


Figure 4.20: Output layer reward computation based on directional error between commanded and executed motion.

directional error between the commanded motion from the second hidden layer and the executed action. Figure 4.20a shows how the population activity is decoded into a commanded direction. Red arrows show the activity of each neuron in the second hidden layer. Figure 4.20b depicts how the directional discrepancies are translated into per-neuron rewards for the output layer. In this example, the commanded direction is at 60° , while the executed action by the output neurons is at 18.4° . The executed angle (θ_{act}) is measured using IMU (a_x and a_y). The resulting rewards encourage increased activity in the r_{Y^-} neuron and reduce activity in the r_{X^-} neuron to correct the directional error. The reward also encourages the activity of the neurons that will help align the executed action with the commanded direction.

Weight Update using R-STDP

Simultaneous plasticity across multiple layers in spiking neural networks leads to unstable learning [99]. This occurs because downstream synapses adapt while upstream feature rep-

resentations are still evolving. Accordingly, prior work in R-STDP, biologically inspired deep SNNs, and hierarchical reinforcement learning employs mechanisms that delay, gate, or attenuate learning signals until lower-level representations stabilize [100].

To ensure consistent hierarchical learning, synaptic plasticity in each layer is modulated by the convergence state of the preceding layer. Let $r_i^{\ell-1}$ denote the instantaneous reward signal associated with neuron i in layer $\ell - 1$. The overall activity-dependent instability of that layer is summarized by the scalar measure

$$\mathcal{R}^{\ell-1} = \max_i |r_i^{\ell-1}|, \quad (4.67)$$

which captures the largest magnitude of reward-driven fluctuations within the layer.

Since the reward signals $r_i^{\ell-1}$ are bounded by construction, the quantity $\mathcal{R}^{\ell-1}$ is finite and nonnegative. Using this measure, the effective reward driving learning in layer ℓ is defined as

$$r_{\text{eff}}^\ell = r^\ell \exp(-a_L \mathcal{R}^{\ell-1}), \quad (4.68)$$

where $a_L > 0$ is a tunable damping coefficient.

This exponential modulation suppresses downstream plasticity when the preceding layer exhibits large reward fluctuations, indicating ongoing adaptation. As learning in layer $\ell - 1$ stabilizes and its weight adjustment becomes negligible ($|r_i^{\ell-1}|$ goes to zero), the $\mathcal{R}^{\ell-1}$ decreases, enabling gradual adaptation in layer ℓ .

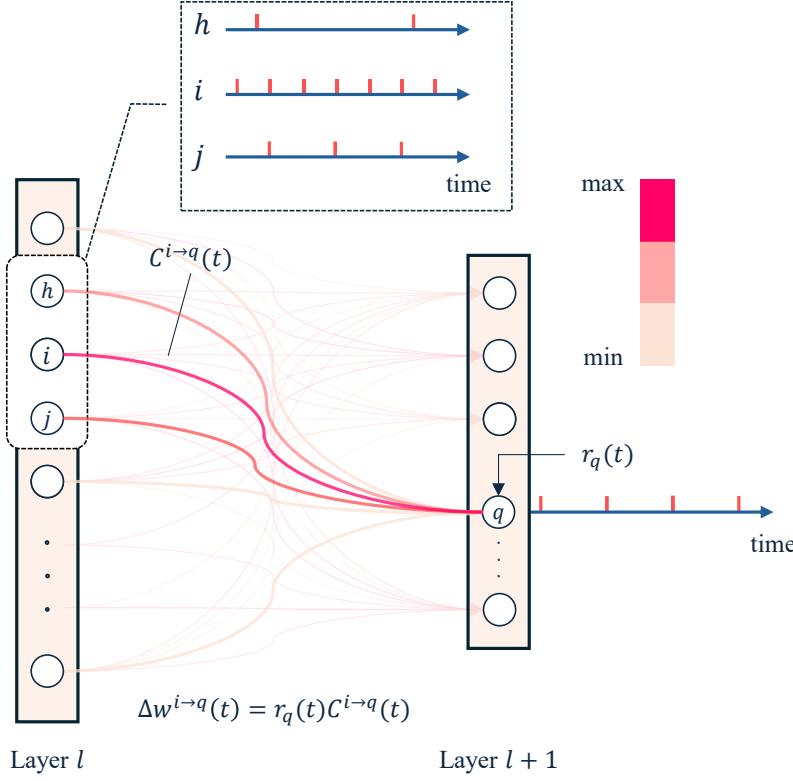


Figure 4.21: Schematic of the per-layer R-STDP weight update. The eligibility trace is calculated based on the firing rate of the neurons in layer ℓ and $\ell + 1$. The per-neuron reward (r_q) is computed based on the layer-specific reward functions. The weight update is obtained by multiplying the eligibility trace by the per-neuron reward.

As shown in Figure 4.21, the network is partitioned into layers using indices $\{\ell = 1, \dots, L - 1\}$. For each layer ℓ , let \mathcal{J}_ℓ be the set of presynaptic indices (e.g., neurons h, i , and j in Figure 4.21) and $\mathcal{I}_{\ell+1}$ the set of postsynaptic indices (h, i , and j in Layer l). Denote by $\mathbf{C}^{(\ell)}(t) \in \mathbb{R}^{|\mathcal{I}_{\ell+1}| \times |\mathcal{J}_\ell|}$ the eligibility (credit-assignment) matrix produced by the R-STDP kernel from pre- and post-spike timings up to time t . In continuous time an admissible form is

$$\dot{\mathbf{C}}^\ell(t) = -\frac{1}{\tau_c} \mathbf{C}^\ell(t) + \mathbf{STDP}(\tau) \delta(t - \mathbf{T}^\ell) \quad (4.69)$$

where, τ_c is the constant for the decay of $\mathbf{C}^\ell(t)$, the τ is the spike time difference between pre- and post-synaptic neurons (e.g., h and q in Figure 4.21), and \mathbf{T}^ℓ is the matrix that shows

the firing times of pre- and post-synaptic neurons. In discrete implementation this eligibility is maintained internally and sampled at update times; the weight rule below treats $\mathbf{C}^{(\ell)}(t)$ as known.

Per-neuron rewards computed on the postsynaptic side of each layer. The instantaneous, gated R-STDP update for layer ℓ is then

$$\Delta \mathbf{W}^{(\ell)}(t) = \mathbf{r}_{\text{eff}}^{(\ell)}(t) \circ \mathbf{C}^{(\ell)}(t), \quad (4.70)$$

where \circ denotes element-wise multiplication.

4.3 Results

4.3.1 Multi-Agent Docking Task Simulation Setup

In our simulation of the multi-agent docking task, the central payload is modeled as a rigid disc of radius R_p and mass m_p . A team of identical agents, each represented as a point mass m with its own collision radius R , cooperatively pushes the payload towards the docking center. Agent–payload and inter-agent collisions are handled via a linear spring–damper model with stiffness k and damping coefficient c . Each agent can exert a thrust force of up to F_{\max} in the x - and y -directions. The specific parameter values used in all simulations are listed in Table 4.1 [101].

Component	Parameter	Value	Units
Payload	Radius, R_p	2	m
	Mass, m_p	14000	kg
Agent	Mass, m	30	kg
	Radius, R	0.5	m
	Stiffness, k	100	kg/s ²
	Damping, c_{damp}	0.1	kg/s
Max thrust, F_{\max}		400	N

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

According to the International Docking System Standard (IDSS) [102], soft capture is

required to occur with a lateral (radial) misalignment of less than 0.10 m and a relative lateral rate not exceeding 0.04 m/s at initial contact.

In our SNN implementation, we configure four layers of Izhikevich neurons, each simulated with a time step of $\Delta t = 0.1$ ms. The input layer comprises 1024 neurons that encode the preprocessed DVS currents. The first hidden layer contains 512 visual-processing neurons together with 20 PAI and 20 NAAI units, for a total of 552 neurons; the ALM repositories operate in the Class 2 excitability regime for smooth frequency modulation. The second hidden layer comprises 130 Class 2 neurons: 45 RPR, 45 DPR, 20 RPI, and 20 DPI. Finally, the output layer consists of 4 Bistability neurons that generate directional PWM control signals for the X - and Y -axis thrust commands. Table 4.2 summarizes these layer configurations.

Layer	Type	# Neurons	Description
Input	Bistability	1024	Preprocessed DVS input currents
Hidden 1	Class 2	552	256 Payload Attention Repository (PAR) 256 Neighboring Agent Attention Repository (NAAR) 20 Payload Attention Inhibitor (PAI) 20 Neighboring Agent Attention Inhibitor (NAAI)
Hidden 2	Class 2	130	45 Rendezvous Phase Repository (RPR) 20 Rendezvous Phase Inhibitor (RPI) 45 Docking Phase Repository (DPR) 20 Docking Phase Inhibitor (DPI)
Output	Bistability	4	Generates PWM thrust commands for X - Y axes

Table 4.2: Neuron layer configurations with full descriptions of each functional subgroup.

