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

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

# Abstract

This thesis presents a comprehensive study of multiagent reinforcement learning using Spiking Neural Networks (SNNs) for decentralized control in edge computing environments. Chapter 1 introduces the motivation and literature review, highlighting the need for biologically inspired neural models to achieve robust, energy-efficient learning in dynamic multiagent systems. Chapter 2 develops adaptive mechanisms for SNNs, focusing on stabilizing learning under sparse 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. The chapter demonstrates that event-triggered policy exchanges can improve efficiency without requiring full synchronization or centralized orchestration.

Chapter 4 delivers the main simulation results, introducing a vision-based multiagent docking framework that combines Dynamic Vision Sensor (DVS) event streams, adaptive light modulation, and deep SNN controllers. The proposed system enables a swarm of agents to cooperatively dock a 14,000 kg payload within 99 seconds, meeting IDSS soft-capture standards (lateral misalignment ¡ 0.1 m, relative lateral rate ¡ 0.04 m/s) and maintaining formation stability despite thruster minimum-impulse-bit constraints.

By integrating adaptive SNN learning, efficient policy sharing, and event-driven control, this research advances the development of resilient, responsive, and scalable multiagent systems for edge computing. The results demonstrate that neuromorphic approaches can deliver robust autonomy in complex, real-world cooperative tasks.

# Contents

# List of Figures

# List of Tables

# Acronyms

**ALM** Active LED Marker

**CDM** Collaborative Docking Mission

**DPD** Docking Phase Dominance

**DVS** Dynamic Vision Sensor

**FFM** Formation Flying Mission

**FL** Federated Learning

**FPD** Formation Phase Dominance

**GRF** Gaussian Receptive Fields

**LIF** Leaky Integrate-and-Fire

**MIB** Minimum–Impulse Bit

**N2A** Neighboring Agent Attention

**N2AD** Neighboring Agent Attention Dominance

**PA** Payload Attention

**PAD** Payload Attention Dominance

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

**RCSE** Reward-Modulated Competitive Synaptic Equilibrium

**SCS** Soft Capture System

**SNN** Spiking Neural Network

# Chapter 1

# Introduction

Spiking Neural Network (SNN)s are a class of neural models inspired by the event-driven signaling of biological neurons. Unlike conventional artificial neural networks, SNNs process information through discrete spikes, enabling efficient temporal coding and low-power operation. This event-driven paradigm makes SNNs particularly suitable for edge computing and real-time control tasks, where responsiveness and energy efficiency are critical [1, 2].

Recent advances in SNNs have demonstrated their ability to learn complex behaviors using biologically plausible mechanisms such as Reward-modulated Spike-Timing-Dependent Plasticity (R-STDP) and reward-modulated learning. These features allow SNNs to adapt to dynamic environments and perform robustly under sparse or delayed feedback, making them attractive for multiagent reinforcement learning and autonomous systems.

Federated Learning (FL) complements SNNs by enabling decentralized training across distributed devices without sharing raw data. In FL, each agent trains its local model and periodically exchanges updates with a central server or among peers, preserving privacy and reducing communication overhead. The combination of SNNs and FL supports scalable, privacy-preserving, and energy-efficient learning in multiagent systems, especially in scenarios where data locality and resource constraints are important.

The energy-efficient nature of SNNs aligns perfectly with the objectives of FL, where models are trained across a network of distributed devices without centralized data collection.

Unlike conventional artificial neural networks that require continuous data flow and computation, SNNs' event-driven operation allows for significant reductions in energy consumption, which is critical for battery-operated agents participating in FL. This characteristic of SNNs supports the development of low-power, distributed learning solutions, enabling more agents to participate in FL without compromising on power efficiency [3].

Furthermore, the integration of SNNs in FL scenarios can also help overcome limitations associated with bandwidth-constrained environments. The sparse nature of data representation and communication in SNNs means that less information needs to be exchanged between devices and the central server during the training process. This efficiency is crucial in FL environments, where network bandwidth and connectivity can significantly impact the feasibility and performance of distributed learning systems [4].

Another critical advantage of SNNs in the context of FL is their compatibility with neuromorphic hardware. Neuromorphic chips, designed to replicate the neural structures of the human brain, provide an ideal platform for deploying SNNs. This synergy between neuromorphic computing and SNNs paves the way for the development of highly efficient, scalable, and adaptive FL systems capable of leveraging the full potential of edge computing [5].

Despite these advantages, the integration of SNNs with FL presents several challenges. The primary hurdle is the complexity of training SNNs in complex problems. The dynamic and temporal nature of SNNs introduces new challenges in developing effective FL protocols that can accommodate the unique online learning rules of SNNs. Proper scheduling of communication rounds within the local SNN time steps is essential for successful collaborative training [6].

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

of FL systems powered by SNNs [7].

Additionally, communication constraints and the non-stationarity of data distribution pose significant challenges. The need for larger model update intervals to reduce communication costs and the changing nature of data over time (e.g., reward function in RL) require innovative solutions to maintain the accuracy and reliability of FL systems employing SNNs [8].

Despite these challenges, the potential benefits of integrating SNNs with FL justify continued research and development in this area. The combination of energy efficiency, privacy preservation, and compatibility with neuromorphic hardware, along with the distributed and collaborative nature of FL, represents a compelling approach to deploying machine learning models in real-world applications [9].

## 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.

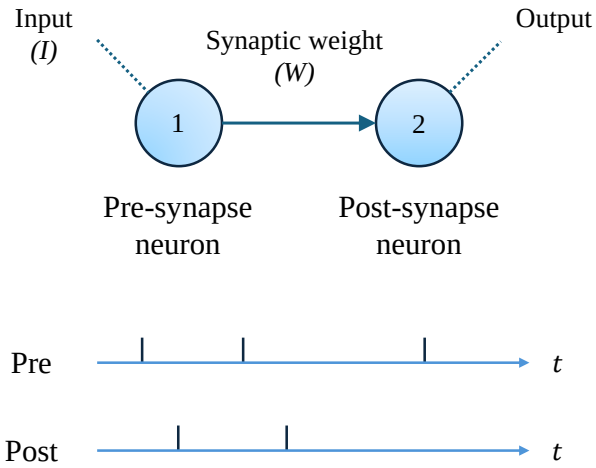



Figure 1.1: Simple model of a SNN. The spike pattern shows that the neurons spike whenever the voltage of the neuron reaches a threshold.

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** | $\dot{V}_1 = \frac{1}{\tau_m}\left[E_L - V_1 + R_m I\right]$ |
| **Post-Neuron** | $\dot{V}_2 = \frac{1}{\tau_m}\left[E_L - V_2 + R_m\left(W\delta(t - t_{\text{pre}})\right)\right]$ |

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 [10],

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

where $V_m(t)$ is the neuron's membrane potential 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 [11].

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

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

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

where $\tau_m$ is the membrane time constant.

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

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

or

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

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

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

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

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

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

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

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

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

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

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

### 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 [12]:

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

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

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):

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

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 [13]. Hebbian learning, particularly within the context of SNN, primarily revolves around the modulation of synaptic strengths based on the firing rates of pre- and postsynaptic neurons. The principles of locality and joint activity are fundamental, emphasizing that synaptic changes occur only when both neurons are active simultaneously.

**Methods of Synaptic Modification:**

- **Local Rules:** Synaptic changes are influenced directly by the activities of the connecting neurons without external influences.

- **Bounded Growth:** To avoid uncontrolled increases in synaptic strength, models typically incorporate mechanisms such as hard and soft bounds. Hard bounds prevent any further increase once a maximum weight is achieved, while soft bounds slow down the rate of increase as the maximum is approached [14].

- **Synaptic Decay:** Realistic models also consider mechanisms for reducing synaptic strengths, typically through a decay term that weakens connections in the absence of activity [15].

**Advanced Hebbian Models:**

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

- **Oja's Rule:** A self-stabilizing rule that ensures synaptic weights do not grow indefinitely by normalizing the weight vector, thereby maintaining the overall stability of the network [17].

- **BCM Rule:** The Bienenstock-Cooper-Munro rule introduces an adaptive threshold for synaptic modification, which evolves based on the historical activity of the neuron, allowing for more refined potentiation and depression based on relative activity levels. [18].

- **RCHP:** Rarely Correlating Hebbian Plasticity focuses on synaptic changes driven by rare, significant coincidences in neuronal activity, aiming to strengthen connections that are crucial for neural function while avoiding over-strengthening due to common activity patterns [19].

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

## Neo-Hebbian Learning and Modulation Mechanisms

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

**Distal rewards and credit assignment**

This model introduces a sophisticated approach to synaptic modification through the interaction of R-STDP with a modulatory signal reflective of reward, embodying the essence of TD learning within the realm of spiking neurons.

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

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

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 $\tau_C$. The Dirac delta function, $\delta(t - t^{(f)})$, signifies the occurrence of a spike, serving as a pivotal factor in the temporal credit assignment process.

The STDP function can be represented as follows,

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

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

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

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

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

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

The Izhikevich model merges the biological realism of the Hodgkin-Huxley model with the computational simplicity of integrate-and-fire models, offering a balanced approach for simulating neuronal behavior [23]. This model is particularly noted for its ability to produce

diverse firing patterns with computational efficiency, making it suitable for simulating large networks of neurons while still capturing key aspects of neuronal activity.

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

In a significant study, the distal reward problem was tackled by linking STDP with dopamine signaling, illustrating a novel approach to reward-based learning in the brain [21]. This study proposed a model where dopamine modulates STDP mechanisms, effectively capturing the essence of reward-based learning processes.

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

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

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

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

In a pioneering study, the implementation of probabilistic inference within networks of LIF neurons demonstrated how Bayesian networks can be transformed into a computable

framework for SNNs [29]. This methodology underscores SNNs' capability to perform complex cognitive functions through probabilistic reasoning.

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

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

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

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

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

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

A methodology for multi-task autonomous learning in mobile robots using SNNs was proposed, employing Modified Integrate-and-Fire and LIF neuron models, alongside a task switch mechanism inspired by lateral inhibition and a novel learning rule based on R-STDP.

This approach demonstrated SNNs' capabilities in efficiently learning and adapting across various tasks such as obstacle avoidance and target tracking [36].

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

A transfer learning algorithm for SNNs aimed at deep learning tasks was introduced, utilizing Centered Kernel Alignment for domain distance measurement and focusing on domain-invariant representation. This methodology enables effective knowledge transfer, showcasing the scalability and adaptability of SNNs [38].

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

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

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

The synchronization and coordination of multi-agent systems (MAS) are central to formation tasks in space docking missions, with leader-following consensus often achieved using event-triggered impulsive control for fast agreement without continuous communication [42], or dynamic event-triggered mechanisms with auxiliary variables to prevent Zeno behavior, where an infinite number of events are triggered within a finite time interval, and reduce bandwidth usage in spacecraft clusters [43]. Fixed-time consensus schemes ensure convergence within a specified time, supporting tightly synchronized docking maneuvers

[44]. Formation control under strict timing and resource constraints has progressed through neural-network-based and adaptive methods, including event-based adaptive neural network controllers using radial basis function networks to handle nonlinear uncertainties [45] and adaptive tracking controls that integrate disturbance observers and event-triggered thresholds for maintaining performance with minimal updates [46]. Deep learning has further enhanced MAS control, with centralized training and decentralized execution in deep reinforcement learning proving effective for complex coordination tasks such as distributed assembly and formation [47]. In contrast, event-triggered adaptive control for nonaffine dynamics and fully distributed NN-based adaptive control enable reduced communication and asymptotic synchronization in heterogeneous spacecraft formations [48, 49]. Robustness to sensor faults, input saturation, and precise timing remains essential in docking, which is achieved through event-triggered controllers using Nussbaum functions and barrier Lyapunov methods [50], alongside prescribed-time consensus frameworks that enable strict temporal adherence in docking sequences [51].

Autonomous docking has been extensively studied across marine, aerial, and space domains. Vision-based pose estimation techniques have first demonstrated the real-world docking of work-class Remotely Operated Vehicles (ROVs) to both static and dynamic transportation management system platforms, using asymmetrically arranged LED beacons and multi-step image processing integrated with PID control loops [52]. Monocular vision systems with novel attitude estimators and particle-filter trackers have similarly enabled high-accuracy docking of hovering autonomous underwater vehicles, maintaining continuous marker visibility even under partial occlusion [53]. These approaches highlight the potential of lightweight vision sensors for proximity operations where acoustic methods may be limited.

In spacecraft applications, adaptive controllers enforcing time-varying state constraints have been shown to guarantee safe Payload-carrying docking despite unknown mass variations, by employing Barrier Lyapunov Functions to bound position and velocity errors within decaying envelopes [54]. Hardware tests on planar air-bearing platforms confirmed that such

adaptive schemes outperform unconstrained methods under significant disturbances, a critical requirement for on-orbit servicing missions, where safety is paramount.

Geometric and potential-based guidance laws have enabled cooperative docking between aerial vehicles. Quadrotor pairs employed artificial potential functions for approach guidance and nonlinear 3D controllers for precise port alignment during docking, demonstrating effective mutual maneuvering in simulation [55]. Similarly, VTOL UAVs docking to mobile manipulators achieved on-site recharging through a hybrid visual-servoing and path planning framework, fusing predictive and reactive control to maintain a line of sight and ensure a secure attachment [56].

Formation control frameworks extended docking capabilities to heterogeneous mobile robot teams. A subsumption architecture combined layered kinematic objectives with robust dynamic controllers to manage autonomous docking, formation switching, and collision avoidance under model uncertainties, verified in simulation with up to a dozen robots [57]. Compact self-assembling modules with parallel-gripper docking and onboard visual perception further illustrate scalable docking and reconfiguration for modular robot chains traversing complex terrains [58].

In structured environments, sensor-fusion and learning-based methods have improved docking precision. YOLOv7-based detection, fused with LiDAR alignment, enabled robust industrial mobile robot recharging, achieving real-time recognition and sub-decimeter positioning in manufacturing settings [59]. Regression-based monocular systems trained on LiDAR ground truth can influence ArUco marker distortions to estimate distance and orientation with centimeter-level accuracy, vastly outperforming SolvePnP methods while using only low-cost cameras [60].

While these model-based and vision-driven controllers provide strong performance in defined scenarios, proximity missions in space docking, characterized by time-varying dynamics, unforeseen disturbances, and energy constraints, require more adaptive, data-driven control architectures. SNNs trained via the Reinforcement Learning (RL) paradigm can learn end-to-end docking policies from sensory inputs, offering fast inference, on-chip adaptability, and

robustness to unmodeled effects [61].

RL paradigms in SNNs enable agents to learn control policies through trial-and-error interactions, a capability crucial for autonomous proximity operations in space docking. Bio-plausible learning rules, such as Spike-Timing-Dependent Plasticity (STDP) combined with reward signals, have been shown to approximate backpropagation and support multi-layer learning [62–64]. By incorporating eligibility traces and global reward feedback, these methods allow SNNs to adjust synaptic weights in response to docking success metrics, facilitating rapid adaptation to new docking trajectories.

Scaling RL-trained SNN architectures to deep networks requires addressing vanishing spike activity and ensuring information propagation across layers. Techniques such as Spike Activation Lift Training (SALT) and Switched-BN enhance spike throughput in very deep SNNs, improving learning capacity under RL-like reward cues [65]. Likewise, weighted spikes encode multi-bit information per event, reducing decision latency, an essential factor for timely thruster commands during docking [66]. These advances pave the way for deep SNNs that learn complex docking maneuvers through RL while maintaining low energy consumption.

Proximity missions impose strict hardware constraints on power, area, and latency. Neuromorphic signal-processing systems that employ ternary spike encoding and quantized SNNs achieve substantial memory and energy savings, making on-board RL plausible [67, 68]. In-memory ReRAM-based architectures with ternary deep SNNs further reduce data movement and support fast synaptic updates under RL protocols [68]. Such hardware platforms can host RL-trained SNN controllers directly on spacecraft avionics, offering robust, low-power autonomy.