Parameter	Hidden Layer 1	Hidden Layer 2	Output Layer
State update interval $T_{\text{upd}}^{(\ell)}$ [ms]	10	10	50
Eligibility constant $\tau_C^{(\ell)}$ [ms]	7	7	70
STDP time constant $\tau_{\text{STDP}}^{(\ell)}$ [ms]	10	10	100
Reward interval $T_r^{(\ell)}$ [ms]	10	10	50

Table 4.4: Layer-wise Learning Parameters

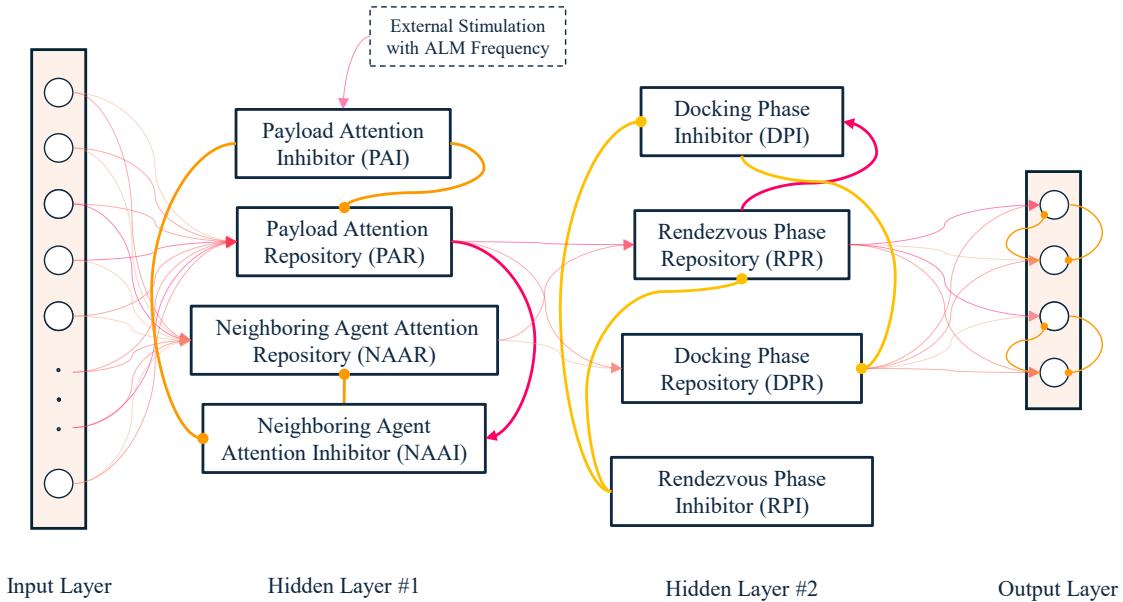


Figure 4.22: High-level architecture of the spiking neural network for multi-agent docking. The network processes DVS input through attention and phase repositories to generate thrust commands for docking maneuvers.

Table 4.3.1 details the learning-related parameters and schedules used by the network during training. Each layer employs STDP modulated by reward signals, with eligibility traces governing synaptic updates. The parameters are carefully chosen to balance learning speed and stability, ensuring effective adaptation to the docking task.

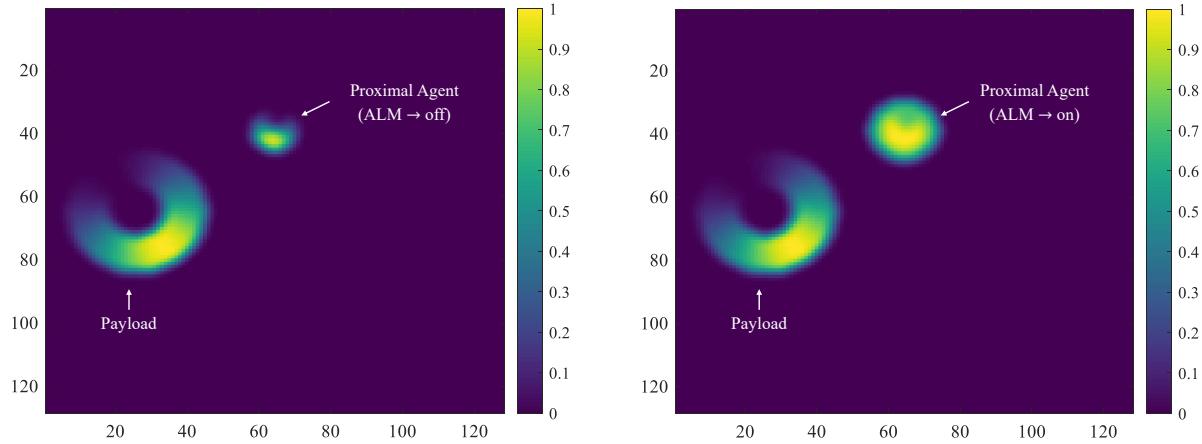
To generate a stable inhibitory control signal, each inhibitory repository (PAI, NAAI, RPI, and DPI) is implemented as a small population of neurons rather than a single inhibitory

unit. Each neuron in the repository emits inhibitory spikes, which are delivered to the target repository via inhibitory synapses. The effective inhibitory signal is the sum of the population’s spiking activity, rather than the output of an individual neuron.

Distributing inhibition across multiple neurons improves temporal reliability. Spike trains from individual neurons are inherently variable. A single inhibitory neuron may remain silent for extended intervals because of stochastic firing. By contrast, a population of inhibitory neurons ensures spikes are distributed over time, so at least one neuron fires within each sampling window. As a result, the downstream repository receives a continuous inhibitory or gating signal rather than brief, intermittent bursts.

To achieve this, small random perturbations are applied to neurons’ initial membrane potentials and Izhikevich parameters within each inhibitory repository. This heterogeneity prevents synchronous firing. Without it, identically initialized neurons tend to spike simultaneously, producing short inhibitory bursts and long periods of inactivity. Random initialization causes neurons to spike at slightly different times, spreading inhibitory events, and maintaining stable inhibition throughout each phase.

Figure 4.23 illustrates the effect of activating the ALM on the agents’ visual perception. When the ALM is turned off (Figure 4.23a), the proximal agent produces relatively weak brightness variations compared to the payload, resulting in limited event generation in the DVS. As a result, the visual perception layer struggles to distinguish the smaller agent from the background. In contrast, when the ALM is turned on (Figure 4.23b), the proximal agent generates a higher concentration of brightness and corresponding DVS events despite its smaller physical size relative to the payload. This increased event activity enhances the network’s attention mechanism, allowing it to more clearly separate and detect proximal agents alongside the payload.



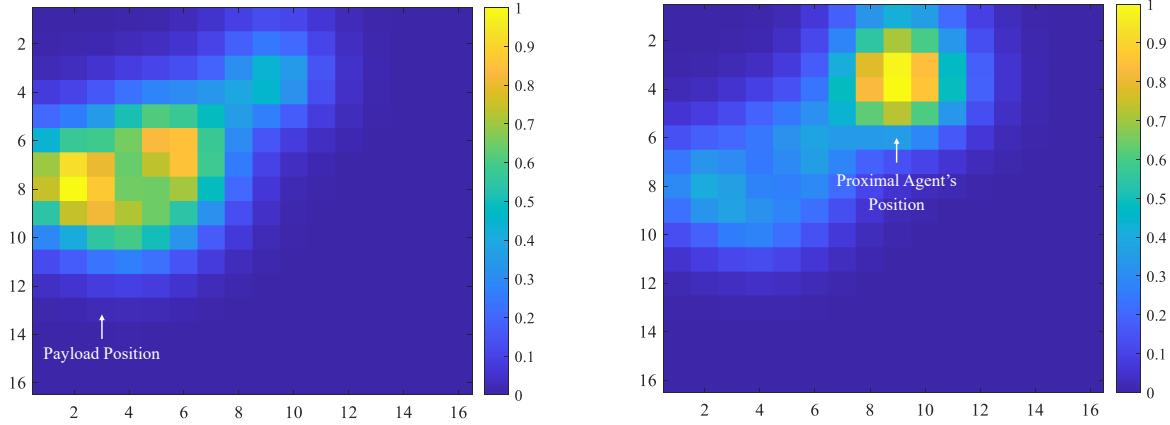
(a) Simulation of the environment with Agents' ALM turned off, showing baseline brightness distribution. (b) Simulation of the environment with Agents' ALM turned on, illustrating enhanced visibility and separation of each agent.

Figure 4.23: Effect of the ALM on visual perception. The ALM significantly enhances the visibility of proximal agents in the DVS input, improving the network’s ability to detect and differentiate them from the payload.

To visualize the activity patterns of the first hidden layer, the spiking responses of each repository are mapped onto a two-dimensional grid. Each repository consists of 256 neurons, spatially organized into a 16×16 grid, with each neuron represented by a corresponding cell. The color intensity of each cell reflects the normalized firing rate of the associated neuron, providing an interpretable heatmap of repository activity. This representation allows us to observe how different object detections—payloads or proximal agents—are encoded as localized activity clusters within their respective repositories under different ALM conditions.

Figure 4.24 demonstrates how the attention mechanism is modulated by the ALM. When the ALM is inactive (Figure 4.24a), the attention map is primarily focused on the payload, as its larger size produces stronger brightness cues in the DVS stream. However, once the ALM is activated (Figure 4.24b), the frequency-modulated blinking of the ALM redirects the attention toward the proximal agent. This shift results from periodic light emission, which produces distinct bursts of DVS events. Synchronization of these bursts with inhibitory

gating connections from ALM neurons in the first hidden layer suppresses competing channels and enables the network to isolate the agent’s position.



(a) Attention map when the Agents’ ALM is off, highlighting the detected payload position. (b) Attention map when the Agents’ ALM is on, emphasizing the detected proximal agent’s position.

Figure 4.24: Attention maps illustrating the effect of the ALM on visual focus. The ALM enables dynamic shifting of attention from the payload to the proximal agent by generating distinct DVS event patterns. Colorbar values indicate the normalized attention strength, where brighter regions correspond to locations receiving stronger attention.

Figure 4.25 illustrates the structural configuration of the SNN’s first hidden layer under two conditions. In Figure 4.25a, the agents’ ALM is turned off. In Figure 4.25b, the ALM is turned on. In both scenarios, the first hidden layer contains two distinct repositories: one for payload detection and one for proximal agent detection. The inhibitory connections from the ALM neurons play a crucial role in modulating the activity of these repositories. When the ALM is off, the payload detector repository dominates. This allows the network to focus on learning representations of the payload. When the ALM is activated, it inhibits the payload detector repository. As a result, the proximal agent detector repository is released from suppression.