Accurate temporal and spatial precision is vital for docking proximity control. Fractional-order STDP (FO-STDP) enhances gradient descent in SNNs, improving convergence speed and accuracy in gesture-like input scenarios analogous to docking sensor streams [69]. Biologically plausible multi-spike learning algorithms integrate synaptic delay training to fine-tune event timing, enabling precise control of approach velocities and alignment [64].

Optimizing the training pipeline further accelerates RL convergence in SNNs. Parameter initialization based on neuronal response asymptotes prevents vanishing gradients, enabling deeper RL-trained SNNs to learn docking policies from sparse reward signals [70]. Modeling ANN-SNN conversion residuals as additive noise yields low-latency networks that maintain policy fidelity under time constraints [71]. Hybrid training methods that jointly optimize membrane leak, thresholds, and weights reduce inference timesteps to a few ms[72].

These advances in RL-trained SNNs, spanning learning rules, deep architectures, neuromorphic hardware, temporal precision, and training optimizations, provide a unified foundation for developing energy-efficient, low-latency controllers for space docking proximity missions. Dynamic Vision Sensor (DVS) cameras can supply sparse, event-driven input streams to these SNN controllers, further enhancing their responsiveness and energy efficiency in docking scenarios.

DVSs offer a fundamentally different paradigm from conventional frame-based cameras by asynchronously encoding pixel-level brightness changes as a sparse stream of events, yielding microsecond-scale temporal resolution and minimal data redundancy. This event-driven operation allows DVS to detect fast motions with latencies as low as 3 milliseconds, while consuming only a fraction of the power needed by CMOS imagers [73,74]. The high dynamic range (up to 120 dB) of DVS further enables reliable operation under extreme illumination variations without saturating the sensor [74, 75]. These attributes are particularly advantageous for proximity operations in space docking, where rapid, energy-efficient sensing of relative motion is critical.

SNNs naturally complement DVS by processing event streams natively through time-encoded spikes, enabling highly efficient feature extraction and pattern recognition. Early work demonstrated that the STDP can robustly learn temporal correlations from raw DVS data, extracting motion patterns such as vehicle trajectories with minimal synaptic complexity and high noise tolerance [75,76]. Architectures with localized lateral inhibition have been shown to learn hierarchical features in a single pass, achieving an accuracy above 96 % with only one presentation of the data [76,77].

Recent advances have expanded DVS–SNN systems to more complex applications relevant to autonomous navigation and control. For instance, deep convolutional SNNs trained with surrogate gradients achieve competitive object localization accuracy with up to 126× energy savings compared to ANNs [78, 79]. In RL scenarios, directly-trained SNNs on neuromorphic accelerators have shown superior generalization for UAV obstacle avoidance, reaching goals 98 % of the time while consuming an order of magnitude less energy than CNN counterparts [35, 79]. Event-based action recognition using novel benchmarks, such as DVS-Gesture-Chain, highlights the inherent temporal processing advantages of SNNs, enabling feed-forward networks to perceive event sequences without recurrent connections [79, 80].

For space docking missions, precise detection of geometric features such as planar edges and circular apertures is essential for alignment and approach control. Event-based adaptations of the Hough Transform implemented in SNNs allow real-time line detection and 3D parameter estimation directly on DVS outputs with no external memory or CPU, significantly reducing latency and power consumption [74]. Self-supervised information bottleneck learning further enhances event-based optical flow estimation, improving robustness to noise and reducing energy usage by over 90 % compared to ANN methods [81]. These capabilities suggest that DVS–SNN pipelines can provide the low-latency, low-power sensing and processing required for autonomous proximity operations in on-orbit docking scenarios.

FL aggregation methods form a cornerstone of how distributed models learn collectively while maintaining the privacy of local data. The seminal aggregation method in FL is FedAvg, or Federated Averaging. As introduced by McMahan et al. [82], FedAvg operates by averaging the weights of models updated locally by clients. This simple yet effective approach allows a global model to benefit from diverse local updates without sharing the data itself.

Addressing heterogeneity in client data distributions, the FedProx method [83], introduced by Li et al., modifies the local optimization process. FedProx introduces a proximal term which moderates the deviation of local updates from the global model, aiming to stabilize training across clients with non-IID data.

The Scaffold method [84] deals with statistical heterogeneity and client drift, issues prevalent in scenarios where client data is not Independent and Identically Distributed (IID). Karimireddy et al. propose using control variates that correct the local updates toward the global objective, which helps in reducing variance in updates and aligning clients closer to the global model.

Introduced by Wang et al., the FedNova method [85] employs a novel normalization mechanism in the aggregation process that adjusts based on the number of local updates performed by each client. This design accommodates the diversity in client update contributions, particularly beneficial in non-IID data scenarios.

The MOON method, or Model-contrastive Federated Learning, minimizes the contrastive loss between local and global models to maintain consistency across updates [86]. By focusing on minimizing differences in model parameters using a similarity metric, MOON seeks to preserve the quality of the global model despite the diversity of local datasets.

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

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

The limitations imposed by on-device training of SNNs, due to the constrained data availability on each device, is studied in [1]. Based on the results, the limitation can be mitigated through cooperative training utilizing FL. An online FL-based learning rule, termed

FL-SNN, for networked on-device SNNs is introduced. Through this scheme, local feedback signals are utilized within each SNN instead of backpropagation, and global feedback is facilitated through communication via a base station, showcasing the potential to significantly enhance training over isolated approaches by enabling a flexible compromise between communication load and accuracy through selective synaptic weight exchange.

A novel approach of selective model aggregation in FL within the context of Vehicular EDGE Computing (VEC) is explored in [87]. This approach focuses on the unique challenges posed by the diversity in image quality and computational capacity of vehicular clients for image classification. The approach, which selectively aggregates 'fine' local DNN models based on an evaluation of local image quality and computation capability without compromising client privacy, employs two-dimensional contract theory for model selection due to the central server's unawareness of these local parameters. This methodology is shown to outperform conventional federated averaging in terms of accuracy and efficiency on the MNIST and BelgiumTSC datasets, also offering higher utility at the central server.

The paper by [88] discusses a novel approach for fast global model aggregation in FL via cloud computation, aiming to address the primary bottleneck of limited communication bandwidth in the aggregation of locally computed updates across devices with strict latency and privacy requirements, like drones and smart vehicles. This approach, employing a sparse and low-rank optimization problem solution through a Difference-of-Convex-functions algorithm for joint device selection and beamforming design, demonstrates significant algorithmic advantages and performance improvements in numerical results.

According to [89], FL is suggested as a solution to the challenges faced in training machine learning models on IoT data due to constraints in network bandwidth, storage, and privacy. The focus is on how existing work has primarily centered on learning algorithms' convergence time, leaving issues like incentive mechanisms largely unexplored. A deep reinforcement learning-based incentive mechanism is proposed to motivate edge nodes towards model training contribution, with its efficiency validated through numerical experiments.

In [3], the exploration of synergies between wireless communications and artificial intel-

ligence, particularly the challenges of implementing ML models on battery-powered devices connected via bandwidth-constrained channels, is discussed. The paper highlights how FL for distributed training of SNNs and the integration of neuromorphic sensing with SNNs can help overcome these challenges.

The research presented in [90] introduces a FL method designed for training decentralized and privacy-preserving SNNs, focusing on the energy efficiency advantages of SNNs in FL scenarios across energy-constrained devices. The method's effectiveness is experimentally validated through improved accuracy and energy efficiency in comparison to ANNs on CIFAR10 and CIFAR100 benchmarks.

A comprehensive study on deploying distributed learning over wireless edge networks, highlighting the challenges and emerging paradigms like FL and distributed inference, is provided in [91]. It presents an overview of communication techniques for efficient deployment and future research opportunities within wireless networks.

The effect of user selection and resource allocation on the convergence time and accuracy of FL over wireless networks was investigated in [92]. A probabilistic user selection scheme is proposed alongside the use of ANNs to estimate unselected users' local FL models, showing a reduction in convergence time and improvement in model accuracy through simulations.

Federated deep learning approaches for cybersecurity in IoT applications are reviewed and experimentally analyzed in [93]. The article highlights the superiority of FL in preserving IoT device data privacy and enhancing attack detection accuracy across various deep learning models.

The challenge of training FL algorithms over wireless networks is studied in [94], where the impact of wireless factors on training quality and the optimization of learning, wireless resource allocation, and user selection to minimize FL loss function are discussed. Simulation results demonstrate the effectiveness of the proposed framework in improving identification accuracy.

In [95], an investigation is conducted into the allocation of energy-efficient transmission and computation resources for FL over wireless communication networks. The model

considered involves users training local FL models with their data and sending these to a base station (BS), which then aggregates these models and broadcasts the aggregated model back to the users. The optimization problem formulated aims to minimize total energy consumption under a latency constraint, with an iterative algorithm proposed that derives closed-form solutions for various parameters at every step. Initial feasible solutions are generated through a bisection-based algorithm aimed at minimizing completion time. It is shown through numerical results that energy consumption can be reduced by up to 59.5% compared to conventional FL methods.

In [8], a lead federated neuromorphic learning (LFNL) technique is proposed, designed as a decentralized, energy-efficient, brain-inspired computing method utilizing SNNs. This technique allows edge devices to collaboratively train a global model, preserving privacy and significantly reducing both data traffic and computational latency, with only a slight accuracy loss reported. The efficacy of LFNL in reducing energy consumption while maintaining competitive recognition accuracy, despite uneven dataset distribution, is demonstrated through experimental results.

The aggregation strategies in FL through a comprehensive mathematical convergence analysis, leading to the derivation of novel aggregation algorithms that adapt model architecture by differentiating client contributions based on loss values, is presented in [96]. This approach, which extends beyond theoretical assumptions to practical performance evaluations, is compared with the conventional FedAvg method across both IID and non-IID settings without additional hypotheses.

An edge-based backhaul (BH) selection technique aimed at improving traffic delivery by leveraging multiobjective feedback, utilizing advantage-actor-critic deep reinforcement learning methods, is introduced in [97]. To enhance DRL training performance in large-scale IoT system deployments, FL is applied, enabling collaboration among multiple edge devices. The effectiveness of this federated DRL approach in solving the BH selection problem is verified through extensive simulations.

In [98], a hierarchical trajectory planning method named HALOES, which employs deep

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

The hierarchical FL paradigm of HiFL is discussed in [99], which aims to mitigate communication overhead in FL systems by integrating synchronous client-edge and asynchronous edge-cloud model aggregations. An enhanced design, HiFlash, incorporates adaptive staleness control and a heterogeneity-aware client-edge association strategy through deep reinforcement learning, demonstrating improved system efficiency, communication reduction, and maintained model accuracy in experiments.

A survey on poisoning attacks and defense strategies in FL from a privacy-preserving perspective, classifying and analyzing poisoning attacks and defense mechanisms, is provided in [100]. The paper highlights the urgency of defending against poisoning attacks and suggests potential research directions for both attack and defense strategies, emphasizing the need for systematic reviews in this area.

# Chapter 2

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

## 2.1 Introduction

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

The neural structure in SNNs helps us implement complex learning systems. One good example of a complex situation is a differential game like the Active Target Defense (ATD) problem [101].

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

across disciplines like economics and biology. Additionally, these games find applications in Cybersecurity for modeling attacker-defender interactions [102].

There are two distinct outcomes to the ATD game, characterized by two termination sets [103]. 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 [103]. 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 [104].

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

## 2.3 Learning using R-STDP

R-STDP is a biological learning algorithm that is believed to underlie certain learning mechanisms in the brain [105]. The R-STDP algorithm is based on the idea that if a pre-synaptic

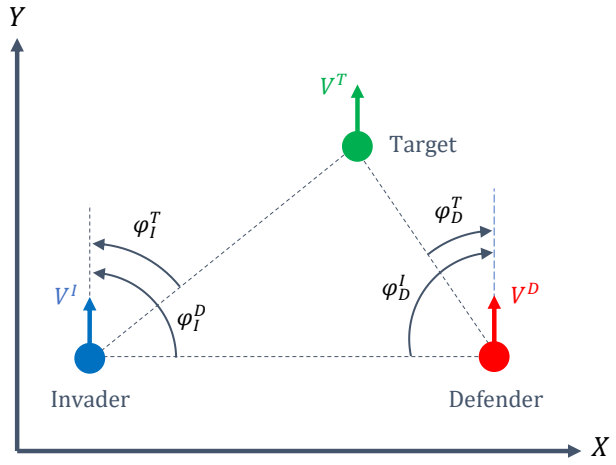



Figure 2.1: Active target defense game with three agents (LOS angles are shown for both agents).

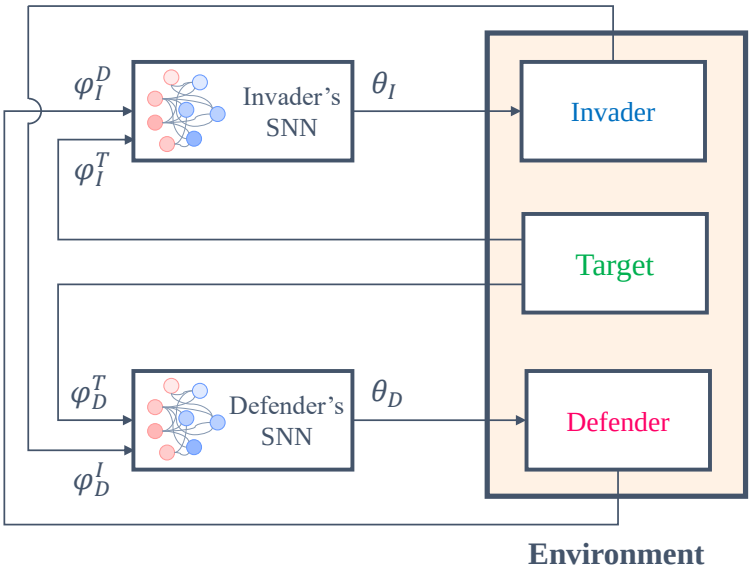



Figure 2.2: Two SNNs simultaneously play and control the invader and defender. Each agent uses the LOS angle and relative velocities for training.

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

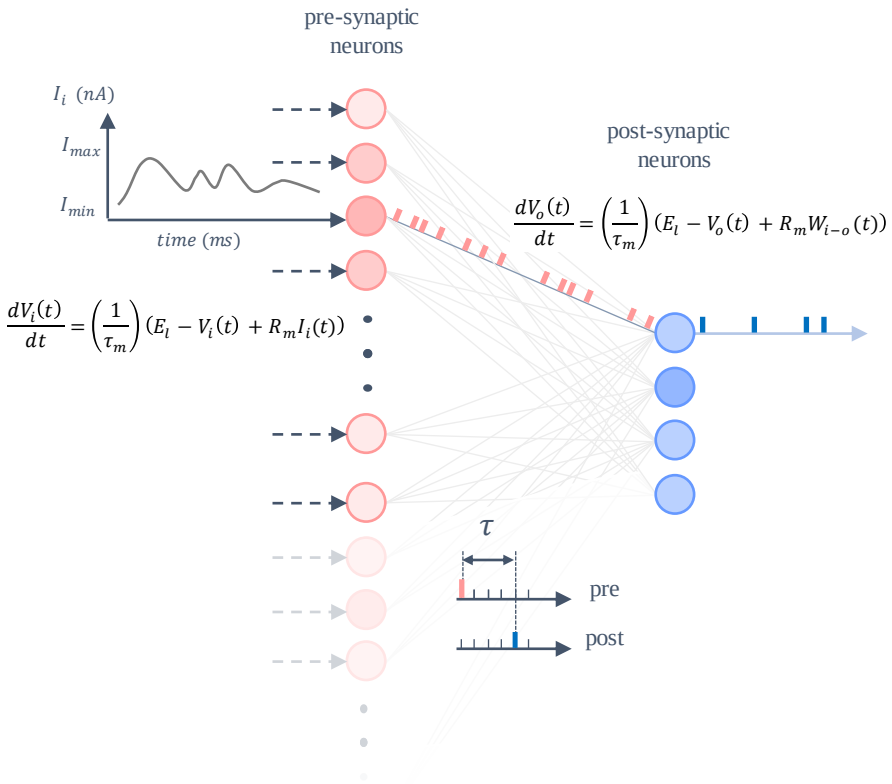



Figure 2.3: An illustration of input current to SNN and synaptic current to the output neurons after training for a single connection.

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

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

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

output neurons as follow,

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

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

The synaptic weights are changed according to the following equation,

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

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

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

The MSN normalizes the synaptic weights based on the cumulative input synaptic weights and the maximum input as follows [105],

$$\overrightarrow{W}^n(t) = \overrightarrow{W}^n(t-1)\left(\frac{I^{max}}{\Sigma_{\iota=1}^{N} W^{\iota}(t)}\right) \tag{2.4}$$

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

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

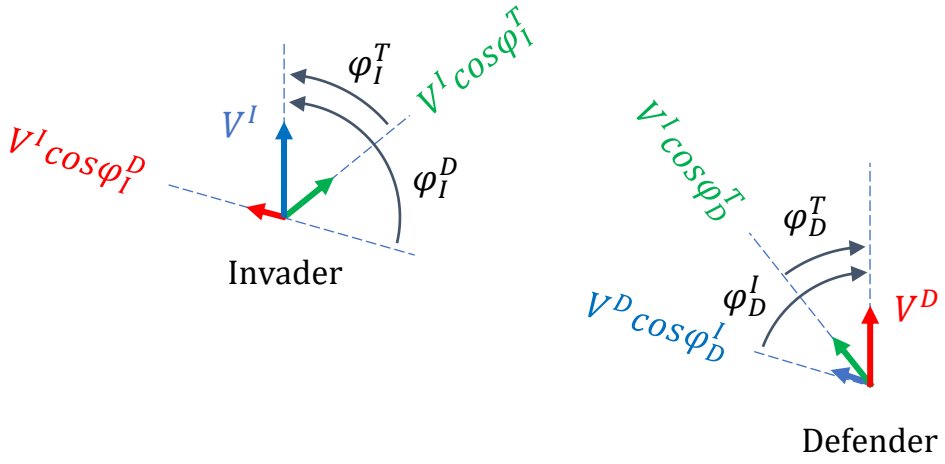



Figure 2.4: Relative velocities and LOS angles used in the reward function

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

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

$$R_I^T(t) = \eta_I^T \left(V^I + V^T\right) cos\left(\phi_I^T\right) \tag{2.5}$$

and

$$R_I^D(t) = \eta_I^D \left(V^D - V^I\right) cos\left(\phi_I^D\right) \tag{2.6}$$

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

$$R_D^T(t) = \eta_D^T \left(V^D + V^T\right) cos\left(\phi_D^T\right) \tag{2.7}$$

and

$$R_D^I(t) = \eta_D^I \left(V^D - V^I\right) cos\left(\phi_D^I\right) \tag{2.8}$$

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

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

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

and

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

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

## 2.4 Network structure and encoding method

Figure 2.5 shows the defender's network structure and encoding process. The invader has the same network with different inputs. The network receives the LOS angles and converts the inputs to Fuzzy Membership Values (FMV) using Gaussian Receptive Fields (GRF) [106]. Since the fuzzy membership values are used just for encoding data into the SNN, the type of the membership function does not affect the computation complexity.

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

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

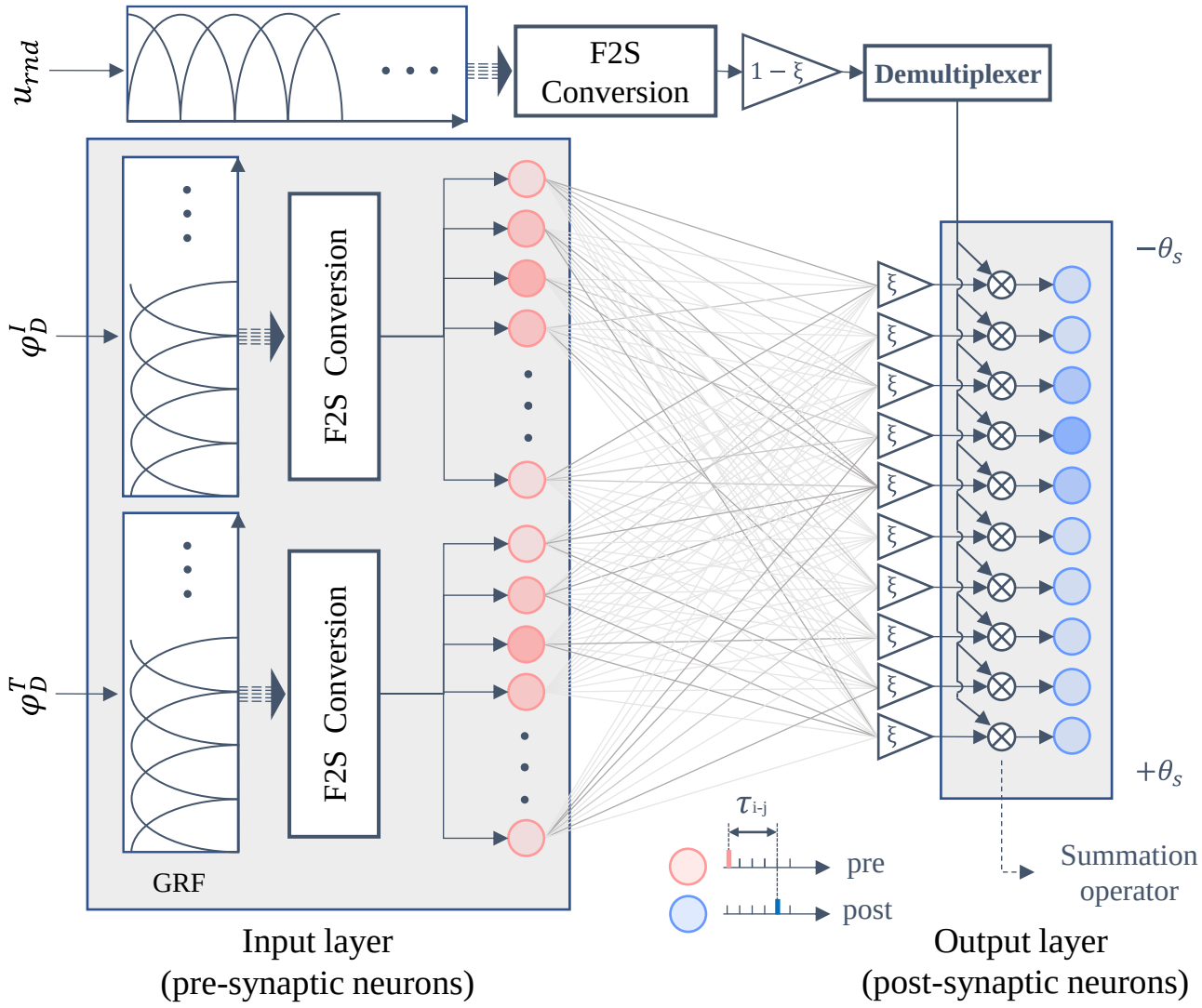



Figure 2.5: Network structure and encoding process for the input layer (Defender). Each neuron is associated with a membership function in GRF. The GRF encodes an input State ($S^t$) at each time step. There is both a training phase when ($\xi = 0$) and an operating phase when ($\xi = 1$).

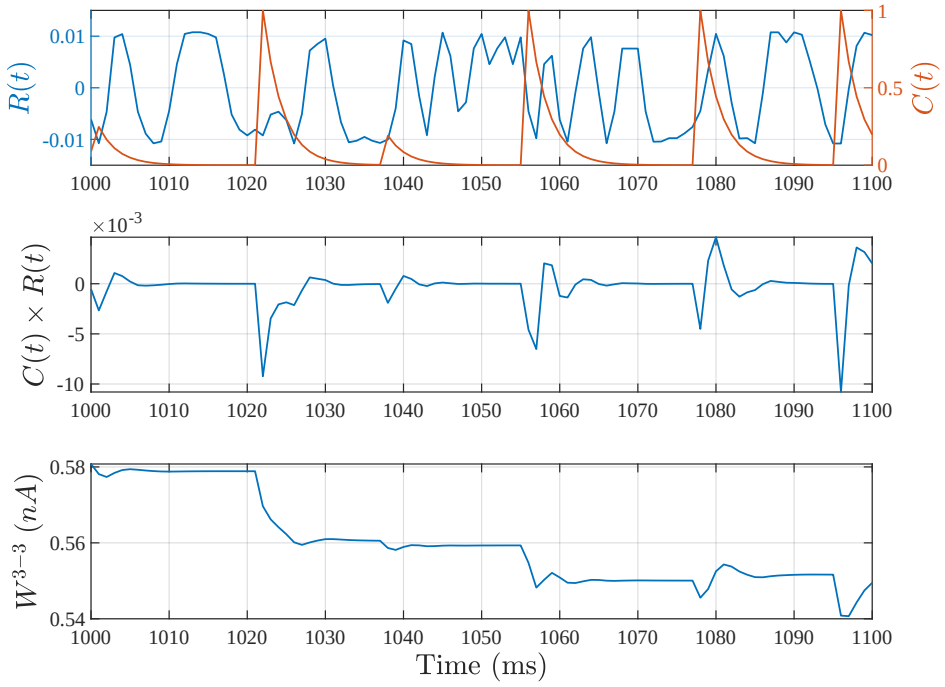



Figure 2.6: Reward, eligibility trace, and weight change during the simulation for $W^{3-3}$. The $C(t)R(t)$ changes the synaptic weight by considering activation strength and reward value.

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

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

or

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

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

**Algorithm 1** Weight training algorithm

**for** $t = 0 : \Delta t : t_{final}$ **do**
  **input** $\phi_I^T$ and $\phi_I^D$
  $\zeta \leftarrow$ number of input neurons
  $\kappa \leftarrow$ number of output neurons
  **for** i = 0:$2\pi/\zeta$:$2\pi$ **do**
    $Z_T(i,1) = exp\left(-0.5(\phi_I^T - i)/\sigma\right)$
    $Z_D(i,1) = exp\left(-0.5(\phi_I^D - i)/\sigma\right)$
  **end for**
  $I^T = (I^{max} - I^{min})\, Z_T + I^{min}$
  $I^D = (I^{max} - I^{min})\, Z_D + I^{min}$
  Generate a uniform random number for exploration $u_{rnd} \in [-\theta_s, \theta_s]$
  $j = 1$
  **for** i = $-\theta_s$:$2\theta_s/\kappa$:$\theta_s$ **do**
    $Z_{rnd}(j,1) = exp\left(-0.5(u_{rnd} - i)/\sigma\right)$
    $j \leftarrow j + 1$
  **end for**
  $I_{rnd} = (I^{max} - I^{min})\, Z_{rnd} + I^{min}$
  **for** j = 1 to $\kappa$ **do**
    $I_{syn}(j,1) = \sum_{i=1}^{\zeta} W^{i-j}\delta(t - t_i)$ ▷ $t_i$ is the firing time of the $i^{th}$ input neuron
  **end for**
  $I_{in} = \begin{bmatrix} I^T \\ I^D \end{bmatrix}$ Input current
  $I_{out} = \xi I_{syn} + (1 - \xi) I_{rnd}$ ($\xi = 0$ in training phase)
  $\mathbf{V_m}(t + \Delta t) = (1 - \frac{\Delta t}{\tau_m})\mathbf{V_m}(t) + \frac{\Delta t}{\tau_m}\left(\mathbf{E_l} + R_m \begin{bmatrix} I_{in} \\ I_{out} \end{bmatrix}\right)$ {$\mathbf{V_m}$ and $\mathbf{E_l}$ are $[\zeta + \kappa] \times 1$}
  Find fired neurons in input and output layer
  Calculate the $\boldsymbol{\tau}$ matrix that shows the difference in firing time between the fired input neurons and fired output neurons (Figure 2.5)
  Calculate $\mathbf{R - STDP}$ for all connection using (2.2)
  Calculate $\mathbf{C}$ matrix using (2.1) for all the connections
  Calculate $\mathbf{W}$ matrix using (2.3) for all the connections considering reward from (2.5) to (2.8)
  **for** i = 1 to $\kappa$ **do**
    **if** sum of the input weights to i$^{th}$ neuron $\geq I^{max}$ **then**
      Normalize input weights of i$^{th}$ neuron using (2.4)
    **end if**
  **end for**
  Set voltage of the fired neurons to reset voltage ($V_{res}$)
**end for**

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

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

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

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

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

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

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

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

## 2.5 Results

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

Table 2.1: Parameter values for LIF neuron model [9]

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

Table 2.2: Parameter values for R-STDP

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

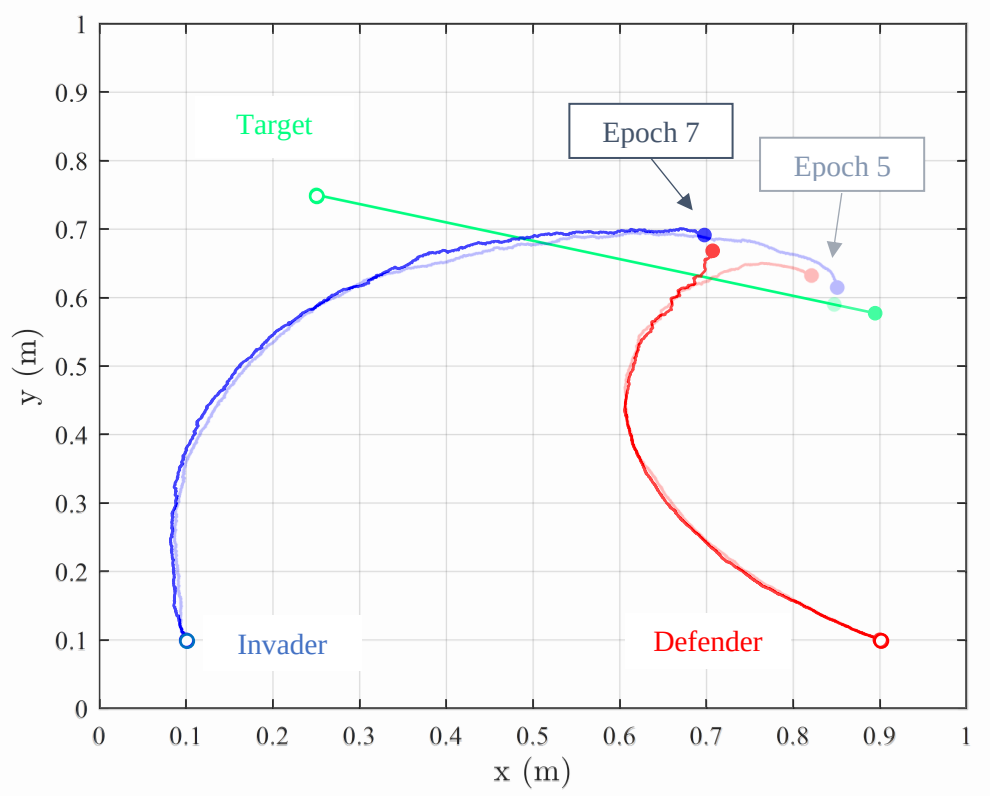



Figure 2.7: SNN's performance during training. Hollow circles show the initial positions.