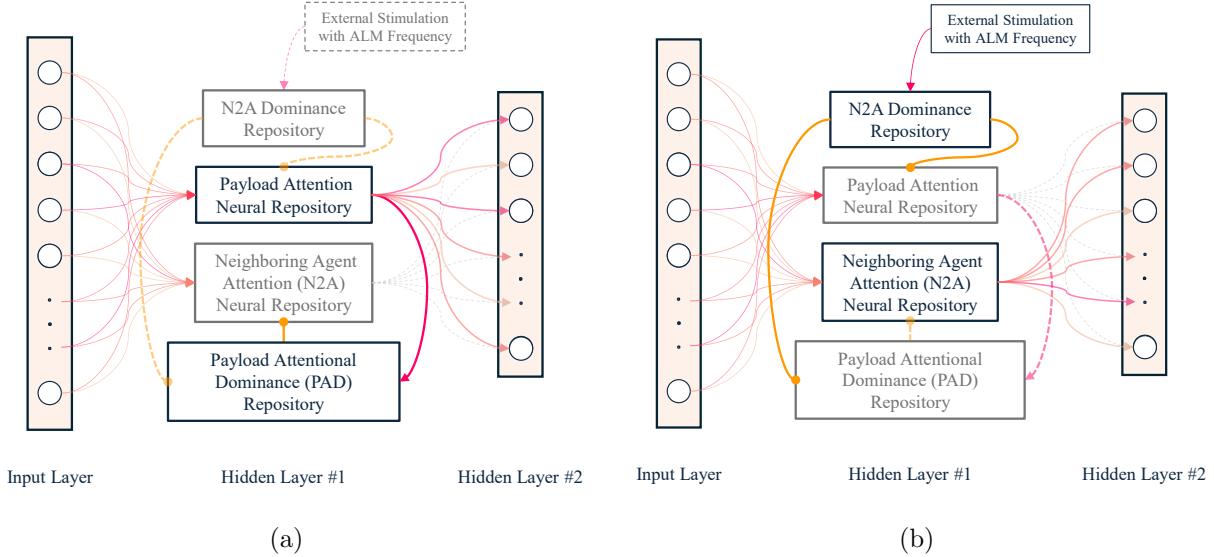
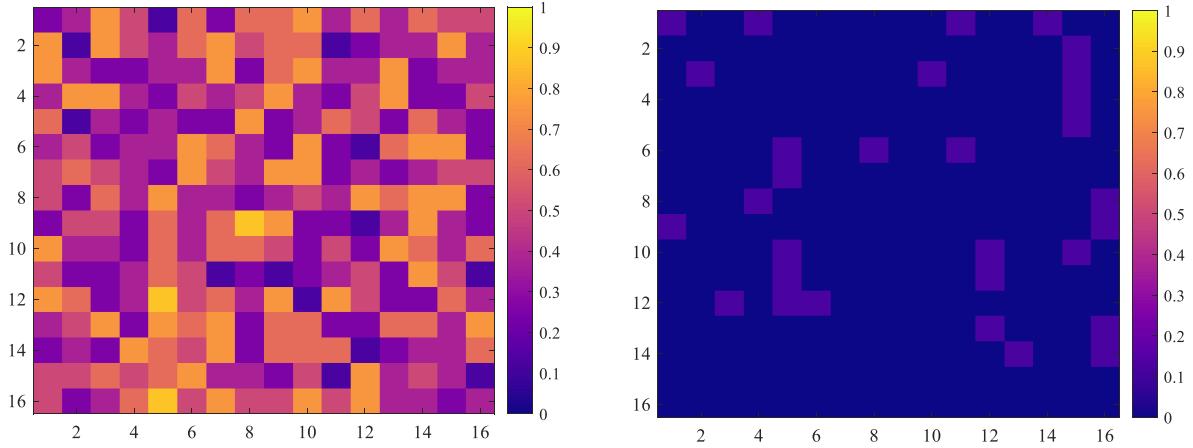


Figure 4.25: SNN structure in the first hidden layer under externally controlled attention signaling: (a) agents' ALM disabled, which corresponds to the absence of ALM illumination and results in payload-focused visual processing; (b) agents' ALM enabled, which corresponds to active ALM illumination and emphasizes proximal agent detection. Inactive neuron repositories and synaptic connections are depicted in gray, while dotted lines represent inactive inhibitory pathways.

Figure 4.26 shows how the payload and proximal agent detector repositories behave during the early training phase when the agents' ALM is inactive. The grid cells are generated according to the neurons' firing rates shown in Figure 4.7. As illustrated in Figure 4.26a, the payload detector repository exhibits strong and consistent activity because it responds directly to the bright and stationary visual signature of the payload. In contrast, the proximal agent detector repository remains largely inactive (Figure 4.26b). Its activity is suppressed by inhibitory signals from the payload detector repository, which dominates the competition when the ALM is off. This inhibitory interaction causes the network to allocate most of its representational capacity to the payload. As a result, the R-STDP learning rule primarily reinforces the synaptic pathways leading to the payload detector repository, while the connections to the proximal agent detector repository receive little or no strengthening at this

stage.

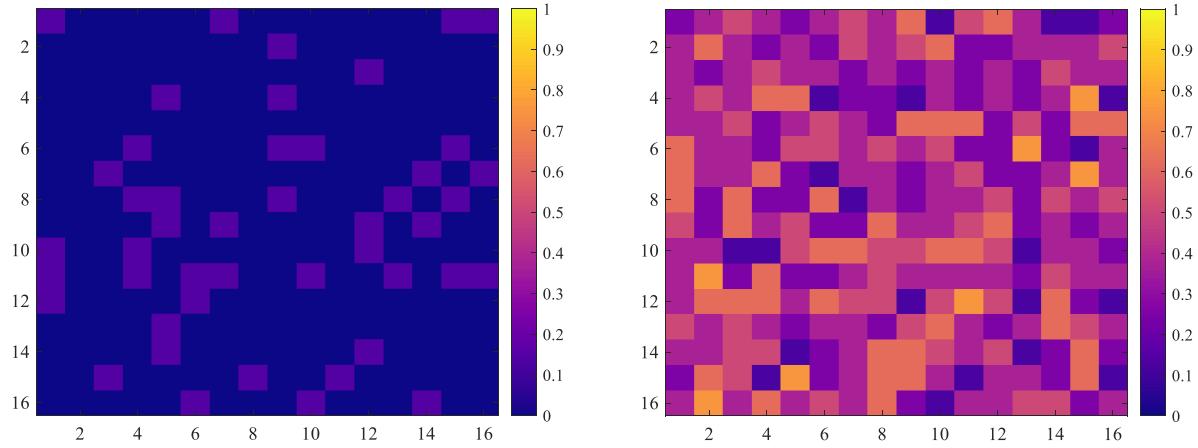


(a) Activity of the payload detector repository during initial training phases with the Agents' ALM turned off.

(b) Activity of the proximal agent detector repository during initial training phases with the Agents' ALM turned off.

Figure 4.26: Detector repository activity during initial training with ALM off. The payload detector remains active, while the proximal agent detector is suppressed due to inhibitory signals from the payload attention Inhibitor.

Figure 4.27 shows the complementary case to the previous condition, where the agents' ALM is active. In Figure 4.27a, the payload detector repository is largely suppressed. This results from the inhibitory influence exerted by the ALM neurons. This inhibition allows the proximal agent detector repository to become strongly active, as shown in Figure 4.27b. In this phase, the R-STDP mechanism focuses synaptic plasticity on the connections between the input layer and the proximal agent repository. This enables the network to learn representations specific to the agents while avoiding interference from the payload channel.

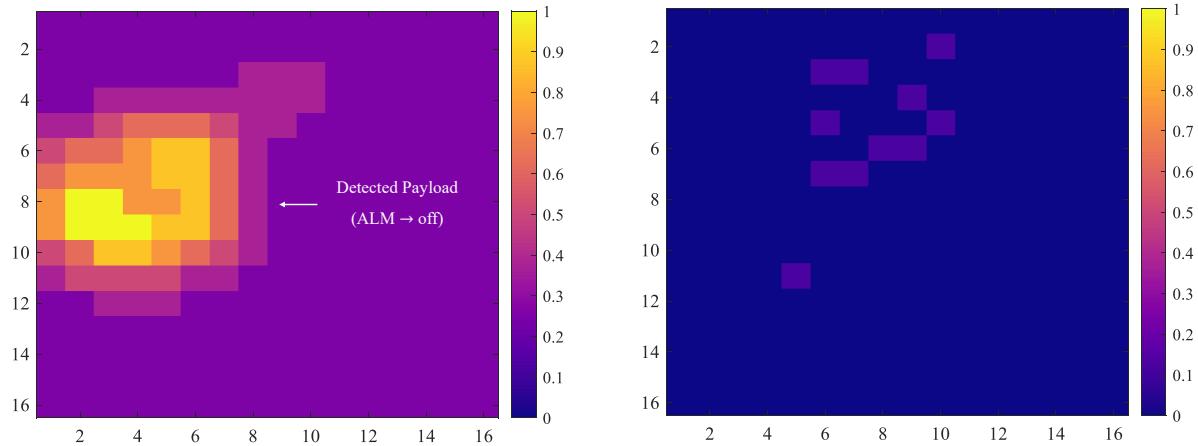


(a) Activity of the payload detector repository during initial training phases with the Agents' ALM turned on.

(b) Activity of the proximal agent detector repository during initial training phases with the Agents' ALM turned on.

Figure 4.27: Detector repository activity during initial training with ALM on. The proximal agent detector becomes active, while the payload detector is suppressed by ALM-induced inhibition.

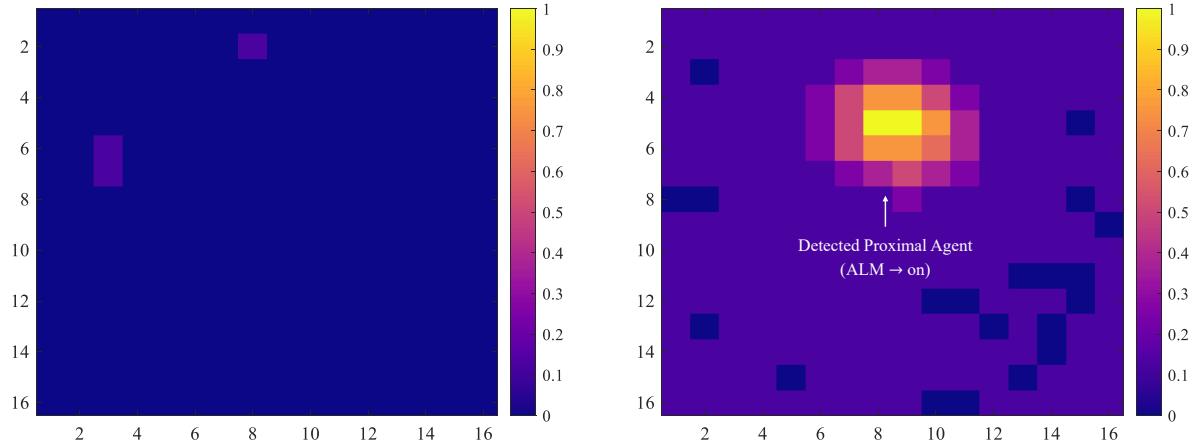
Figure 4.28 shows the activity of the first hidden layer repositories after convergence when the agents' ALM is turned off. As shown in Figure 4.28a, the payload detector repository has successfully learned to represent the payload, producing a dense, localized activation pattern that clearly highlights the detected object. In contrast, the proximal agent detector repository (Figure 4.28b) remains largely silent, consistent with the inhibitory gating mechanism that prevents its activation during this ALM-off phase. These results confirm that, after training with R-STDP, the network converges to a stable representation of the payload in the absence of ALM stimulation, demonstrating the first hidden layer's ability to encode and isolate salient objects once learning stabilizes.



(a) Activity of the payload detector repository after learning convergence with the Agents' ALM turned off.
 (b) Activity of the proximal agent detector repository after learning convergence with the Agents' ALM turned off.

Figure 4.28: Agent detector repository activity after learning convergence with ALM off. The payload detector shows strong, localized activity, while the proximal agent detector remains inactive.

Figure 4.29 shows the activity of the first hidden layer repositories after learning convergence when the agents' ALM is turned on. In Figure 4.29a, the payload detector repository remains almost completely inactive due to inhibitory gating. The proximal agent detector repository (Figure 4.29b) shows a clear, localized activation corresponding to the detected agent. This result demonstrates that, once training converges, the network can reliably isolate and represent the proximal agent by leveraging the ALM-induced event bursts. These findings highlight that the ALM not only enhances visibility but also enables the first hidden layer to switch attention between payload and agent detection.



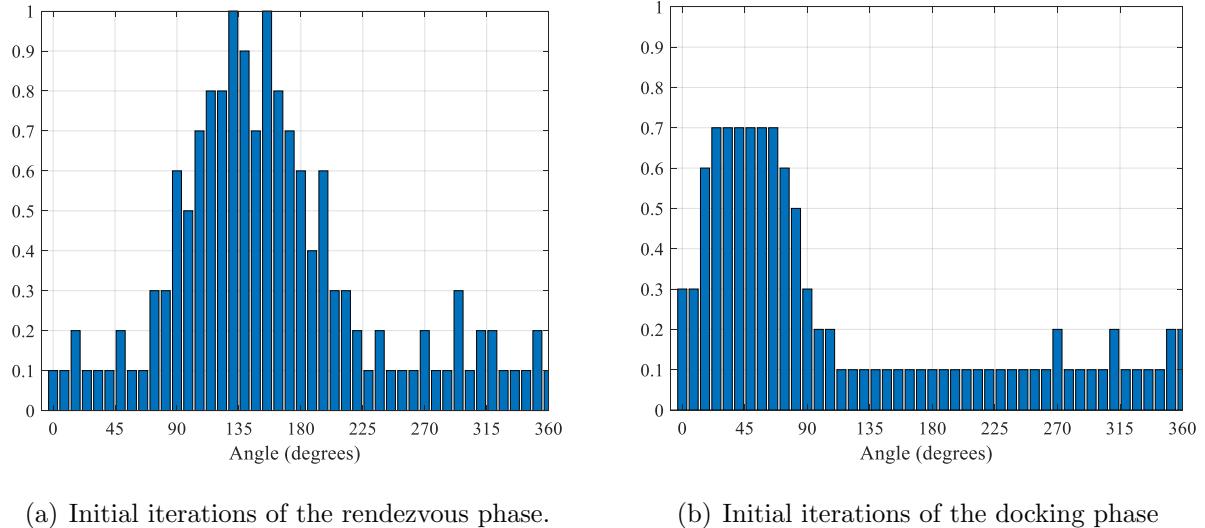
(a) Activity of the payload detector repository after learning convergence with the Agents' ALM turned on.
 (b) Activity of the proximal agent detector repository after learning convergence with the Agents' ALM turned on.

Figure 4.29: Agent detector repository activity after learning convergence with ALM on. The proximal agent detector shows strong, localized activity, while the payload detector remains inactive.

The results in Figures 4.30–4.34 show agent behavior during both the rendezvous and docking phases. At this point, training continues for the second hidden layer and the output layer. These figures illustrate how the agents adapt their actions and control strategies as learning progresses. The network gradually refines its representations and motor outputs to achieve coordinated docking and rendezvous maneuvers.

Figure 4.30 presents the activity patterns of the second hidden layer repositories during both the rendezvous and docking phases for Agent 1. In Figure 4.30a, during the rendezvous phase, the RPR shows a pronounced activity peak around 135° . This aligns with the Line-of-Sight (LOS) direction from Agent 1 to the payload. This indicates that the RPR has learned to orient its activity toward the payload, helping to maintain stable rendezvous. In Figure 4.30b, during the docking phase, the DPR repository displays a significant activity peak around 45° , which corresponds to the LOS direction from Agent 1 to the docking point. This shift shows the DPR has adapted its representation to guide the payload toward the

docking location. It highlights the dynamic reconfiguration of repository activity in response to task demands.



(a) Initial iterations of the rendezvous phase.

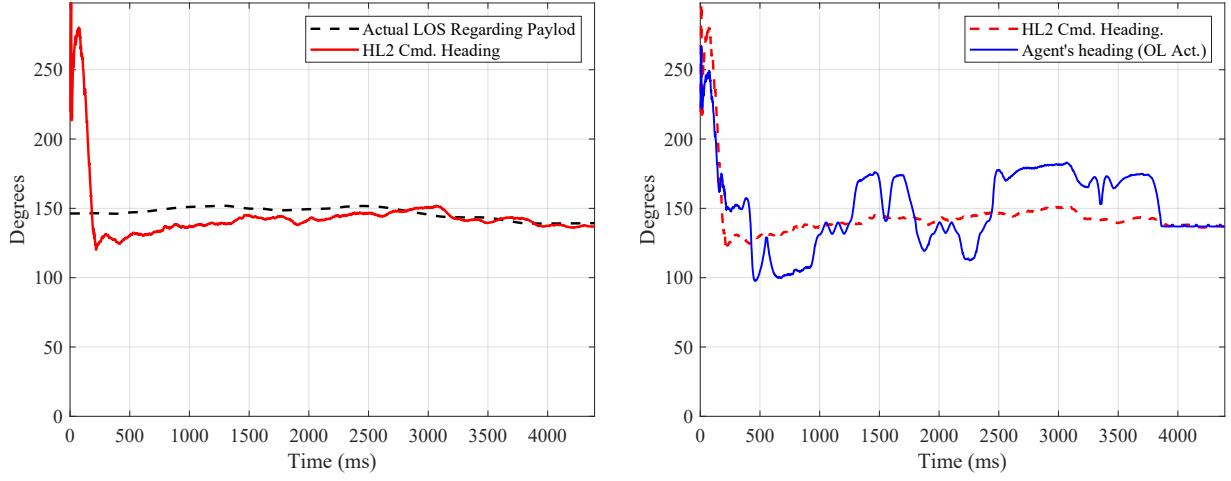
(b) Initial iterations of the docking phase

Figure 4.30: Second hidden layer activity for the rendezvous and docking phases for Agent 1: (a) RPR activity 135 degree that matches the LOS of the payload regarding the Agent 1 and (b) DPR activity shows 45 degree LOS toward the docking point.

It should be noted that the results presented in this section are generated during online learning and control, rather than from a fixed, pre-trained policy. The SNN is initialized with random synaptic weights and adapts its behavior through reward-modulated plasticity during interaction with the environment. Therefore, the results in the following figures reflect adaptation dynamics in a nonstationary setting, rather than steady-state performance after training convergence. Unlike offline training approaches, no explicit on/off switch for learning is used. The network stays adaptive throughout the mission. This allows it to accommodate nonstationary dynamics and previously unseen configurations.

Figures 4.31 and 4.32 display the activity of the second hidden layer and output layer during the rendezvous and docking phases, respectively. Figure 4.31a illustrates that, during the rendezvous phase, the activity of hidden layer 2 provides a close estimation of the actual agent's LOS with respect to the payload. After each change, the SNN requires a finite adaptation period to update its internal representation, resulting in a temporary mismatch. The

corresponding output layer activity in Figure 4.31b shows the learning process of the output layer (actual orientation executed by the agent), suggesting that the learned representations are successfully translated into control commands for rendezvous.



(a) Commanded heading generated by the Hidden layer 2 activity during the rendezvous phase vs the actual agent's LOS regarding the payload.
(b) Output layer activity during rendezvous phase

Figure 4.31: Hidden layer 2 and output layer activity during the rendezvous phase.

Figures 4.31(b) shows the agent's LOS generated by the output layer activity during the rendezvous phase. The output layer activity closely follows the patterns observed in hidden layer 2, indicating that the network effectively translates learned representations into actionable commands for maintaining rendezvous around the payload. There is a lag at the end of the rendezvous phase because the system is still learning to adjust the output-layer weights to match the hidden-layer 2 activity.

The deviations in these figures reflect transient learning dynamics. Since synaptic plasticity remains continuously active, the network adapts its internal representations whenever the LOS geometry or mission-phase context changes. In particular, abrupt changes induce a short realignment period during which the output layer adjusts to updated hidden-layer activity, resulting in temporary discrepancies that disappear as learning proceeds.

In Figure 4.32a, the activity of the second hidden layer continues to follow the actual LOS

during the docking phase, demonstrating stable tracking performance. The sharp change observed in the actual docking LOS at approximately 67 seconds is not a physical discontinuity, but rather an artifact of angle wrapping: as the LOS angle decreases past 0° , it is represented as 359° , producing the sudden upward jump seen in the dashed curve. Despite this wraparound, the second hidden layer learns to anticipate changes within the first 10 seconds of docking. The output layer activity in Figure 4.32b shows that it adapts itself with the hidden layer 2 activity, indicating successful learning of the docking maneuvers. The residual lag observed toward the end of the mission reflects ongoing online adaptation rather than incomplete learning. Since synaptic plasticity remains continuously active and no explicit learning on-off mechanism is employed, the output layer continues to adjust its weights to maintain alignment with the second hidden layer whenever the LOS geometry changes.