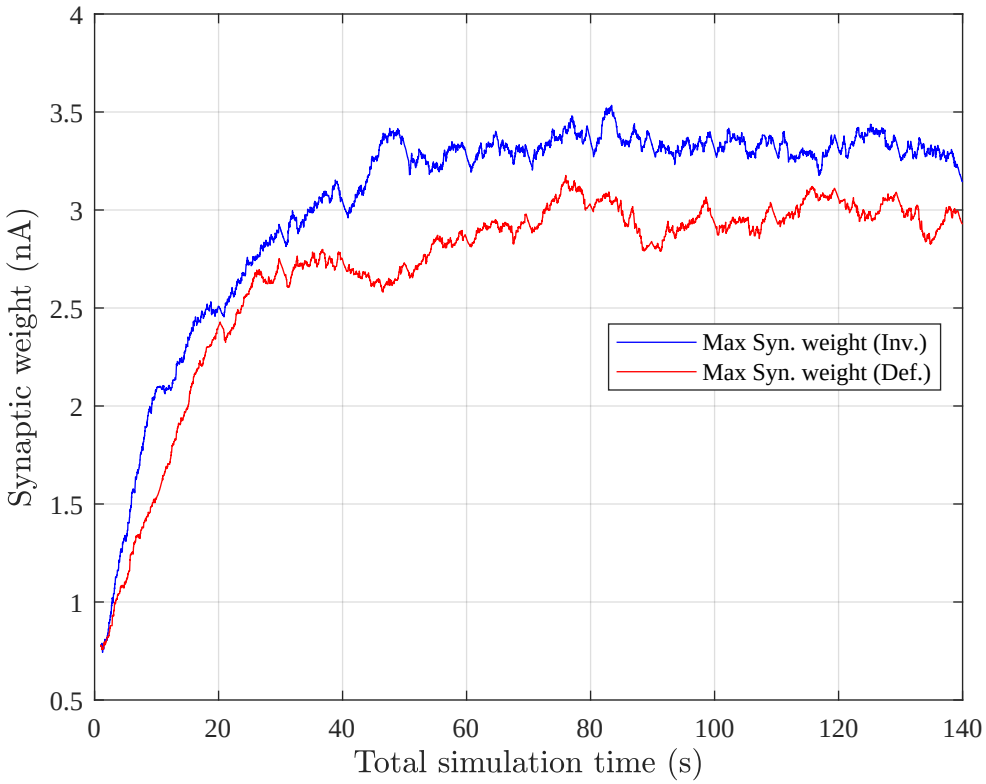



Figure 2.8: Changes in synaptic weights during training. Only Maximum synaptic weights for both agents are shown.

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

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

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

The target's velocity is 0.15 $m/s$. The invader's velocity is 0.3 $m/s$. The $\gamma$ is 0.75, so the defender's velocity is 0.225 $m/s$. The defender's capture radius ($\rho$) 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 $\sigma$ is set to 1.25. The parameter values for R-STDP, shown in Table 2.2, are set through several simulations.

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

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

### 2.5.1 Simulation without noise

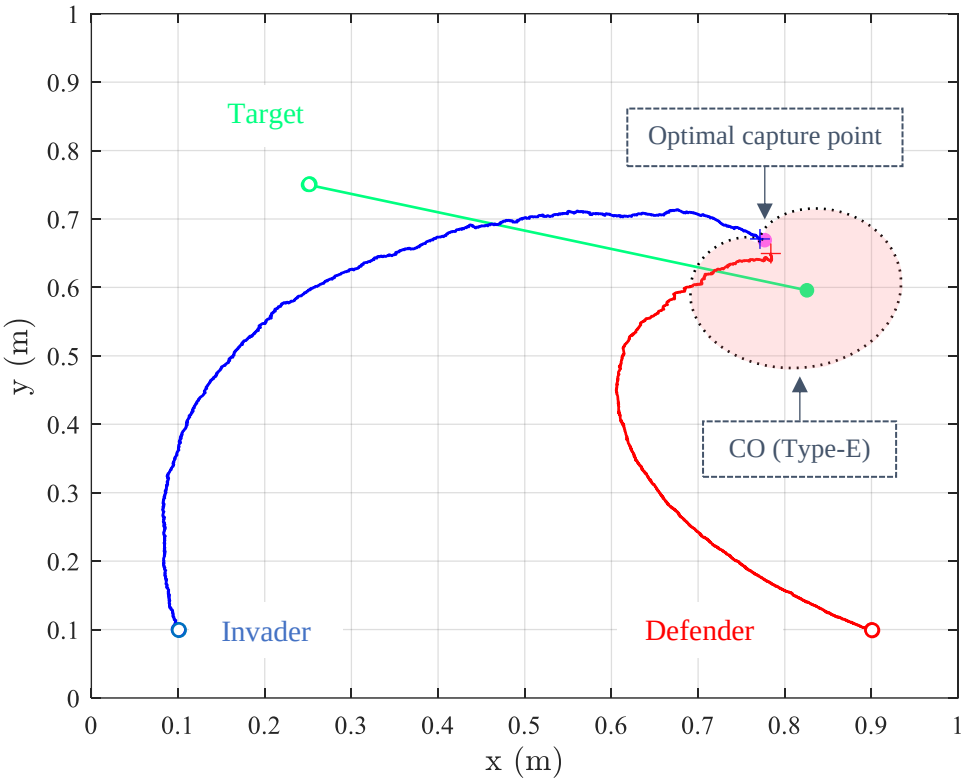



Figure 2.9: SNN's performance after training. The highlighted region shows the defender's dominant region. The purple dot is the optimal capture point. The CO Type-E shows the reachable regions of the Invader and the Defender.

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

According to Figure 2.9, the invader's SNN can find the optimal [103] capture point for the invader, while the defender's SNN can protect the moving target against a superior invader. It should be noted that this solution is obtained without a global reference frame.

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

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

### 2.5.2 Simulation with noise

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

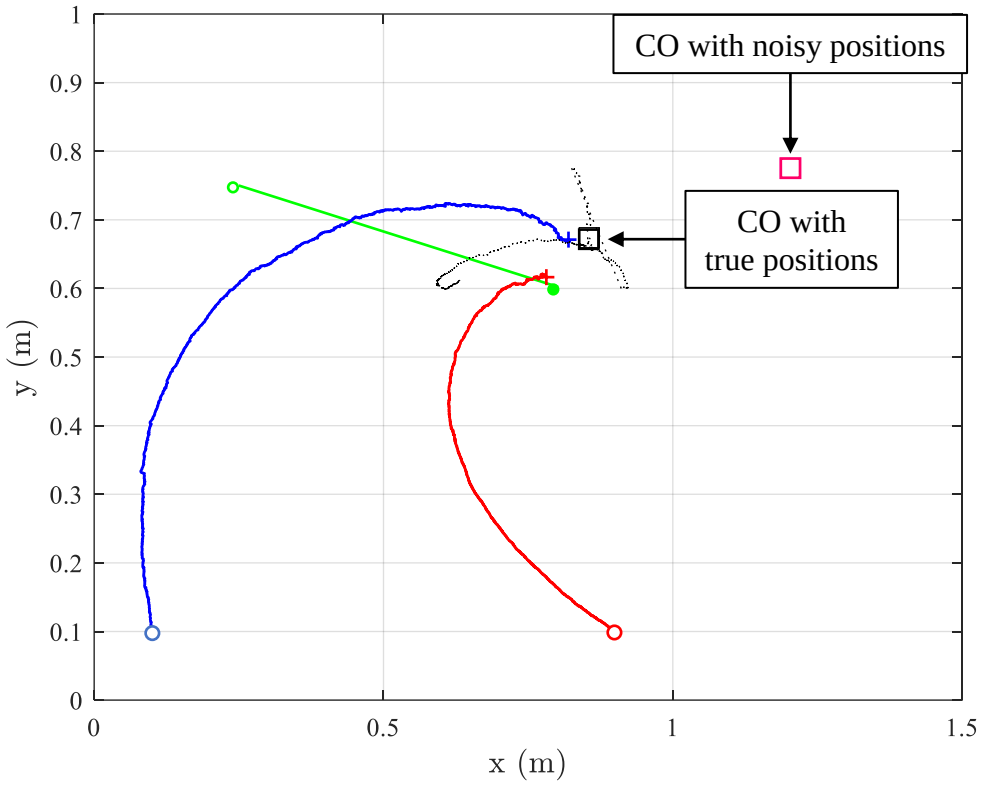



Figure 2.10: SNN's performance in noisy conditions. A white Gaussian noise with a variance of 0.01 is added to the measured inputs.

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

## 2.6 Conclusion

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

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

# Chapter 3

# Integration of R-STDP and Federated Learning

## 3.1 Consensus Flying Problem

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

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

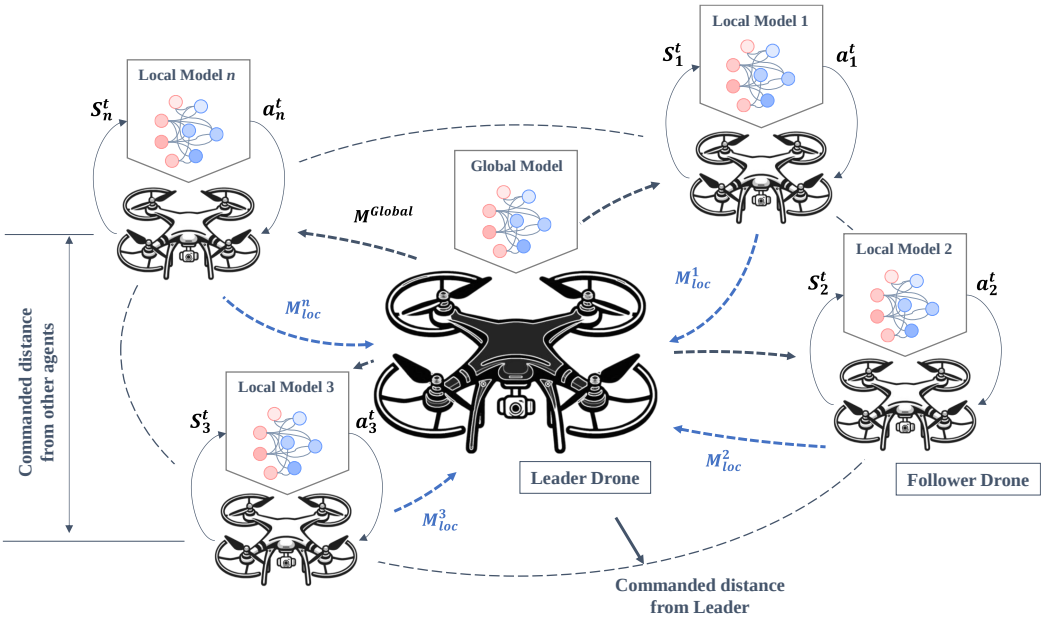



Figure 3.1: The central server (the leader) and the surrounding follower agents (white drones). The follower agents learn to fly in a formation to maintain the commanded distance. The Local models trained individually by follower agents are sent to the leader. The leader aggregates the models and sends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

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

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

- The chapter presents a comprehensive method for stabilizing and enhancing the learning process in SNN. This method focuses on controlling the unbounded growth of synaptic weights in SNNs, utilizing a strategy that dynamically adapts to changes in reward conditions and coefficients. It introduces a decay rate and learning rate adjustment based on the status of synaptic weights and enhances the responsiveness of the SNN weights to reward change.

- In terms of advancements in FL with R-STDP, the chapter addresses the FL challenges in the R-STDP framework. It introduces an event-triggered mechanism for model publishing and receiving within the network, improving network traffic. Additionally, the chapter implements a novel weighted aggregation method on the server. This method calculates weights based on the time of arrival of the models, effectively tackling the asynchronous issues in FL.

## 3.2 Proposed Method

### 3.2.1 Network Structure

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

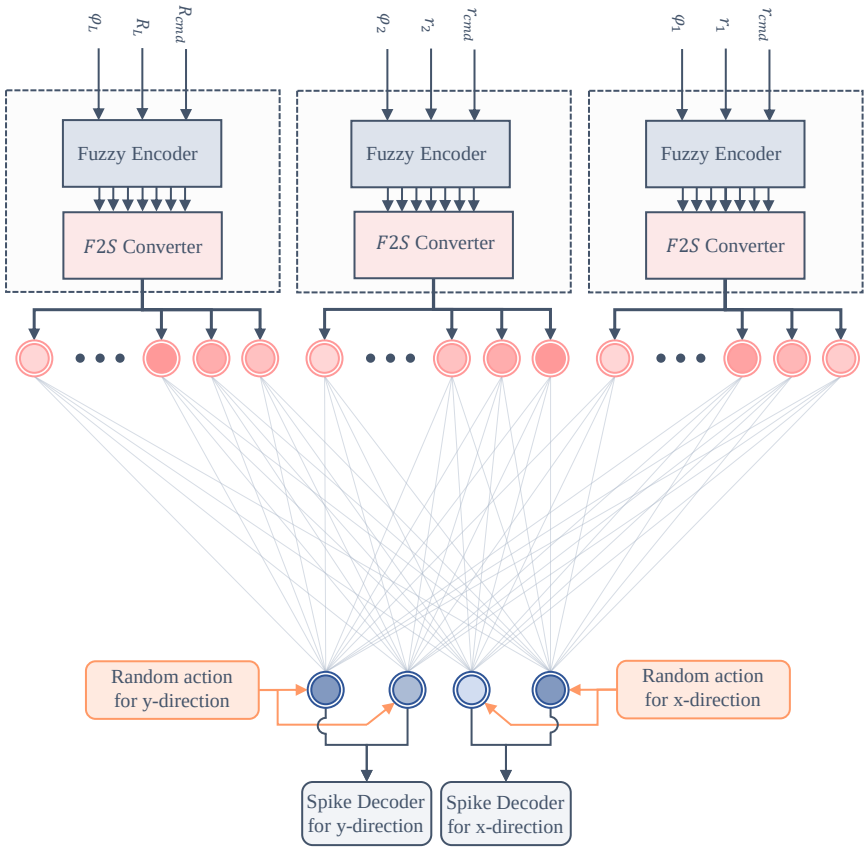



Figure 3.2: SNN structure with encoding and decoding layers. Each sub-layer consists of a fuzzy encoder and the F2S Converter, with the output layer receiving inputs from synaptic weights and a random action selector. During the training phase, the output layer receives input only from the random action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

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

encoding function for the input layer is expressed as follows,

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

where $\zeta$ and $\sigma$ are the Gaussian membership functions' center and standard deviation. The $r_i$ is the distance from the corresponding agent, $\phi_i$ is the LOS angle, and $\mu_I$ is the vector of the membership degrees. Here, $r$ is a placeholder that can either represent $r_{cmd}$ or $R_{cmd}$, depending on the context. The firing strengths from fuzzy encoders are then converted to the spiking input based on the neuron model as follows [107],

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

or

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

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

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

$$\mu_{Ox} = \frac{\Delta x}{\Delta X_{max}} \tag{3.4}$$

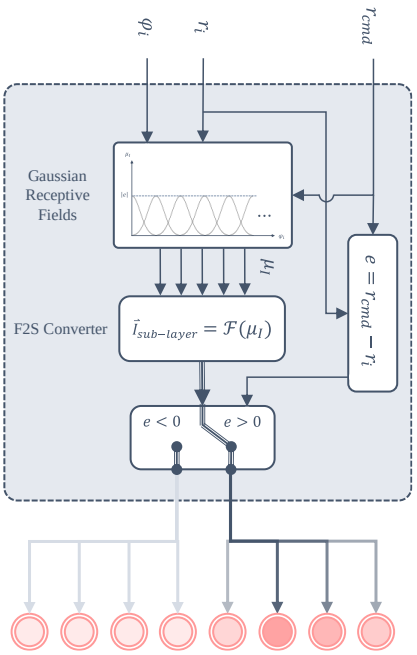



Figure 3.3: The fuzzy encoding principle for the input sub-layer.

$$\mu_{Oy} = \frac{\Delta y}{\Delta Y_{max}} \tag{3.5}$$

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

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

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

The equation for decoding this activity can be expressed as:

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

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:

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

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

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

$$\amalg_t = \gamma \cdot \amalg_{t-1} + (1 - \gamma) \cdot \Upsilon_t \tag{3.8}$$

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

### 3.2.2 Training algorithm

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**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 $W^{\mathcal{S}}_{max}(t)$. Let's assume that $I^{max} = 15.5$ and $\epsilon$ is a small positive number (e.g., 0.0001).

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

**Case 1:** $W^{\mathcal{S}}_{max}(t) < \Psi^{\mathcal{S}}$

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

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

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

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

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

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

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

**Case 2:** $W^{\mathcal{S}}_{max}(t) > \Psi^{\mathcal{S}}$

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

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

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

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

**Case 3: When $\Psi^{\mathcal{S}}$ increases from 7.75 to 15.5 (Reward Change)**

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

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

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

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

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

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

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

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

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

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

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

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

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.