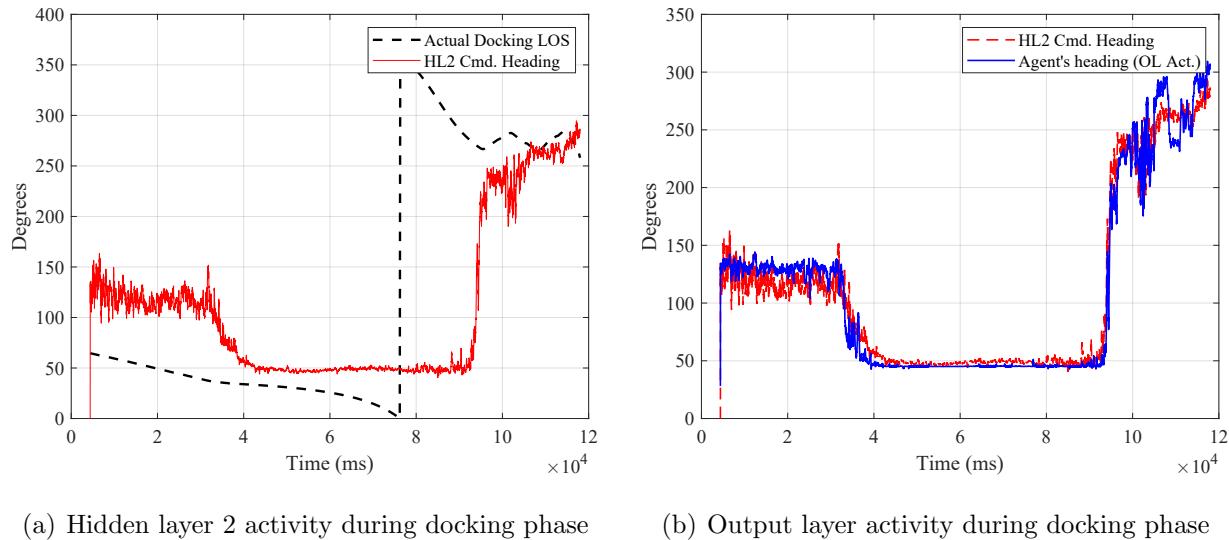
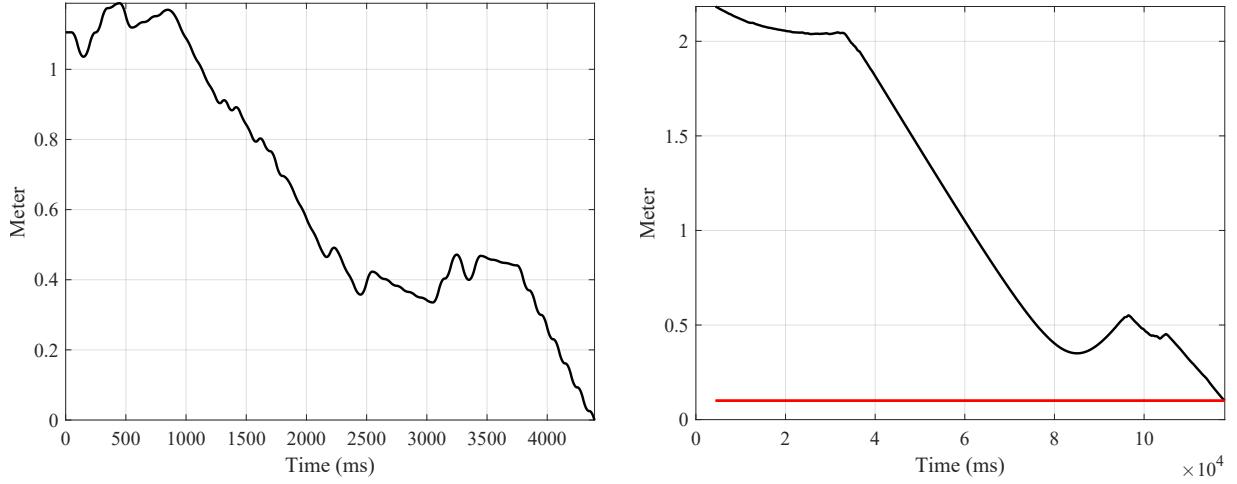


Figure 4.32: Hidden layer 2 and output layer activity during the docking phase.

Figures 4.33 illustrates the distances during the rendezvous and docking phases. In Figure 4.33a, Agent 1's distance from the payload decreases steadily during the rendezvous phase, indicating successful approach towards the payload. Figure 4.33b shows the payload's distance from the docking center during the docking phase, demonstrating effective guidance towards the target location.



(a) Agent 1's distance from the payload during rendezvous phase

(b) Payload's distance from the docking center

Figure 4.33: Distances during time.

Figure 4.34 presents the reward profiles during both the rendezvous and docking phases. In Figure 4.34a, during the rendezvous phase of Agent 1, the reward signal shows variation over time, indicating that the agent is effectively learning to minimize its distance from the payload and maintain formation. Around 3000 to 3700 ms, the reward signal exhibits a notable decrease (negative values), suggesting that the agent is refining its strategies to avoid a collision with Agent 2 as it gets closer. In Figure 4.34b, during the docking phase, the reward profile continues to improve as the payload approaches the docking center, reflecting successful navigation and alignment towards the target location. The oscillations in the reward signal towards the end of the docking phase in Figure 4.32b indicate fine-tuning of the docking maneuvers as the payload nears the docking point, which can be seen also in the training of the output layer in Figure 4.32b.

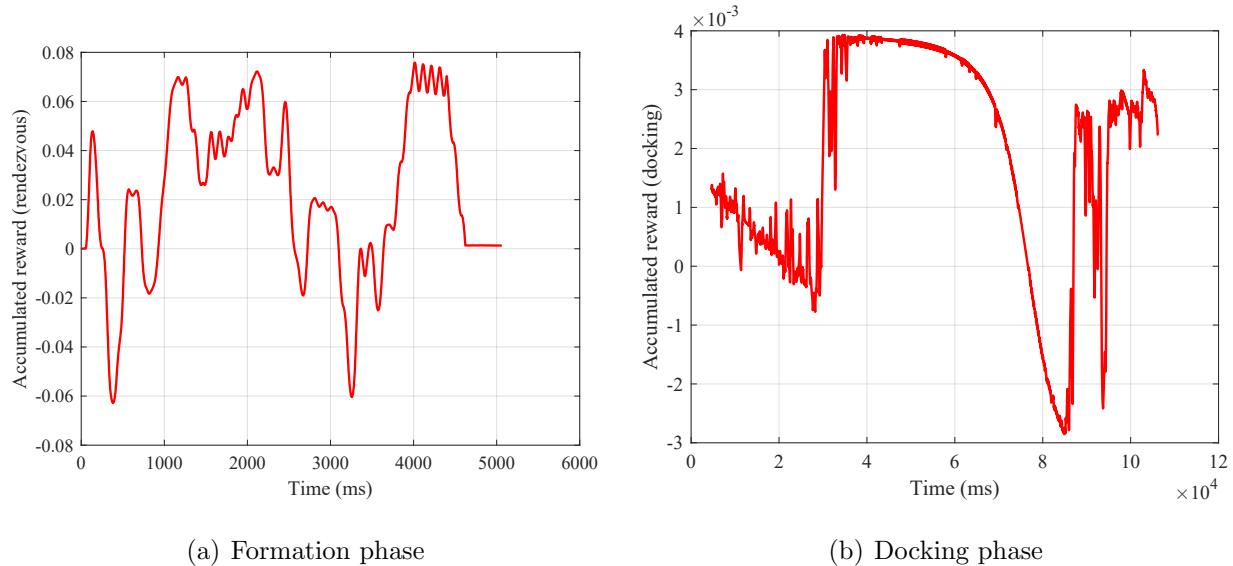


Figure 4.34: Reward profile during the docking phase

The cooperative docking sequence can be divided into three main phases: formation, docking approach, and final soft capture. Figures 4.35–4.37 illustrate these phases, showing both the agents' spatial configuration and the corresponding event-based visual responses. The yellow crosshair in each frame indicates the center of the SCS, providing a reference point for assessing alignment and positioning throughout the maneuver.

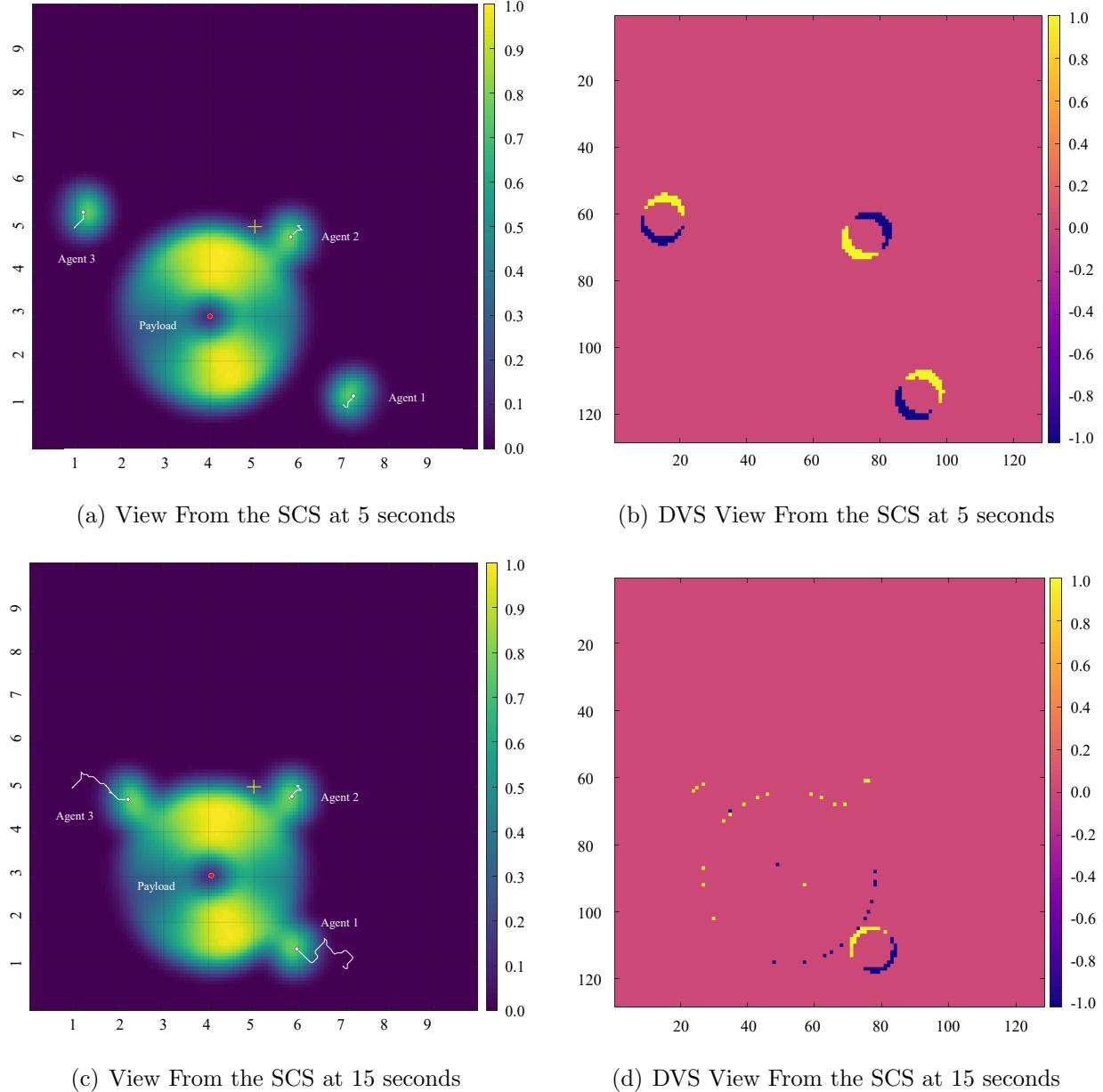


Figure 4.35: Views during the rendezvous phase at different time steps. In the DVS views, yellow regions represent the direction of motion as determined by event polarities, while white lines in (a) and (b) depict the trajectories of the payload and agents.

During the rendezvous phase, agents converge toward the payload and form a rendezvous around it (Figure 4.35). At 5 s and 15 s, the agents gradually align with the payload, reducing their relative displacement. The corresponding DVS View From the SCS records the events

triggered by agents' motion, encoding temporal luminance changes as positive and negative polarities that indicate motion direction and relative displacement.

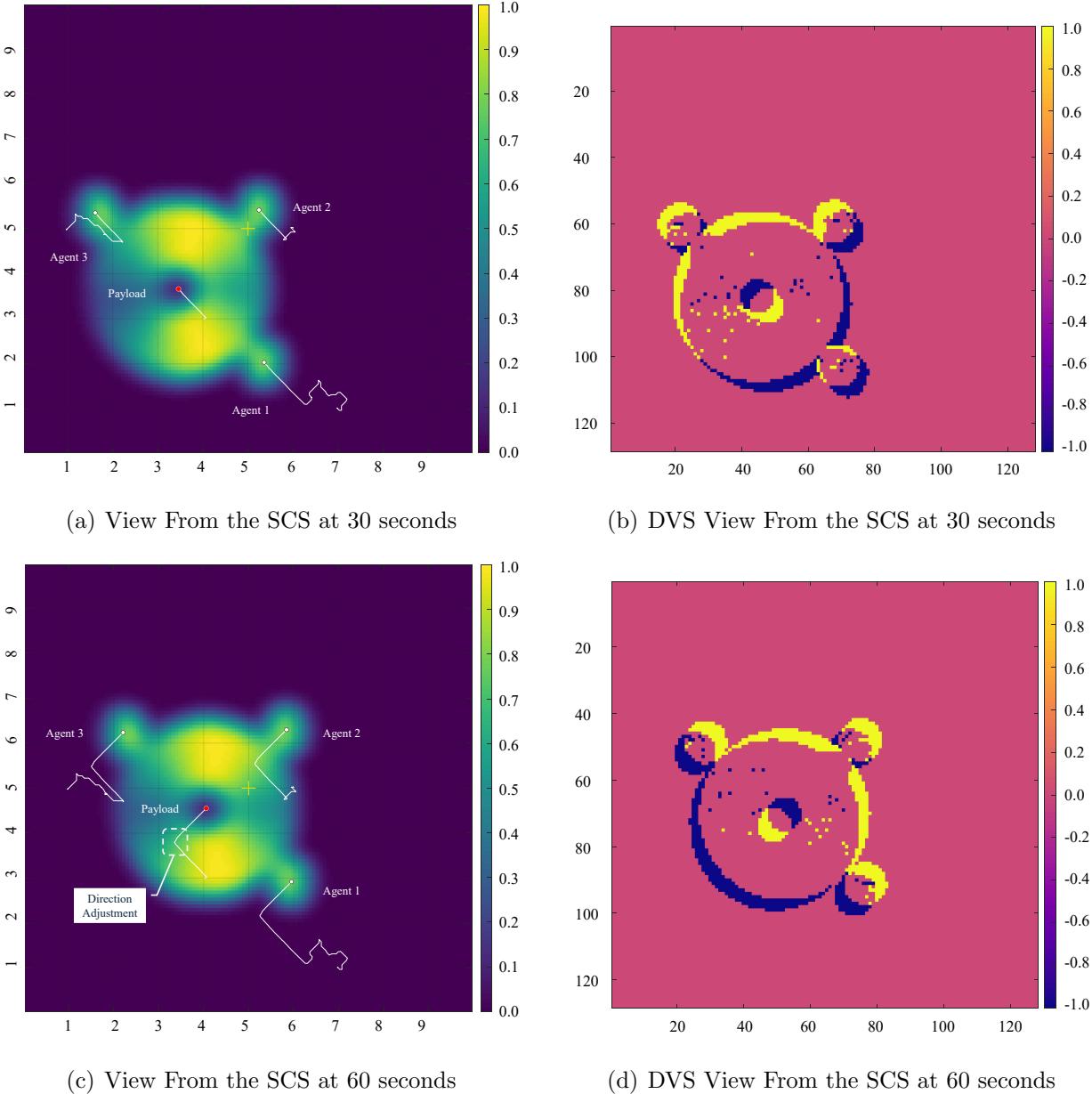


Figure 4.36: Views during the docking phase at different time steps. In the DVS views, yellow regions represent the direction of motion as determined by event polarities, while white lines in (a) and (b) depict the trajectories of the payload and agents.

In Figure 4.36(a), the rendezvous stabilizes, and the payload starts moving by the agents.

The agents begin moving the payload toward the docking center. The agents maintain synchronized thrust while compensating for small misalignments. However, after 30 seconds, the direction of movement in (a) (shown by white lines) produces a negative reward signal, making the agents adjust their thrust direction (90° change in the payload's direction) as shown in Figure 4.36(c).

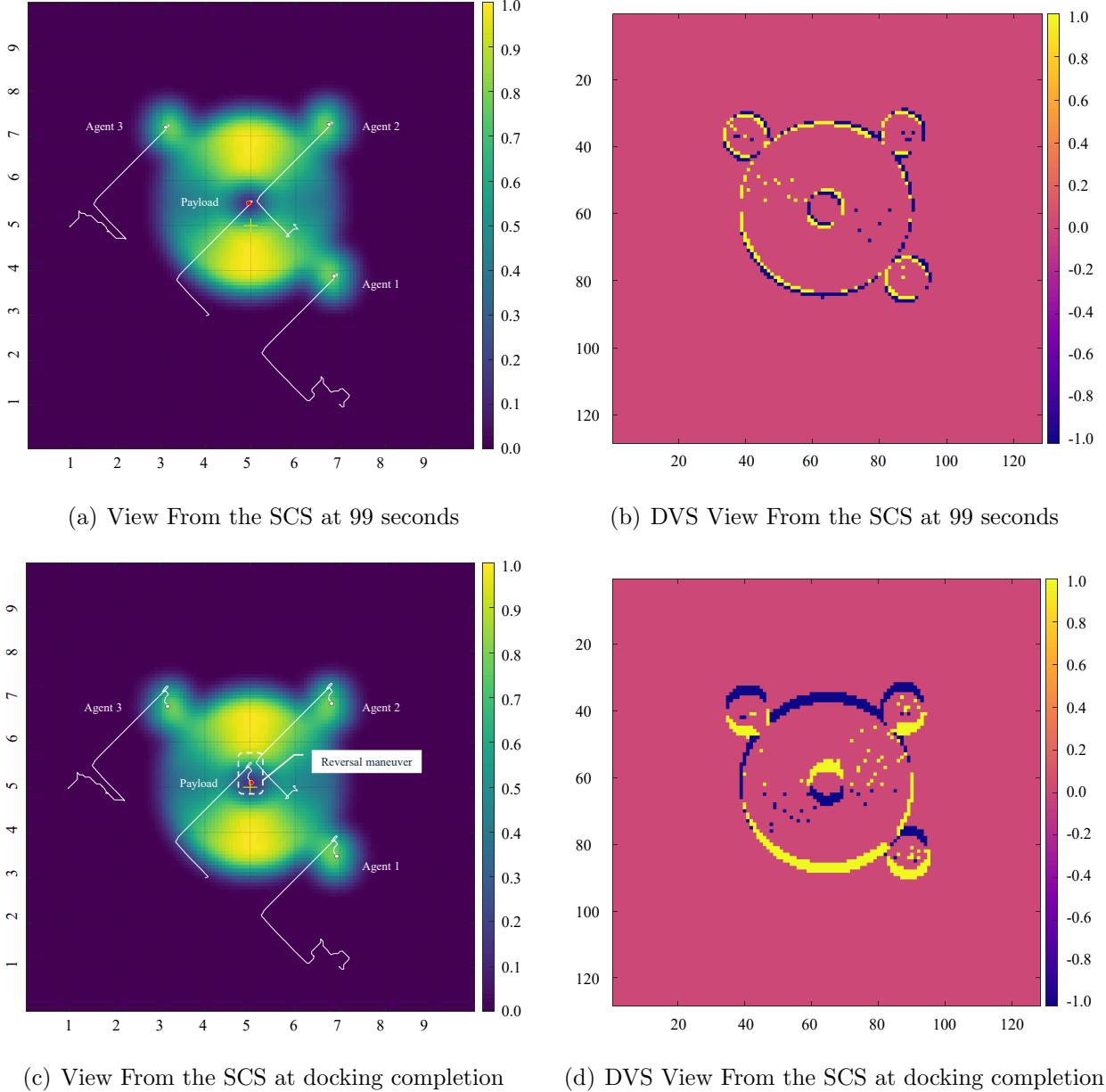


Figure 4.37: Views during the final approach and soft capture. In the DVS views, yellow regions represent the direction of motion as determined by event polarities, while white lines in (a) and (b) depict the trajectories of the payload and agents.