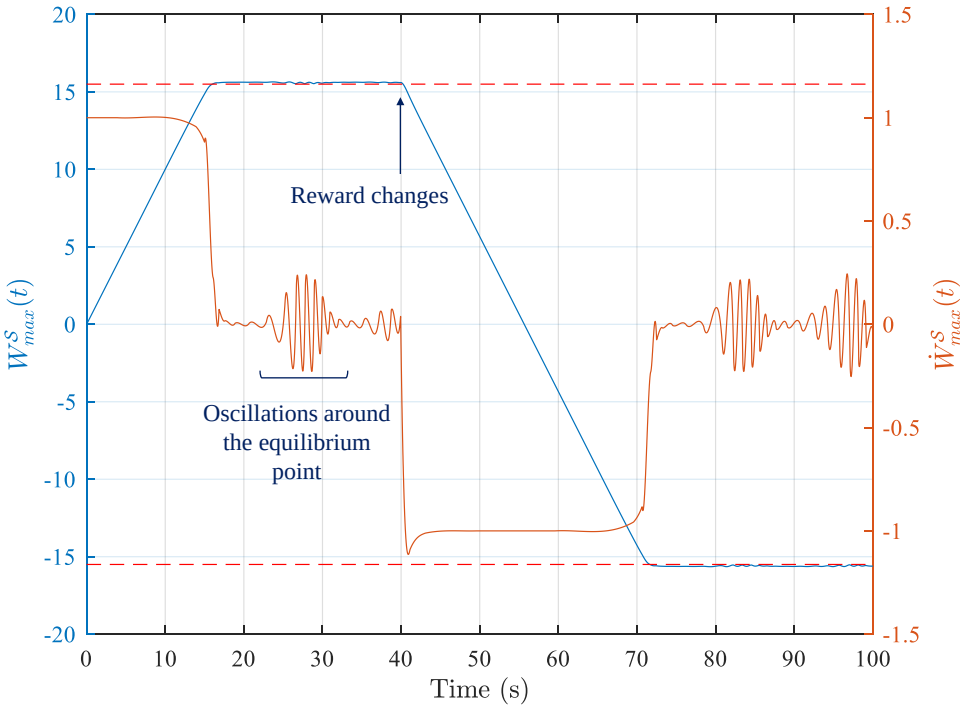



Figure 3.4: Synaptic weight change for $\lambda = 5, \Psi^{\mathcal{S}} = 15.5$,and $\mathcal{AR}^{G}_{max} = 1$.

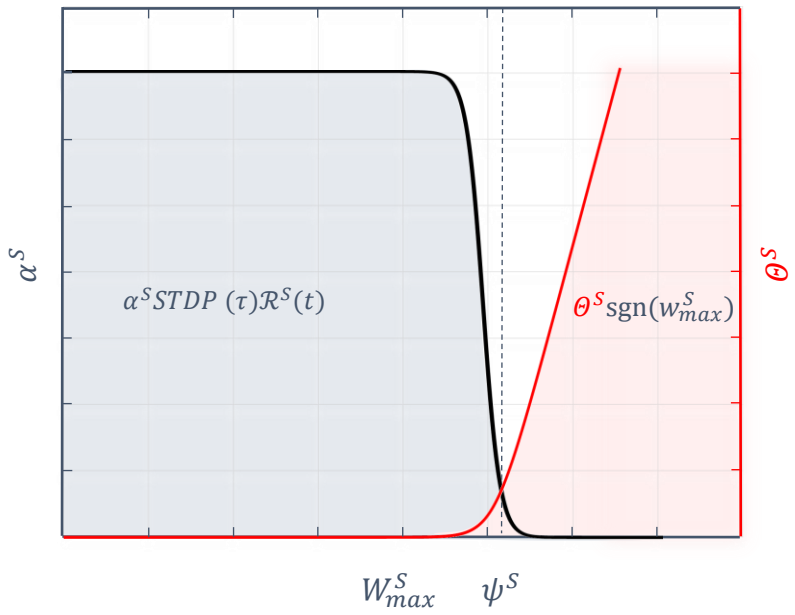



Figure 3.5: Reward-based learning rate and decay rate functions. In the blue region (active learning rate), the reward adjusts the weights, and in the red region (active decay rate), the RCSE method controls synaptic growth.

Figure 3.4 demonstrates the performance of the RCSE when it regulates the synaptic weights to prevent unbounded growth. The red dotted line represents the value of $\Psi^{\mathcal{S}}$, which is derived from the maximum reward value of the corresponding network section. As shown in Figure 3.4, the synaptic weight oscillates around $\Psi^{\mathcal{S}}$, and when it drops below $\Psi^{\mathcal{S}}$, 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 $W^{\mathcal{S}}_{max} \leq \Psi^{\mathcal{S}}$, synaptic weights in set $\mathcal{S}$ increase. If the reward changes, $\Psi^{\mathcal{S}}$ also changes. Depending on the current value of $W^{\mathcal{S}}_{max}$, the RCSE either increases or decreases the synaptic weights within set $\mathcal{S}$.

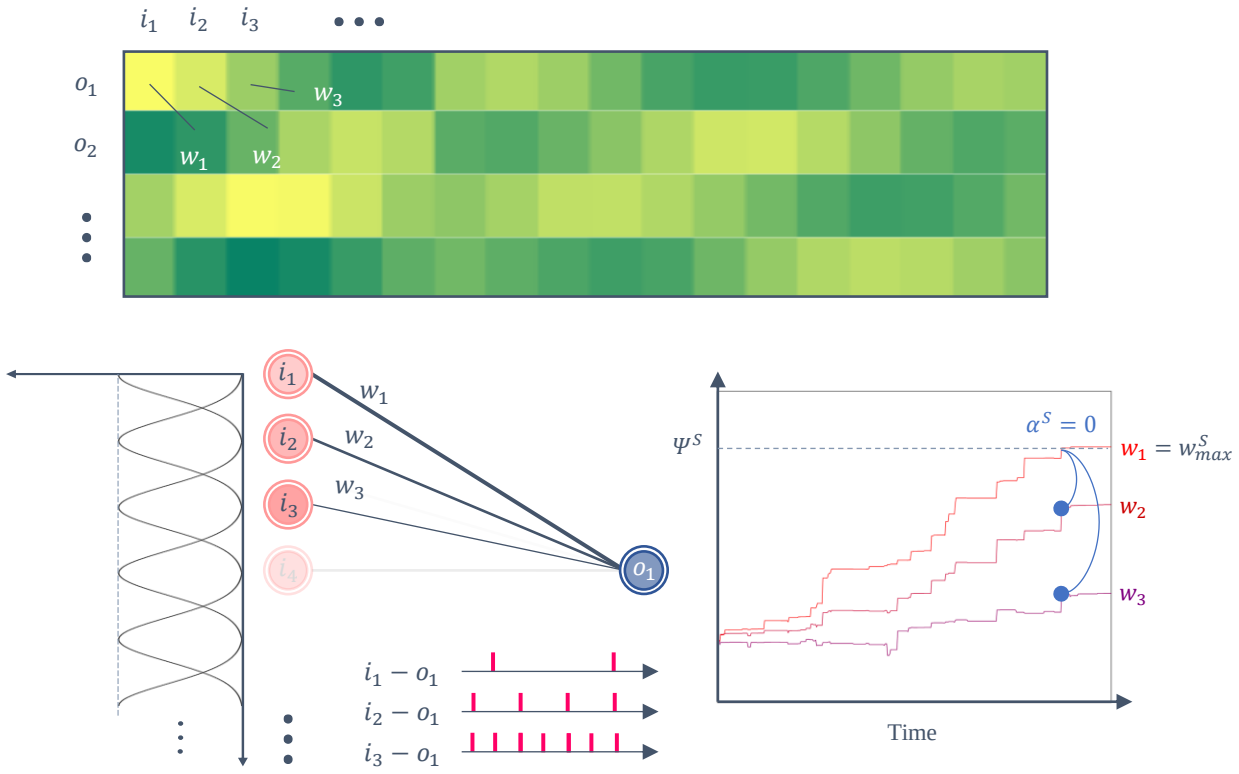



Figure 3.6: RCSE working principle in inhibiting the adjacent synaptic connections. The heatmap shows the synaptic weight matrix. Neurons have different firing strengths due to the difference in fuzzy membership values, which affects the increase or decrease rate and shapes the patterns in the synaptic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

Figure 3.6 shows how the maximum synaptic weight of the active synapses in set $\mathcal{S}$ 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 [108]. 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 [109, 110], 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 [111], 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:

$$\overline{\boldsymbol{W}}_k(t) = \frac{1}{I_k^{max}}[\boldsymbol{W}_k(t)] \tag{3.15}$$

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

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

$$\overline{\boldsymbol{W}}_G(t) = \frac{\sum_{k=1}^{N} \omega_k . \overline{\boldsymbol{W}}_k(t)}{\sum_{k=1}^{N} \omega_k} \tag{3.16}$$

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

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

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

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

current synaptic weights matrix, as follows,

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

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

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

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

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

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

$$\begin{aligned}\dot{\boldsymbol{W}}_k(t) =& (1 - \delta(t - T_{cs}))\left[\boldsymbol{\alpha} \odot \mathbf{STDP}(\tau) \odot \mathcal{R}(t) - \boldsymbol{\Theta} \odot \operatorname{sgn}(\boldsymbol{W}_k(t))\right] \\ &+ \delta(t - T_{cs}) I_k^{max}\left(\overline{\boldsymbol{W}}_G(t) - \overline{\boldsymbol{W}}_k(t)\right)\end{aligned} \tag{3.20}$$

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

Initialize the agents and Central Server with default parameters for model publication threshold, Euclidean distance, and model publish status
Initialize the Global Model on the Central Server
**if** $t$ is greater than 0 **then**
    Normalize synaptic weights of local models using (3.15)
    Aggregate models from all agents at the Central Server using (3.16)
    Calculate the $\mathcal{D}_G$ between the current and previous global models on the Central Server using (3.19)
    **if** $\mathcal{D}_G >$ the Central Server's threshold **then**
        Publish the global model
        set $T_{cs} = t$
    **end if**
    **for** each Agent in the network **do**
        **if** Central Server publishes a new global model **then**
            Update the local model of the Agent with the global model using (3.20)
        **else**
            Agents evaluate their local models against the latest global model ($\mathcal{D}_a$) using (3.18)
            **if** $\mathcal{D}_a >$ the Agent's threshold **then**
                Send the model to the Central Server
                set $T_k = t$
            **end if**
        **end if**
    **end for**
**end if**

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

## 3.3 Results and Discussion

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

### 3.3.1 Simulation without FL

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

Table 3.1: Parameter values for LIF neuron model [9]

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

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

Table 3.2: Simulation Parameters

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

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

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

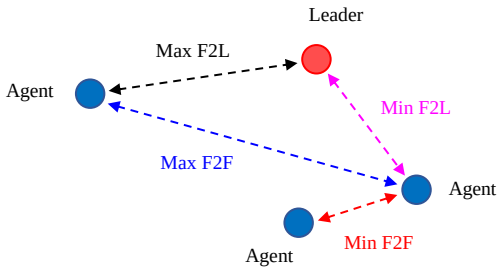



Figure 3.7: Measured distances used for evaluating swarm flight performance and collision detection.

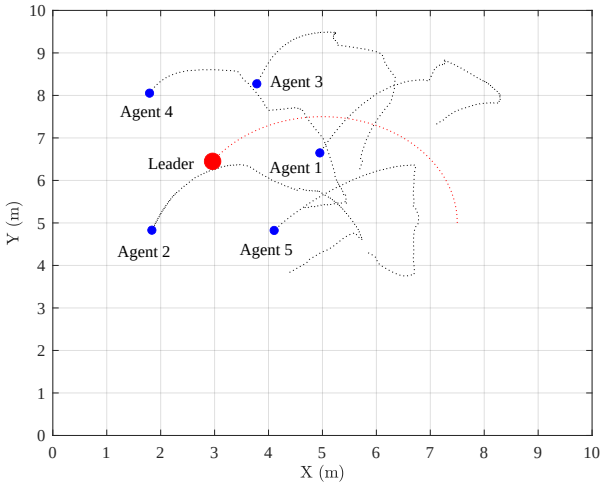



Figure 3.8: Agents' trajectory during the test phase

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

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

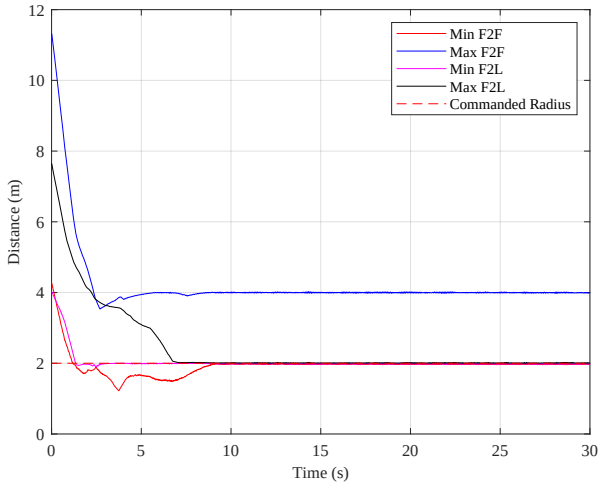



Figure 3.9: Variation of distances within the swarm during the test phase

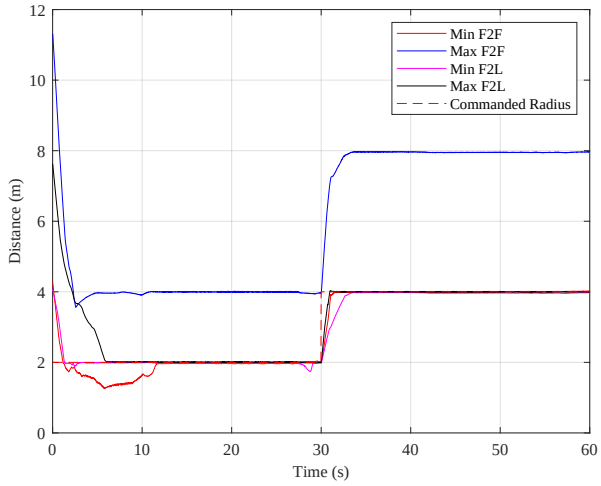



Figure 3.10: Adaptive response to commanded distance adjustments - reconfiguration during the test phase

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

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

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

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

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

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

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

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

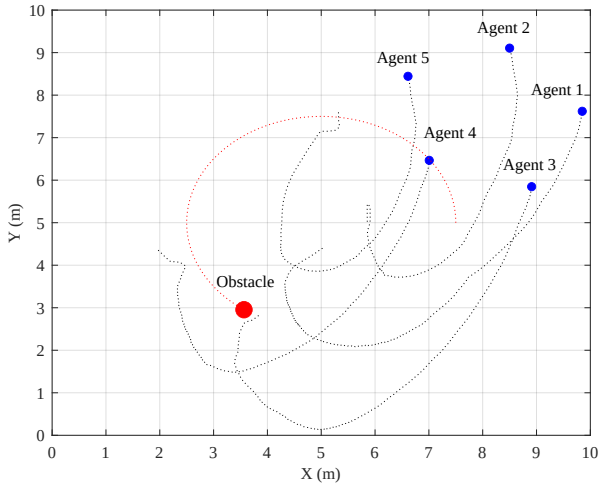



Figure 3.11: Trajectory adaptations of following agents in response to reward change for the leader during the test phase.