During the final phase of the maneuver (see Figure 4.37 (c)), the agents complete the docking sequence. At 99 seconds (Figure 4.37 (a)), they perform a controlled reversal maneuver to stabilize the payload, resulting in the inversion of event polarities due to the

change in motion direction. Once the soft capture system (SCS) ring is activated, the agents achieve full contact and alignment with the payload, meeting the soft capture criteria of the Integrated Docking System Standard (IDSS). Specifically, this requires that the misalignment be less than 0.1 meters and the lateral velocity be less than 0.04 meters per second. These results demonstrate that the proposed event-driven cooperative control framework facilitates precise spatial coordination and visually guided soft capture in accordance with IDSS standards.

4.4 Conclusion

This chapter presents a novel approach to multi-agent cooperative docking using the DVS camera integrated with SNN enhanced with adaptive light modulation. By integrating an attention mechanism with inhibitory gating, the network effectively isolates and represents both the payload and proximal agents, enabling dynamic switching of focus in response to task demands. The use of R-STDP allows the network to learn robust representations for rendezvous and docking maneuvers. Simulation results demonstrate the efficacy of the proposed architecture for achieving precise docking without direct state measurements (distances and relative velocities) under challenging lighting conditions, highlighting its potential for real-world applications in autonomous space operations.

Event-triggered mission-phase switching in the second hidden-layer repositories enables agents to adaptively transition between rendezvous and docking behaviors. The multi-scale synaptic time-constant design further enhances the network's ability to capture the temporal dynamics relevant to each phase. The ability of the SNN to learn and execute complex cooperative behaviors through local synaptic plasticity mechanisms underscores the promise of neuromorphic approaches for distributed autonomous systems, especially for controlling complex actuators such as space thrusters with MIB constraints.

The results indicate that the proposed framework can achieve the stringent requirements of the IDSS for soft capture, demonstrating its viability for future space missions. Future work will focus on hardware implementation using neuromorphic processors to validate

real-time performance and energy efficiency. Additionally, extending the framework to accommodate faults during missions and more complex docking scenarios will further test its scalability and robustness in dynamic environments.

Chapter 5

Discussion and Conclusion

In this thesis, we have explored the integration of Spiking Neural Networks (SNNs) with multiagent reinforcement learning in decentralized edge computing environments. The research has demonstrated the potential of biologically inspired neural models to enhance learning efficiency, robustness, and adaptability in dynamic multiagent systems. The key contributions of this work can be summarized as follows:

- Development of adaptive mechanisms for SNNs that stabilize learning under sparse reward conditions, balancing synaptic plasticity with network equilibrium.
- Implementation of federated learning strategies to facilitate lightweight policy sharing among agents, reducing communication overhead while preserving decentralized autonomy.
- Design and simulation of a vision-based multiagent docking framework utilizing DVS event streams and deep SNN controllers, achieving successful cooperative docking of a substantial payload within stringent performance criteria.

The results obtained from the simulations indicate that SNNs can significantly enhance the performance of multiagent systems in edge computing (FL) scenarios. The adaptive learning mechanisms developed in this thesis have shown promise in addressing the challenges posed

by sparse and delayed rewards, which are common in real-world applications. Furthermore, the developed approach has proven effective in enabling agents to share knowledge without the need for centralized coordination, thus maintaining the benefits of decentralization. The vision-based docking framework has successfully demonstrated the practical application of SNNs in complex cooperative tasks, showcasing their potential for real-time decision-making and control.

While this research has made significant strides in the integration of SNNs with multiagent reinforcement learning, several avenues for future work remain. Future research could focus on the following areas:

- **Neuron Model Diversity:** Investigating the implications of neuron model diversity within SNNs, examining how heterogeneity affects learning outcomes and system performance.
- **Advanced Optimization Techniques:** Employing methodologies such as Neuroevolution of Augmenting Topologies (NEAT) and transfer learning to optimize SNN architectures and accelerate learning processes.
- **Real-World Deployment:** Developing the FL methods to nonhomogeneous multiagent systems and testing the proposed frameworks in real-world scenarios to validate their effectiveness and robustness.

By addressing these areas, future research can further enhance the capabilities of multiagent systems, utilizing the synergies between SNNs and federated learning in dynamic and potentially adversarial environments.

Bibliography

- [1] 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.
- [2] 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.
- [3] 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.
- [4] 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.
- [5] 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.
- [6] L. Long and G. Fang, “A review of biologically plausible neuron models for spiking neural networks,” *AIAA Infotech@ Aerospace 2010*, 2010.
- [7] E. M. Izhikevich and R. FitzHugh, “Fitzhugh-nagumo model,” *Scholarpedia*, vol. 1, no. 9, 2006.

- [8] D. O. Hebb, *The organization of behavior: A neuropsychological theory*. Psychology press, 2005.
- [9] 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.
- [10] 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.
- [11] D. L. Young and C.-S. Poon, “Hebbian covariance learning: a nexus for respiratory variability, memory, and optimization?,” *Advances in Modeling and Control of Ventilation*, pp. 73–83, 1998.
- [12] E. Oja, “Simplified neuron model as a principal component analyzer,” *Journal of mathematical biology*, vol. 15, pp. 267–273, 1982.
- [13] T. Toyoizumi, J.-P. Pfister, K. Aihara, and W. Gerstner, “Generalized bienenstock–cooper–munro rule for spiking neurons that maximizes information transmission,” *Proceedings of the National Academy of Sciences*, vol. 102, no. 14, pp. 5239–5244, 2005.
- [14] A. Soltoggio and J. J. Steil, “Solving the distal reward problem with rare correlations,” *Neural computation*, vol. 25, no. 4, pp. 940–978, 2013.
- [15] H. S. Seung, “Learning in spiking neural networks by reinforcement of stochastic synaptic transmission,” *Neuron*, vol. 40, no. 6, pp. 1063–1073, 2003.
- [16] E. M. Izhikevich, “Solving the distal reward problem through linkage of stdp and dopamine signaling,” *Cerebral cortex*, vol. 17, pp. 2443–2452, 2007.
- [17] E. M. Izhikevich, “Simple model of spiking neurons,” *IEEE Transactions on neural networks*, vol. 14, no. 6, pp. 1569–1572, 2003.

- [18] 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.
- [19] H. Z. Shouval, S. S.-H. Wang, and G. M. Wittenberg, “Spike timing dependent plasticity: a consequence of more fundamental learning rules,” *Frontiers in computational neuroscience*, vol. 4, 2010.
- [20] 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.
- [21] 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.
- [22] A. Tavanaei and A. Maida, “Bp-stdp: Approximating backpropagation using spike timing dependent plasticity,” *Neurocomputing*, vol. 330, pp. 39–47, 2019.
- [23] S. Schmidgall, J. Ashkanazy, W. Lawson, and J. Hays, “Spikepropamine: Differentiable plasticity in spiking neural networks,” *Frontiers in neurorobotics*, vol. 15, 2021.
- [24] 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.
- [25] 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.
- [26] 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.
- [27] 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.
- [28] J. Liu, H. Lu, Y. Luo, and S. Yang, “Spiking neural network-based multi-task autonomous learning for mobile robots,” *Engineering Applications of Artificial Intelligence*, vol. 104, 2021.
- [29] 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.
- [30] X. Tan, J. Cao, and X. Li, “Consensus of leader-following multiagent systems: A distributed event-triggered impulsive control strategy,” *IEEE Transactions on Cybernetics*, vol. 49, no. 3, pp. 792–801, 2019.
- [31] S. Du, T. Liu, and D. W. C. Ho, “Dynamic event-triggered control for leader-following consensus of multiagent systems,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 50, no. 10, pp. 3243–3251, 2020.
- [32] J. Liu *et al.*, “Fixed-time leader–follower consensus of networked nonlinear systems via event/self-triggered control,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 11, pp. 5029–5037, 2020.
- [33] L. Cao, Z. Cheng, Y. Liu, and H. Li, “Event-based adaptive neural network fixed-time cooperative formation for multiagent systems,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 5, pp. 6467–6477, 2024.
- [34] Y. Zhang, J. Sun, H. Liang, and H. Li, “Event-triggered adaptive tracking control for multiagent systems with unknown disturbances,” *IEEE Transactions on Cybernetics*, vol. 50, no. 3, pp. 890–901, 2020.

- [35] T. T. Nguyen, N. D. Nguyen, and S. Nahavandi, “Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications,” *IEEE Transactions on Cybernetics*, vol. 50, no. 9, pp. 3826–3839, 2020.
- [36] Y. Liu, X. Wang, H. Zhang, and G. Chen, “Event-triggered control for multiagent systems with sensor faults and input saturation,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 51, no. 7, pp. 3855–3866, 2021.
- [37] Y. Wang, Y. Song, D. J. Hill, and M. Krstic, “Prescribed-time consensus and containment control of networked multiagent systems,” *IEEE Transactions on Cybernetics*, vol. 49, no. 4, pp. 1138–1151, 2019.
- [38] P. Trslić, M. Rossi, L. Robinson, C. W. O’Donnel, A. Weir, J. Coleman, J. Riordan, E. Omerdić, G. Dooly, and D. Toal, “Vision based autonomous docking for work class rovs,” *IEEE Journal of Oceanic Engineering*, 2020. Available online 2019.
- [39] A. B. Figueiredo and A. C. Matos, “Mvido: A high performance monocular vision-based system for docking a hovering auv,” *Journal of Field Robotics*, 2020.
- [40] V. N. Sankaranarayanan, A. Banerjee, S. Satpute, and G. Nikolakopoulos, “Safe docking of a payload-carrying spacecraft using state constrained adaptive control,” *Journal of Guidance, Control, and Dynamics*, 2025.
- [41] A. Nikhilraj, H. Simha, and H. Priyadarshan, “Modeling, guidance and control of quadrotor uavs on SE(3) in application to autonomous docking,” *International Journal of Robotics Research*, 2025.
- [42] E. Narváez, A. A. Ravankar, A. Ravankar, T. Emamu, and Y. Kobayashi, “Autonomous vtol-uav docking system for heterogeneous multirobot team,” *IEEE Robotics and Automation Letters*, 2021.
- [43] N. Lashkari, M. Biglarbegian, and S. X. Yang, “Development of a novel robust control method for formation of heterogeneous multiple mobile robots with autonomous docking capability,” *Robotics and Autonomous Systems*, 2020.

- [44] H. Li, H. Wang, L. Cui, J. Li, Q. Wei, and J. Xia, “Design and experiments of a compact self-assembling mobile modular robot with joint actuation and onboard visual-based perception,” *IEEE Transactions on Robotics*, 2022.
- [45] J. Lee, T. Delbruck, M. Pfeiffer, P. K. Park, C.-W. Shin, H.-Y. Ryu, *et al.*, “Real-time gesture interface based on event-driven processing from stereo silicon retinas,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 9, pp. 2250–2263, 2014.
- [46] S. Seifozzakerini, W.-Y. Yau, K. Mao, and H. Nejati, “Hough transform implementation for event-based systems: Concepts and challenges,” *Frontiers in Computational Neuroscience*, vol. 12, p. 103, 2018.
- [47] O. Bichler, D. Querlioz, S. J. Thorpe, J.-P. Bourgoin, and C. Gamrat, “Extraction of temporally correlated features from dynamic vision sensors with spike-timing-dependent plasticity,” *Neural Networks*, vol. 32, pp. 339–348, 2012.
- [48] J. C. Thiele, O. Bichler, and A. Dupret, “Event-based, timescale invariant unsupervised online deep learning with stdp,” *Frontiers in Computational Neuroscience*, vol. 12, p. 46, 2018.
- [49] B. Zhao, R. Ding, S. Chen, B. Linares-Barranco, and H. Tang, “Feedforward categorization on aer motion events using cortex-like features in a spiking neural network,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 9, pp. 1963–1978, 2014.
- [50] S. Barchid, G. M. Caron, K. Ouni, *et al.*, “Spiking neural networks for frame-based and event-based single object localization,” *Neurocomputing*, vol. 559, 2023.
- [51] L. Zanatta, A. D. Mauro, F. Barchi, A. Bartolini, L. Benini, and A. Acquaviva, “Directly-trained spiking neural networks for deep reinforcement learning: Energy efficient implementation of event-based obstacle avoidance on a neuromorphic accelerator,” *Neurocomputing*, vol. 562, 2023.

- [52] S. Yang, B. Linares-Barranco, Y. Wu, and B. D. Chen, “Self-supervised high-order information bottleneck learning of spiking neural network for robust event-based optical flow estimation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.
- [53] A. Tavanaei and A. Maida, “Bp-stdp: Approximating backpropagation using spike timing dependent plasticity,” *International Journal of Neural Systems*, vol. 28, no. 10, p. 1850032, 2018.
- [54] N. Anwani and B. Rajendran, “Training multi-layer spiking neural networks using normad based spatio-temporal error backpropagation,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 12, pp. 4939–4951, 2020.
- [55] A. Taherkhani, A. Belatreche, Y. Li, and L. P. Maguire, “A supervised learning algorithm for learning precise timing of multiple spikes in multilayer spiking neural networks,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 12, pp. 6079–6090, 2018.
- [56] Y. Kim and P. Panda, “Optimizing deeper spiking neural networks for dynamic vision sensing,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 2, pp. 674–687, 2021.
- [57] J. Kim, H. Kim, S. Huh, J. Lee, and K. Choi, “Deep neural networks with weighted spikes,” *Advances in Neural Information Processing Systems*, vol. 31, pp. 1129–1138, 2018.
- [58] J. Ding, J. Zhang, T. Huang, J. K. Liu, and Z. Yu, “Assisting training of deep spiking neural networks with parameter initialization,” *IEEE Transactions on Circuits and Systems I*, vol. 72, no. 1, pp. 56–68, 2025.
- [59] J. Lin and J.-S. Yuan, “A scalable and reconfigurable in-memory architecture for ternary deep spiking neural network with reram based neurons,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 4, pp. 1355–1370, 2020.

- [60] S. Wang, D. Zhang, A. Belatreche, Y. Xiao, H. Qing, W. Wei, M. Zhang, and Y. Yang, “Ternary spike-based neuromorphic signal processing system,” *Nature Electronics*, vol. 8, no. 1, pp. 45–58, 2025.
- [61] 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.
- [62] 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.
- [63] 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.
- [64] 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.
- [65] 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.
- [66] D. Ye, R. Yu, M. Pan, and Z. Han, “Federated learning in vehicular edge computing: A selective model aggregation approach,” *IEEE Access*, vol. 8, pp. 23920–23935, 2020.
- [67] K. Yang, T. Jiang, Y. Shi, and Z. Ding, “Federated learning via over-the-air computation,” *IEEE transactions on wireless communications*, vol. 19, no. 3, pp. 2022–2035, 2020.
- [68] Y. Zhan, P. Li, Z. Qu, D. Zeng, and S. Guo, “A learning-based incentive mechanism for federated learning,” *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6360–6368, 2020.

- [69] M. Chen, D. Gündüz, K. Huang, W. Saad, M. Bennis, A. V. Feljan, and H. V. Poor, “Distributed learning in wireless networks: Recent progress and future challenges,” *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 12, pp. 3579–3605, 2021.
- [70] M. Chen, H. V. Poor, W. Saad, and S. Cui, “Convergence time optimization for federated learning over wireless networks,” *IEEE Transactions on Wireless Communications*, vol. 20, no. 4, pp. 2457–2471, 2020.
- [71] M. A. Ferrag, O. Friha, L. Maglaras, H. Janicke, and L. Shu, “Federated deep learning for cyber security in the internet of things: Concepts, applications, and experimental analysis,” *IEEE Access*, vol. 9, pp. 138509–138542, 2021.
- [72] M. Chen, Z. Yang, W. Saad, C. Yin, H. V. Poor, and S. Cui, “A joint learning and communications framework for federated learning over wireless networks,” *IEEE Transactions on Wireless Communications*, vol. 20, no. 1, pp. 269–283, 2020.
- [73] Z. Yang, M. Chen, W. Saad, C. S. Hong, and M. Shikh-Bahaei, “Energy efficient federated learning over wireless communication networks,” *IEEE Transactions on Wireless Communications*, vol. 20, no. 3, pp. 1935–1949, 2020.
- [74] A. Jarwan and M. Ibnkahla, “Edge-based federated deep reinforcement learning for iot traffic management,” *IEEE Internet of Things Journal*, vol. 10, no. 5, pp. 3799–3813, 2022.
- [75] Z. Yuan, Z. Wang, X. Li, L. Li, and L. Zhang, “Hierarchical trajectory planning for narrow-space automated parking with deep reinforcement learning: A federated learning scheme,” *Sensors*, vol. 23, no. 8, 2023.
- [76] Q. Wu, X. Chen, T. Ouyang, Z. Zhou, X. Zhang, S. Yang, and J. Zhang, “Hiflash: Communication-efficient hierarchical federated learning with adaptive staleness control and heterogeneity-aware client-edge association,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 5, pp. 1560–1579, 2023.

- [77] G. Xia, J. Chen, C. Yu, and J. Ma, “Poisoning attacks in federated learning: A survey,” *IEEE Access*, vol. 11, pp. 10708–10722, 2023.
- [78] 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.
- [79] 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.
- [80] 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.
- [81] 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.
- [82] E. Garcia, “Cooperative target protection from a superior attacker,” *Automatica*, vol. 131, 2021.
- [83] 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.
- [84] A. D. Bird, P. Jedlicka, and H. Cuntz, “Dendritic normalisation improves learning in sparsely connected artificial neural networks,” *PLOS Computational Biology*, vol. 17, 2021.
- [85] 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.

- [86] 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.
- [87] D. E. Feldman, “Spike timing–dependent plasticity,” *Neural Circuit and Cognitive Development*, pp. 127–141, 2020.
- [88] 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.
- [89] L. C. Udeigwe, P. W. Munro, and G. B. Ermentrout, “Emergent dynamical properties of the bcm learning rule,” *The Journal of Mathematical Neuroscience*, vol. 7, no. 1, p. 2, 2017.
- [90] 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.
- [91] 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.
- [92] Z. Wang, Z. Peng, X. Fan, Z. Wang, S. Wu, R. Yu, P. Yang, C. Zheng, and C. Wang, “Fedave: Adaptive data value evaluation framework for collaborative fairness in federated learning,” *Neurocomputing*, vol. 574, 2024.
- [93] 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.

- [94] G. Gallego, T. Delbrück, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. J. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza, “Event-based vision: A survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 1, pp. 154–180, 2022.
- [95] P. G. F. Dias, M. C. Silva, G. P. Rocha Filho, P. A. Vargas, L. P. Cota, and G. Pessin, “Swarm robotics: A perspective on the latest reviewed concepts and applications,” *Sensors*, vol. 21, no. 6, 2021.
- [96] R. Graca and T. Delbrück, “Towards a physically realistic computationally efficient dvs pixel model,” *arXiv preprint arXiv:2505.07386*, 2025.
- [97] A. Botelho, P. Rosa, and J. M. Lemos, “Explicit spacecraft thruster control allocation with minimum impulse bit,” *IEEE Transactions on Control Systems Technology*, 2024.
- [98] E. M. Izhikevich, “Which model to use for cortical spiking neurons?,” *IEEE transactions on neural networks*, vol. 15, no. 5, pp. 1063–1070, 2004.
- [99] K. A. Wilmes and C. Clopath, “Dendrites help mitigate the plasticity-stability dilemma,” *Scientific Reports*, vol. 13, no. 1, p. 6543, 2023.
- [100] N. Frémaux and W. Gerstner, “Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules,” *Frontiers in neural circuits*, vol. 9, p. 85, 2016.
- [101] SpaceX, “Dragon.” Online: Available at <https://www.spacex.com/vehicles/dragon/>, 2025. Accessed: 27-Oct-2025.
- [102] NASA, ESA, JAXA, CSA, and Roscosmos, “International docking system standard (idss) – interface definition document (idd), revision f,” tech. rep., National Aeronautics and Space Administration (NASA), July 2022. Approved for Public Release. Available: <https://www.internationaldockingstandard.com/>.