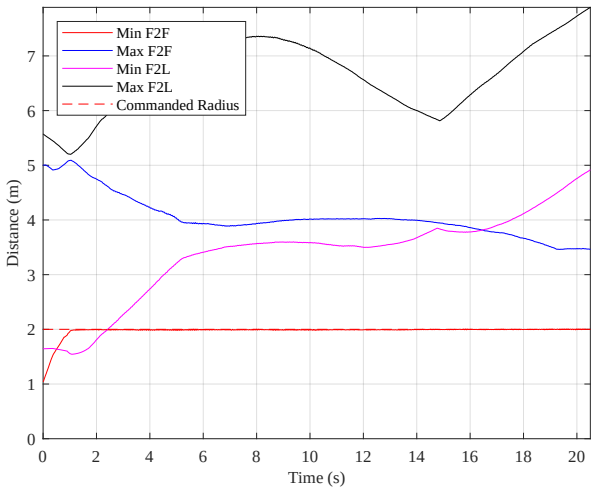



Figure 3.12: Variations in distances after reward changes and Leader becomes Obstacle - test phase.

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

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

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

Considering the numerical values presented in Table 3.2 along with the reward coefficients $\mathcal{C}_{Fi}^{Fj} = 0.02$ and $\mathcal{C}_{L}^{Fj} = 0.07$, and $r_{Fi}^{Fj} = 1\ m/s$ and $r_{Fi}^{L} = 0.1\ m/s$, the maximum rewards at each weight update interval ($\Delta T$) for $\mathcal{R}_{Fi}^{Fj}$ and $\mathcal{R}_{Fi}^{L}$ are calculated using (3.9) and (3.10) as $4 \times 10^{-4}$ and $7.7 \times 10^{-4}$, respectively. Consequently, $\mathcal{R}_{max}^{G} = \max(\mathcal{R}_{Fi}^{Fj}, \mathcal{R}_{Fi}^{L}) = 7.7 \times 10^{-4}$. The $\Psi^{S}$ for the follower section in the network is $\left[\frac{4\times10^{-4}}{7\times10^{-4}}\right] 15.5 = 8.0519$, and for the leader section, it is $\left[\frac{7\times10^{-4}}{7\times10^{-4}}\right] 15.5 = 15.5$. The $\eta$ and $\beta$ for the follower section within the network are $\frac{\mathcal{A}\mathcal{R}_{max}^{G} \ln(2^{\lambda^2}-1)}{\lambda\Psi^{\mathcal{S}} \log(2)} = 0.0011$ and $\beta = \frac{\ln(2^{\lambda^2}-1)}{\Psi^{\mathcal{S}}} = 2.152$, respectively. For the leader section, these values are $\frac{\mathcal{A}\mathcal{R}_{max}^{G} \ln(2^{\lambda^2}-1)}{\lambda\Psi^{\mathcal{S}} \log(2)} = 5.719 \times 10^{-4}$ and $\beta = \frac{\ln(2^{\lambda^2}-1)}{\Psi^{\mathcal{S}}} = 1.118$, respectively.

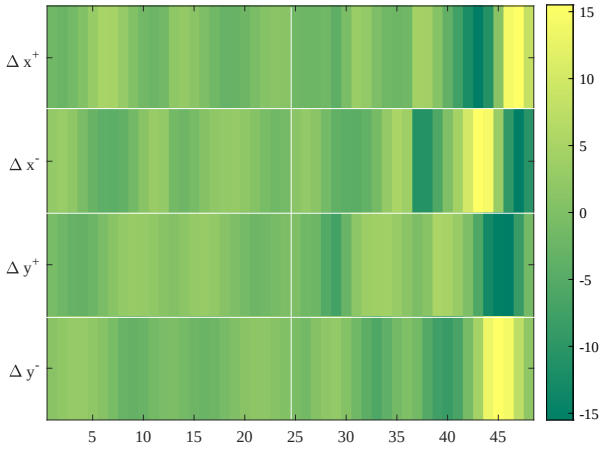



Figure 3.13: Synaptic Weights before Reward change in RCSE method.

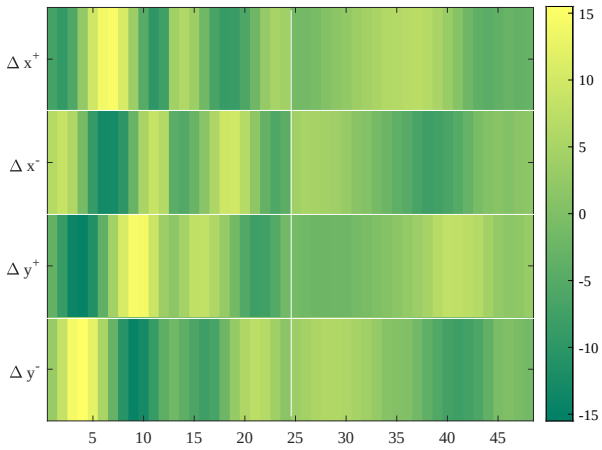



Figure 3.14: Synaptic Weights after Reward change in RCSE method.

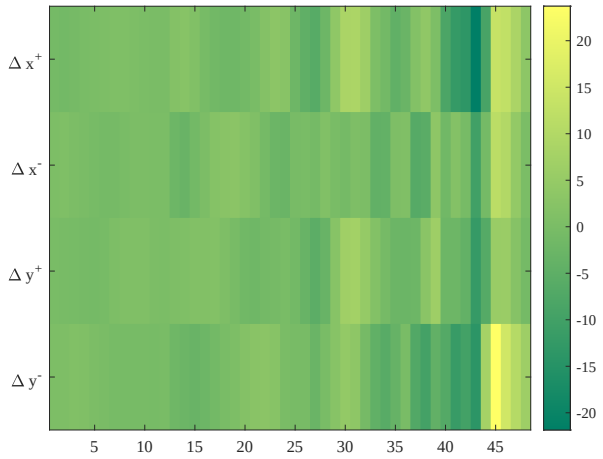



Figure 3.15: Synaptic Weights before Reward change in Multiplicative Synaptic Normalization method.

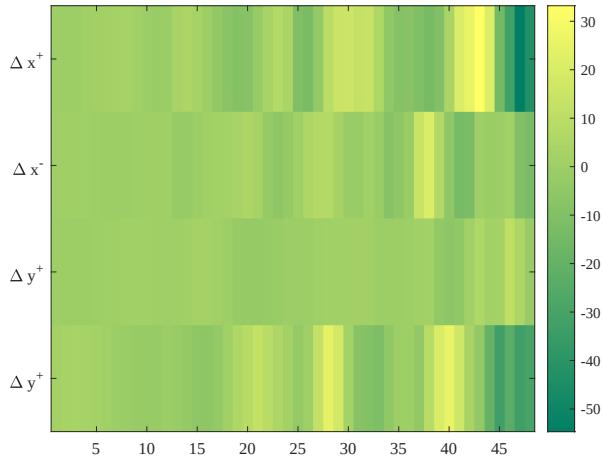



Figure 3.16: Synaptic Weights after Reward change in Multiplicative Synaptic Normalization method.

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

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

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

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

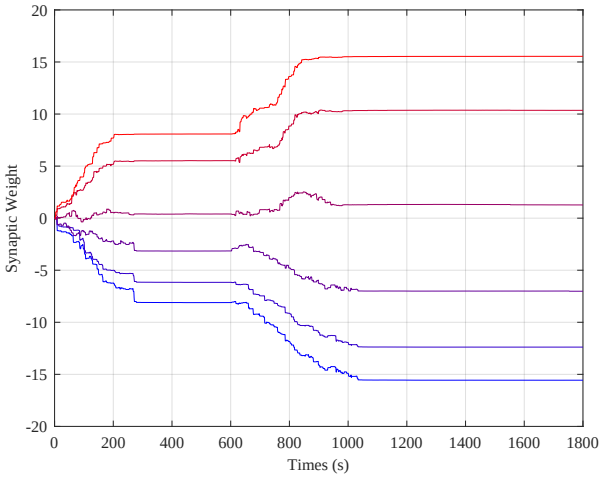



Figure 3.17: Synaptic weights increase after reward change in RCSE method.

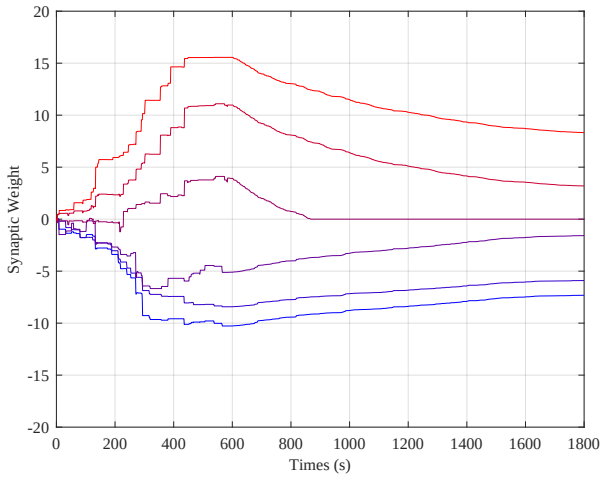



Figure 3.18: Synaptic weights decrease after reward change in RCSE method.

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

### 3.3.2 Simulation with FL and RCSE

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

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

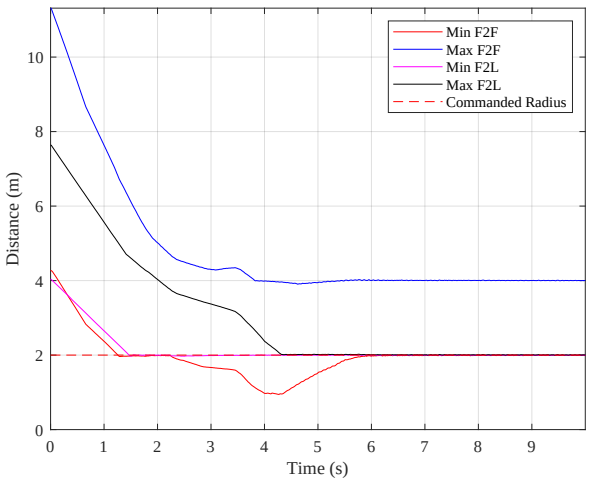



Figure 3.19: Distances during the test phase before reward change in the proposed event-triggered FL method.

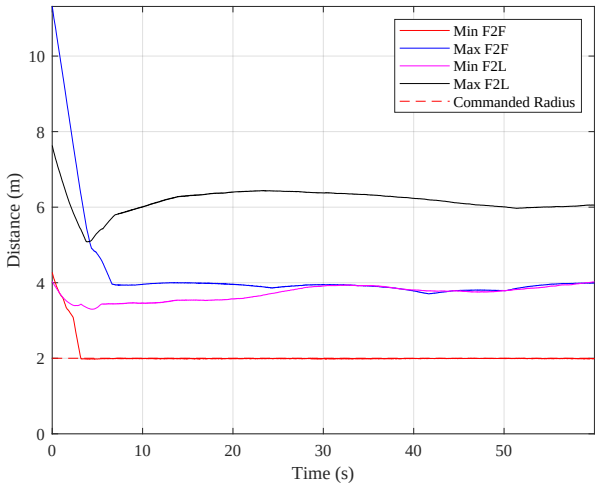



Figure 3.20: Distances during the test phase after reward change in the proposed event-triggered FL method.

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

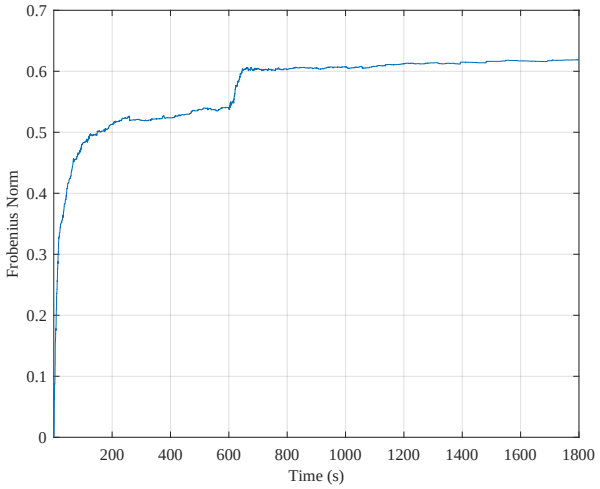



Figure 3.21: Frobenius norm of the Agent 1's weighs during the learning phase. The reward changes for the Leader after 600 s.

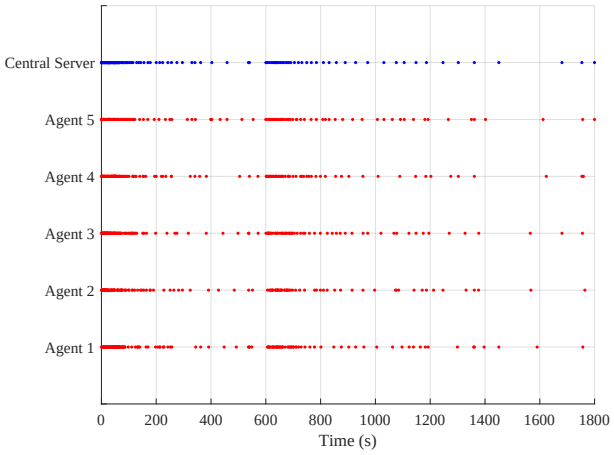



Figure 3.22: Communication times for agents and the Central Server (Leader). Red and blue dots show the times that agents and the Central Server have sent their model, respectively.

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

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

| | **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 |

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

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

## 3.4 Conclusion

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

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

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

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

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

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

# Chapter 4

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

Autonomous docking of multiple agents in close-proximity missions, such as on-orbit servicing, satellite formation flying, 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 Marker (ALM)s, 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 [112].

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 emulate a phase-locked-loop (PLL) mechanism (rather than tracking a sinusoidal phase, the neurons stabilize their membrane potentials so that recent DVS event patterns remain represented for tens of milliseconds.), providing short-term temporal memory of event sequences. These neurons maintain transient membrane states that store DVS activity over tens of milliseconds, supporting temporal continuity in event-driven perception when objects stop moving.

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 [113].

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.

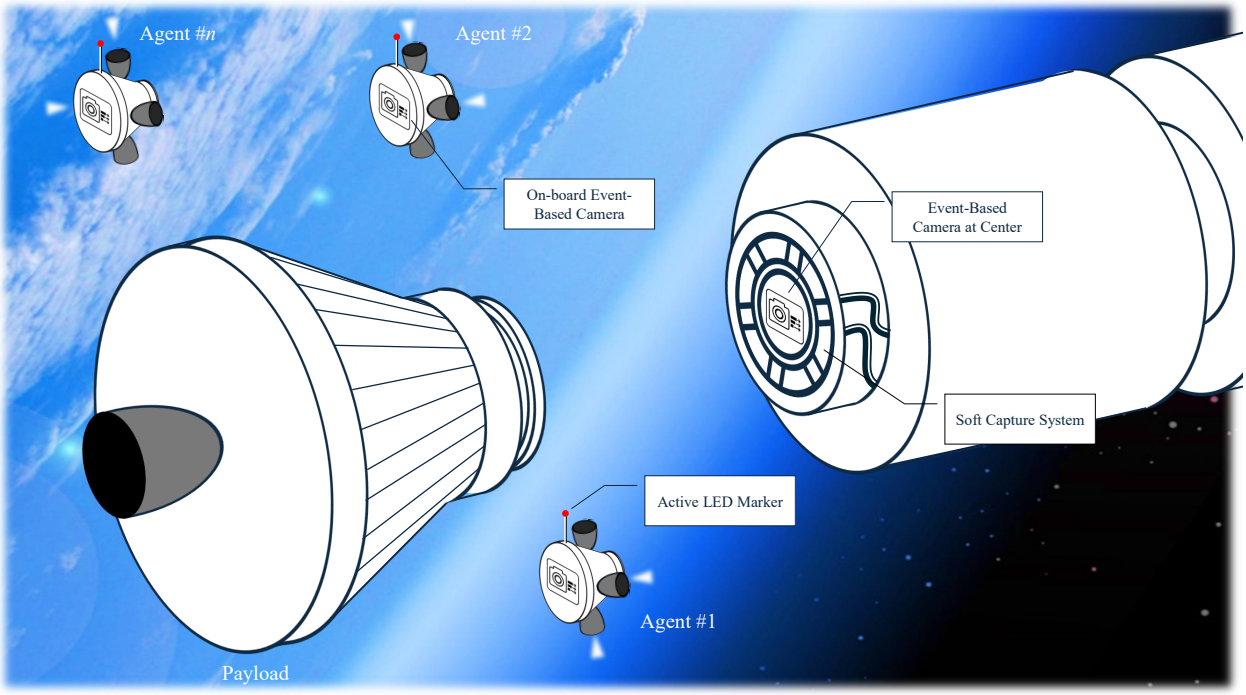



Figure 4.1: 3D schematic of the multi-agent docking mission environment. Multiple agents work collaboratively to push a central Payload towards a SCS for docking alignment.

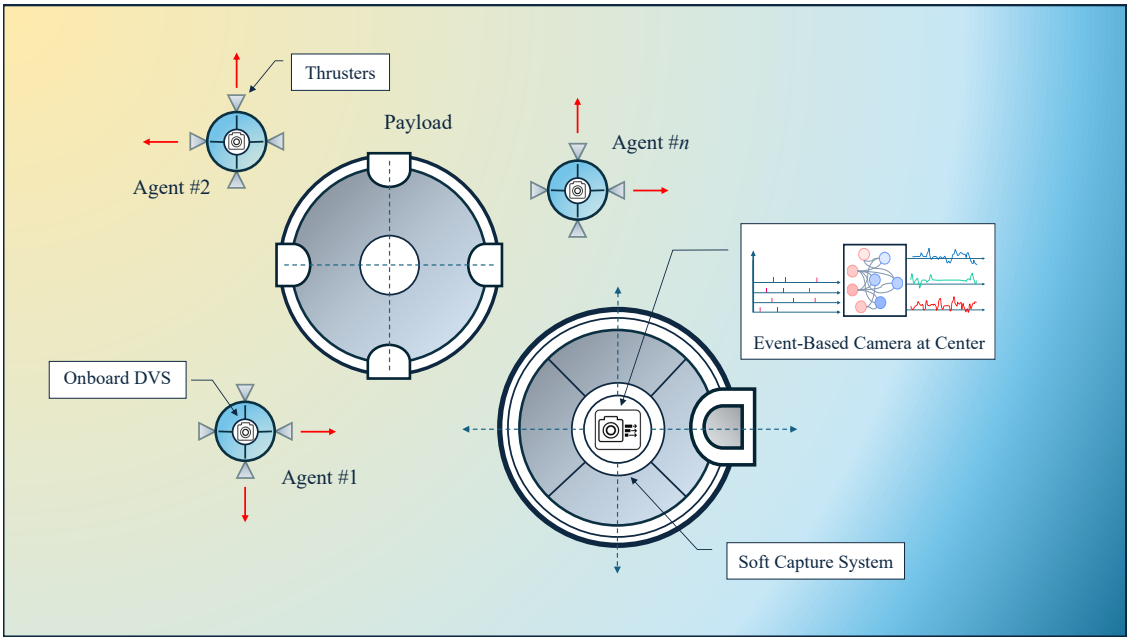



Figure 4.2: 2D representation of robots performing docking mission. Each agent pushes the Payload towards the SCS to align them together. The SCS and agents are equipped with a DVS that streams high-frequency data for the proximity mission.

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 the neighboring agent using DVS events and does not direct measurement of the positions. Because the Payload has a considerably larger surface area, 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 flying 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 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 $\zeta_{\text{bg}}$ 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:

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

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:

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

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

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

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

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

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

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

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

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 biolog-

ical photoreceptors, the brightness is first clamped from below to a minimum threshold $\zeta_{\min}$ (i.e., any value below $\zeta_{\min}$ is raised to $\zeta_{\min}$) and transformed into logarithmic intensity [112]:

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

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

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 [114]:

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

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

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:

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

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

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

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

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

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

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:

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

and its motion follows Newton's second law

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

Agent–Payload interactions are not modeled via a spring–damper; instead, first contact

triggers a rigid attachment for a fixed duration $\tau_{\text{attach}} = 10\,\text{s}$. Let $\mathbf{p}_P, \mathbf{V}_P, R_P, m_P$ be the Payload pose, velocity, radius, and mass. Attachment is triggered when

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

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

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

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:

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

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

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

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

After $\tau_{\text{attach}}$, 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 [115]. A minimum-impulse bit (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:

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

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

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

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

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

where $\mathbf{f}_{\text{coll}}(t)$ denotes the spring–damper collision force modeled previously. Because the pulse–width is fixed by the MIB, the actuation becomes hybrid and time–quantized: whenever $\mathbf{s}_k \neq \mathbf{0}$ the commanded magnitude $F_{\max}$ is maintained for the entire interval $[kT_{\text{mib}}, (k{+}1)T_{\text{mib}})$, precluding mid–window reversals.

To keep the agent within a controllable speed envelope under the discrete thrust actuation, a velocity governor is applied synchronously with the SNN output at every interval $T_{\text{mib}}$. At each update step $k$, the instantaneous velocity $\mathbf{V}_k = [V_{x,k}, V_{y,k}]^{\mathsf{T}}$ is compared against a predefined speed limit $V_{\max} > 0$. When the absolute velocity in an axis exceeds this limit, a corrective force proportional to the excess speed is generated; otherwise, the governor remains inactive. The per–axis control law is expressed as

$$u_{\text{gov},x}[k] = -\,m\,K_g\left(|V_{x,k}| - V_{\max}\right)_+ \text{sign}(v_{x,k}) \tag{4.26}$$

$$u_{\mathrm{gov},y}[k] = -\,m\,K_g\,\big(|V_{y,k}| - V_{\max}\big)_+\,\mathrm{sign}(v_{y,k}) \tag{4.27}$$

where $(x)_+ = \max(0, x)$ denotes the positive–part operator, returning $x$ if $x > 0$ and 0 otherwise, and $K_g > 0$ is the feedback gain. Because the governor is evaluated at the same discrete interval as the SNN output, its effect appears as an intermittent corrective pulse rather than a continuous viscous drag. It effectively prevents the agent from exceeding the controllable velocity envelope, ensuring that thrust reversals can occur within a few actuation periods despite the inertia and the 50 ms minimum–impulse constraint. This interaction between discrete actuation and velocity feedback introduces nonconvexity into optimization–based controllers such as Model-Predictive Contoller [116], since the control inputs $\mathbf{s}_k \in \{-1, 0, +1\}^2$ and their timing are inherently discrete and coupled to the system dynamics. In contrast, the proposed SNN adapts to this hybrid behavior by learning spike patterns that anticipate the motor's quantized response, thereby compensating for the nonlinear and discontinuous actuation without requiring explicit mixed–integer optimization.

### 4.1.4 Neuron model

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

$$\begin{cases} \dfrac{dv}{dt} \;=\; 0.04\,v^2 \;+\; 5\,v \;+\; 140 \;-\; u \;+\; I(t), \\[2ex] \dfrac{du}{dt} \;=\; a\,\big(b\,v \;-\; u\big), \end{cases} \tag{4.28}$$

with a reset rule:

$$\text{if } v \geq 30 \text{ mV, then } \begin{cases} v \leftarrow c, \\[1ex] u \leftarrow u + d. \end{cases} \tag{4.29}$$

where $a$ and $b$ are the neuron parameters, $v$ is the membrane potential (in mV), $u$ is a slow

recovery variable, $I(t)$ is the external input current, and $c$ and $d$ define how the system is reset after a spike.

## 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 for Payload Attention (PA) and Neighboring Agent Attention (N2A), the second hidden layer that governs the mission phases through regulatory and control repositories (Formation Phase Dominance (FPD), Docking Phase Dominance (DPD), Formation Flying Mission (FFM), and Collaborative Docking Mission (CDM)), 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.

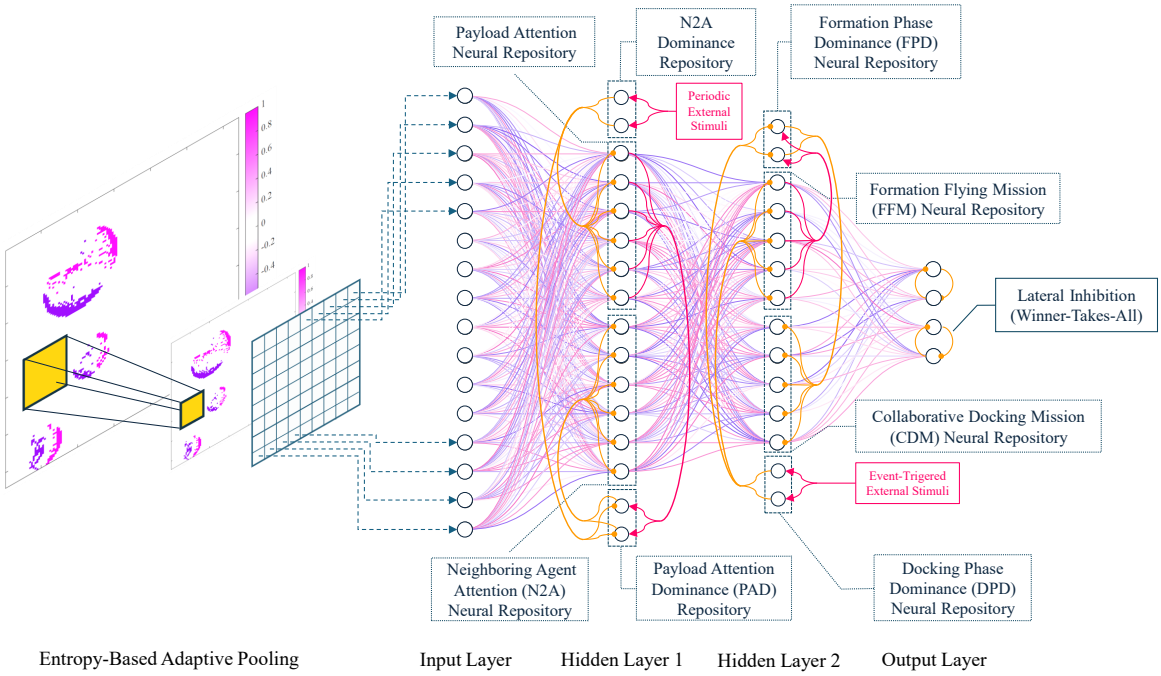



Figure 4.3: Deep SNN Architecture with entropy-based adaptive pooling, multi-repository hidden layers, and biologically-inspired inhibitory and excitatory synapses. Purple, red, and blue arrows denote excitatory and yellow arrows denote inhibitory pathways.

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 $32 \times 32$ 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 (PA and N2A), as well as Payload Attention Dominance (PAD) and Neighboring Agent Attention Dominance (N2AD) 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 FPD, DPD, FFM, and CDM neural repositories. These submodules are composed of bistable and Class 2 Izhikevich neurons that govern phase-gated transitions between formation 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 reward 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 formation flying 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 $\Omega$, each corresponding to a pixel in the downsampled output. For a region $\Omega$, we compute the frequency of event categories:

$$Pr(q) = \frac{1}{|\Omega|} \sum_{(i,j)\in\Omega} \mathbb{I}(E_{i,j} = q), \quad \text{for } q \in \{-1, 0, +1\}. \tag{4.30}$$

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

$$H(\Omega) = -\sum_{q\in\{-1,0,+1\}} Pr(q)\log_2 Pr(q), \tag{4.31}$$

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:

$$\rho(H) = \rho_{\max} - (\rho_{\max} - \rho_{\min}) \cdot \frac{H(\Omega)}{H_{\max}}. \tag{4.32}$$

Low-entropy (homogeneous) regions are pooled using a larger kernel $\rho_{\max}$, 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 weight calibration**

The input layer consists of "bistable" spiking neurons, each implemented using the Izhikevich neuron model operating in the bistable regime [117]. This configuration allows each neuron to remain either in a stable quiescent state or in a self-sustained spiking state, thus providing a persistent memory of recent input events. The bistable dynamics act as a phase-locked loop (PLL)–like mechanism: once a neuron is driven into its spiking orbit by an excitatory input, it continues to oscillate (phase-lock) with its intrinsic limit cycle until an inhibitory current resets it to the resting equilibrium. Consequently, the neuron preserves a trace of

the most recent event polarity, effectively encoding event history over extended timescales.

At steady state, setting $\dot{v} = \dot{u} = 0$ in the Izhikevich model yields the fixed-point equations:

$$u = bv, \tag{4.33}$$

$$0.04v^2 + (5 - b)v + (140 + I) = 0. \tag{4.34}$$

The equilibrium membrane potentials $v^*$ are given by the quadratic roots, and their stability depends on the discriminant

$$\mathbf{D} = (5 - b)^2 - 4(0.04)(140 + I). \tag{4.35}$$

When $\mathbf{D} > 0$, the neuron possesses two distinct real equilibria, corresponding respectively to a stable resting state and an unstable point near a spiking limit cycle. This coexistence of a fixed point and a self-sustained oscillatory attractor defines the bistable regime.

To determine the inhibitory current that precisely stabilizes the resting branch, we compute the critical current $I_{\text{crit}}$ at which the $v$-nullcline touches its extremum:

$$\frac{\partial}{\partial v}\big(0.04v^2 + 5v + 140 - u + I\big) = 0 \implies 0.08v + 5 = 0. \tag{4.36}$$

Hence,

$$v^* = -\frac{5}{0.08} = -62.5, \qquad u^* = b\,v^* = 1.5 \times (-62.5) = -93.75. \tag{4.37}$$

The $v$-nullcline is the curve in the $(v, u)$ phase plane where the instantaneous voltage derivative vanishes ($\dot{v} = 0$); the $u$-nullcline is where $\dot{u} = 0$. Equilibria are intersection points of these two curves.

Substituting $(v^*, u^*)$ into the nullcline equation gives

$$0.04(v^*)^2 + 5v^* + 140 - u^* + I_{\text{crit}} = 0 \tag{4.38}$$

$$I_{\text{crit}} = u^* - 0.04(v^*)^2 - 5v^* - 140 = -93.75 - 156.25 + 312.5 - 140 = -77.5. \tag{4.39}$$

Therefore, the inhibitory pulse amplitude required to place the neuron exactly at the saddle–node boundary is $-77.5$ to silence the neuron and return it to the resting equilibrium. Each input neuron receives currents proportional to the preprocessed DVS event matrix $E'$, where ON events $(+1)$ and OFF events $(-1)$ are mapped to distinct neuron indices. The current injection rule is defined as

$$I_i(t) = \begin{cases} 100, & \text{if } E'_i = +1, \\ -77.5, & \text{if } E'_i = -1, \\ -65, & \text{otherwise,} \end{cases} \tag{4.40}$$

where $I_i(t) = 100$ is a transient excitatory drive, $I_{\text{inh}} = -77.5$ is the inhibitory weight derived above, and $I_i(t) = -65$ represents the baseline current maintaining the neuron at rest. In this implementation, $I_i(t) = 100$ corresponds to the maximum depolarizing current that pushes the neuron toward the spiking attractor, while $I_i(t) = -65$ ensures quiescence in the absence of events.

Upon receiving an ON event, a neuron is momentarily excited with $I_i(t) = 100$ for a single simulation step. Owing to its bistability, it transitions into the spiking attractor and continues to fire even after the input is removed (like a phase-locked oscillator maintaining its phase after synchronization). The state of persistent spiking corresponds to the neuron being "locked" onto the active phase of the event stream. When the corresponding OFF event occurs, the neuron receives an inhibitory pulse of amplitude $I_i(t) = -77.5$, which pushes the state variable back across the saddle–node threshold, thereby terminating spiking and restoring the quiescent equilibrium.

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

asynchronous data and tracking static or intermittently visible targets.

### Hidden Layers

#### First Hidden Layer: 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 fuzzy 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 fuzzy spatial representation guided by reward function (section 4.2.2).

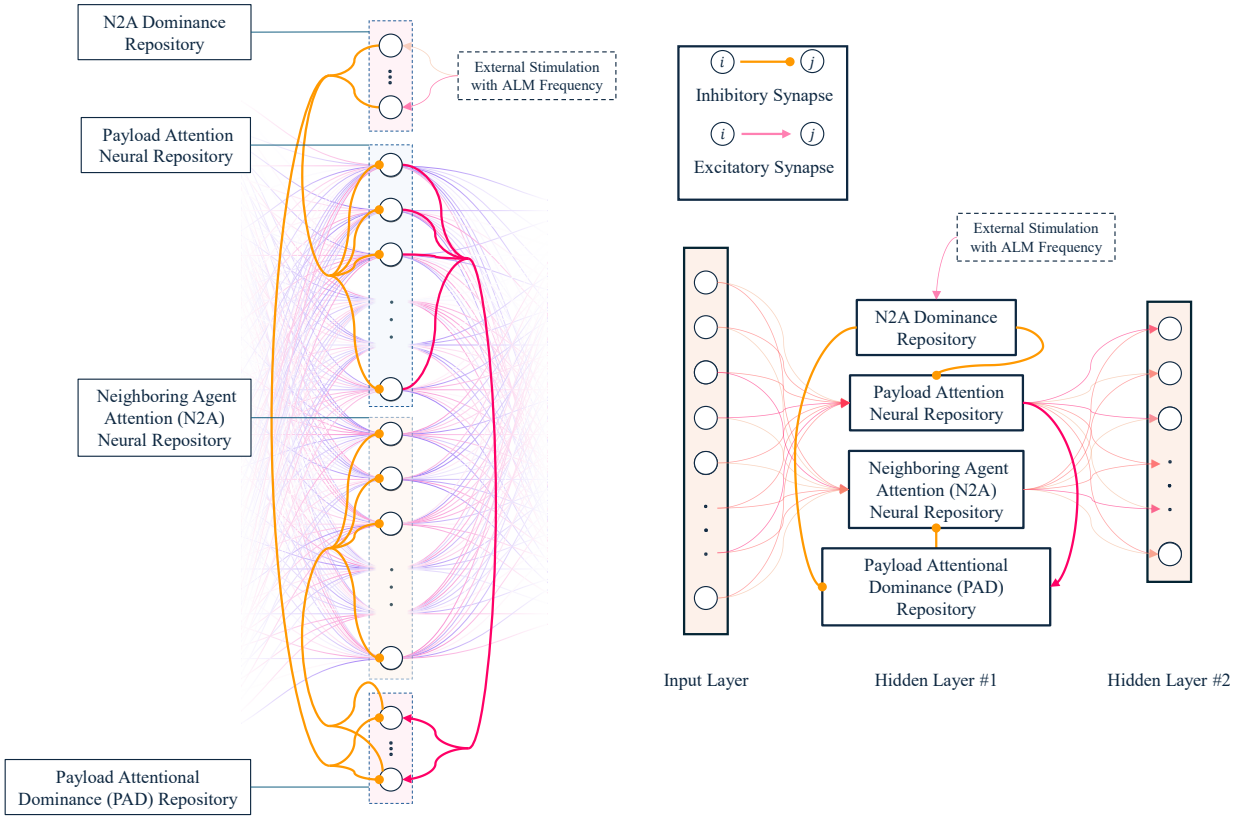



Figure 4.4: Structured First Hidden Layer Architecture with Payload, and Agent Perception Repositories for Visual Perception.

All neurons in this layer employ the Izhikevich model with Class 2 excitability for smooth frequency modulation, while the N2A Dominance Repository neurons specifically use a bistable configuration to maintain persistent spiking during activation intervals. This bistability enables autonomous inhibition without continuous external input: once N2A Dominance Repository neurons are activated (transitioning from stable to unstable state), 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. To avoid "synchronous silences" across the ALM neuron population — i.e., moments when all ALM neurons happen to be silent (or conversely all fire simultaneously) at a single sample time — the stimulation delivered during each activation interval is jittered by adding uniformly distributed random noise to the input current. Specifically, for each time step within the activation window the input current is perturbed by $\delta\mathbf{I}$. This produces sparse, staggered firing so that at least some ALM neurons are active at each sample time, preventing perfectly synchronous firing or silence and thereby maintaining continuous inhibitory/gating activity.

Functionally, the first hidden layer separates the Payload position channel and the proximal agents' position channel, representing them in a fuzzy spatial format (firing rates indicate the degree of activity associated with that spatial region) using the spiking activity of its neurons. This fuzzy coding is achieved by mapping the processed DVS input onto the neurons while allowing the network to maintain partial activation across spatial regions, enabling smooth transitions between detected object configurations. Additionally, the first hidden layer performs dimensionality reduction by projecting the high-resolution DVS input onto two separate repositories (one for the Payload and one for proximal agents) each represented on an $16 \times 16$ 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 inhibitation 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 phase-synchronized ALM signals, ensuring clear separation between agent and Payload representations, maintaining fuzzy spatial coding, reducing input dimensionality, and creating structured, distinguishable states for downstream processing in the SNN.

**Second Hidden Layer: 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 formation-flying and docking behaviors while learning phase-dependent control policies through spiking activity driven by external sensory feedback.

Within this layer, the Formation-Flying Mission (FFM), Collaborative Docking Mission (CDM), Docking Phase Dominance (DPD), and Formation Phase Dominance (FPD) repositories are interconnected to enforce phase-gated learning. The FFM and CDM 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 DPD and FPD repositories regulate these transitions through event-based mutual inhibition and excitatory control.

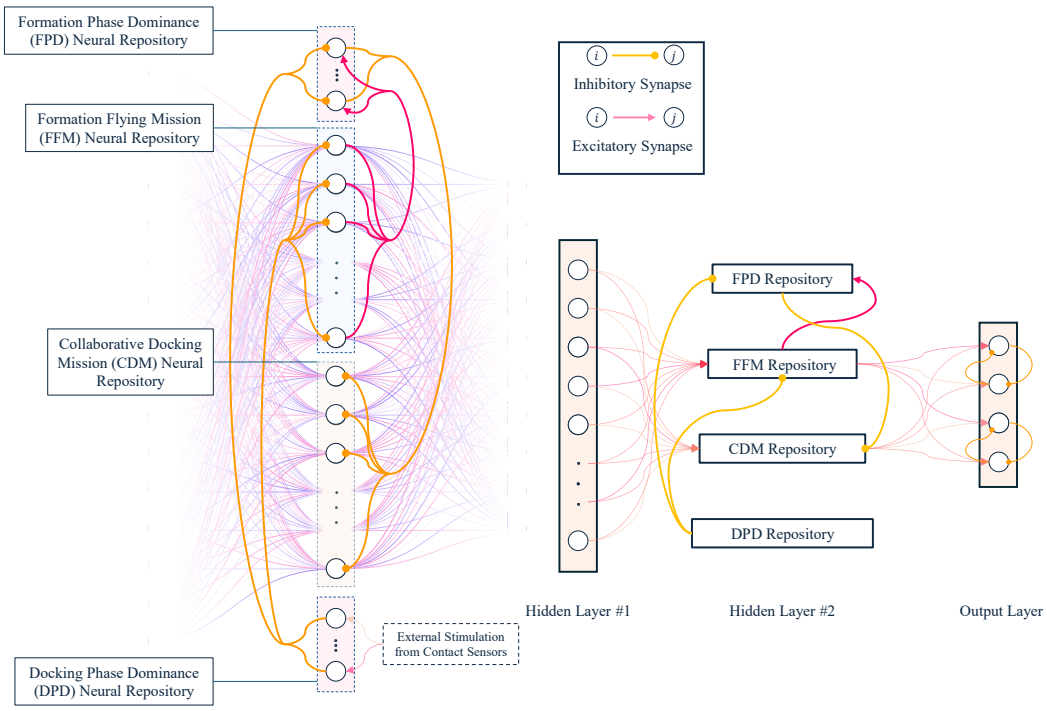



Figure 4.5: Structured Second Hidden Layer architecture showing the Formation-Flying (FFM), Collaborative Docking (CDM), Mission Phase Control (DPD), and Regulatory Formation Phase (FPD) repositories interconnected through inhibitory (yellow) and excitatory (purple) synapses. External stimulation from 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 FFM and CDM neurons are configured to exhibit regular, sustained spiking, enabling them to produce stable outputs during their respective active phases. In contrast, the FPD neurons exhibit bistable firing patterns, enabling them to maintain persistent activity while providing controlled inhibition during phase transitions. The DPD neurons use 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 formation flying, the agent has not contacted the Payload, the FFM neurons maintain persistent spiking activity. In this phase, the active learning pathway extends from the input layer to the visual processing layer, through the FFM, to the output layer, where synaptic connections within the FFM repository are updated according to the R-STDP rule. Upon Payload contact, the DPD neurons are activated, signaling the transition to the docking phase. The DPD neurons inhibit both FFM and FPD neurons. This inhibition silences FFM and FPD activity, releasing the CDM neurons from inhibition and allowing them to fire, thereby enabling the network to transition to the docking phase. During docking, the active learning pathway becomes the input layer to the visual processing layer, to the CDM, and continues to the output layer, allowing the network to learn the docking policy explicitly through the active CDM connections.

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

**Output layer, 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, $T_{\text{mib}} = 50$ 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 motor 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.

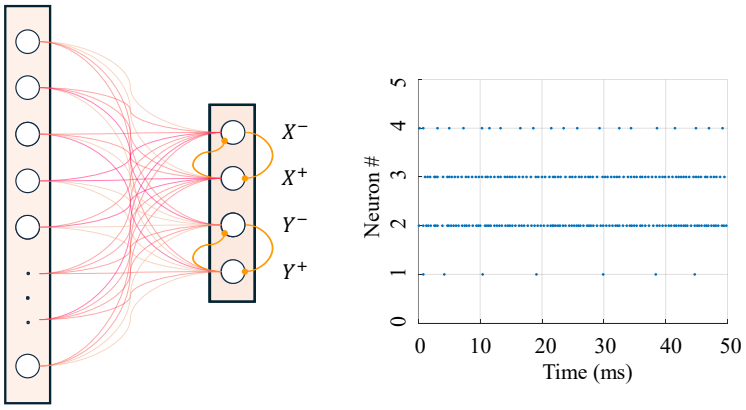



Figure 4.6: Output Layer Architecture with Lateral Inhibition between Opposing Direction Neurons to Enforce Winner-Take-All Dynamics within Each Control Axis.

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 $\tau_{\mathcal{C}} = 100$ ms. This value

corresponds to twice the MIB duration, ensuring that the effect of an action persists long enough for its motor 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 motor layers constrained by actuation dynamic.

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:

$$\text{duty}_i = \frac{1}{T_{\text{mib}}} \sum_{t \in [kT_{\text{mib}}, (k+1)T_{\text{mib}})} f_i(t), \tag{4.41}$$

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:

$$F_X^{\text{thrust}} = F_{\max} \, \text{sign}\big(\text{duty}_{X^+} - \text{duty}_{X^-}\big), \tag{4.42}$$

$$F_Y^{\text{thrust}} = F_{\max} \, \text{sign}\big(\text{duty}_{Y^+} - \text{duty}_{Y^-}\big), \tag{4.43}$$

where $F_{\max}$ is the maximum available thrust. 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.

### 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:

$$A(i)^{\text{input}} = \frac{1}{T} \sum_{t-T}^{t} f_i, \tag{4.44}$$

where $f_i(\tau) \in \{0, 1\}$ indicates whether neuron $i$ generated a spike at time $\tau$. Each entry $A(i)^{\text{input}}$ 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,

$$f_i(\tau) = \begin{cases} 1, & \text{if } v_i(\tau) \geq v_{\text{th}}, \\ 0, & \text{otherwise}, \end{cases} \tag{4.45}$$

where $v_i(\tau)$ is the membrane potential of neuron $i$ at time $\tau$ and $v_{\text{th}}$ 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 $A(i)^{\text{input}}$ represents the fraction of time steps in which neuron $i$ fired within the update window $T$.

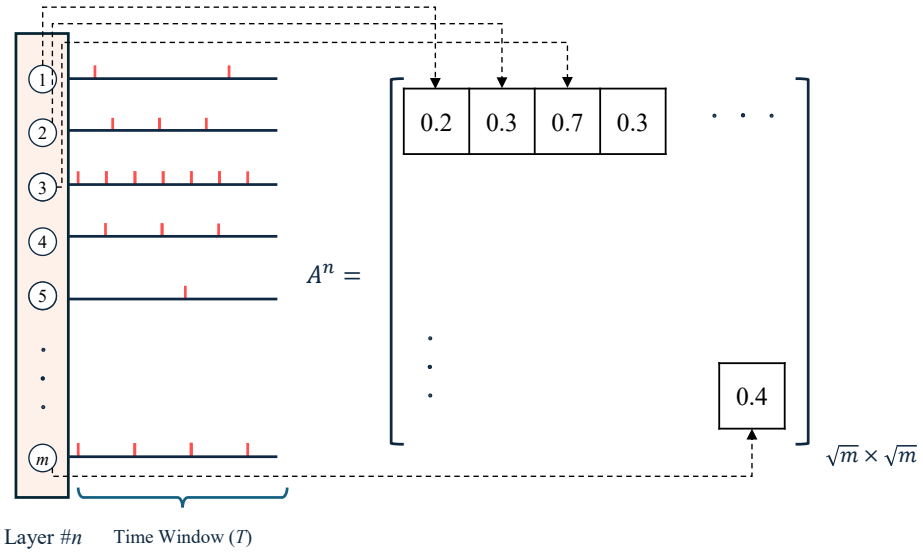



Figure 4.7: Computation of the input activity matrix ($A$) from the time-averaged spiking activity of the input layer neurons.

Similarly, for the first hidden layer ($A_1^{\text{hidden}}$), the Payload detector and agent detector repositories compute:

$$A_1^{\text{hidden}} = \frac{1}{T} \sum_{t-T}^{t} f_i^{\text{hidden}}. \tag{4.46}$$

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).

These form the matrices $A_{\text{Payload}}$ and $A_{\text{agent}}$, representing the spatial activity maps (location of the objects) for the Payload and agents, respectively. To compute the attention map, the input activity matrix $A^{\text{input}}$ is compared with the positions represented in the first hidden layer using a Gaussian-weighted mapping. For the first hidden layer activity matrix at $(x_q, y_q)$ and an input position $(x_k, y_k)$, the squared Euclidean distance is:

$$d_{kq}^2 = (x_k - x_q)^2 + (y_k - y_q)^2. \tag{4.47}$$

and the attention weight assigned is:

$$\gamma_{kq} = \exp\left(-\frac{d_{kq}^2}{2\sigma_A^2}\right), \tag{4.48}$$

where $\sigma_A = [0.05, 0.1]$ is a scale parameter controlling the sensitivity. The modulation signal $r_q^{(1)}$, which serves as the reward for the first hidden layer neuron $q$, is computed directly by comparing the attention-weighted input activity with the current hidden layer activity:

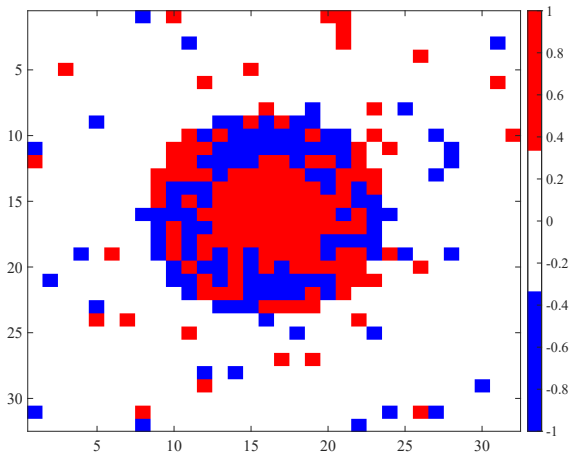
$$\mathcal{A}_q^{\text{Attention}} = \frac{\sum_k \gamma_{kq} \cdot A_k^{\text{input}}}{\sum_{k=1}^{K} \gamma_{kq} + \epsilon}, \tag{4.49}$$

$$r_q^{(1)} = A_{1,q}^{\text{hidden}} \times \left(\mathcal{A}_q^{\text{Attention}} - A_{1,q}^{\text{hidden}}\right), \tag{4.50}$$

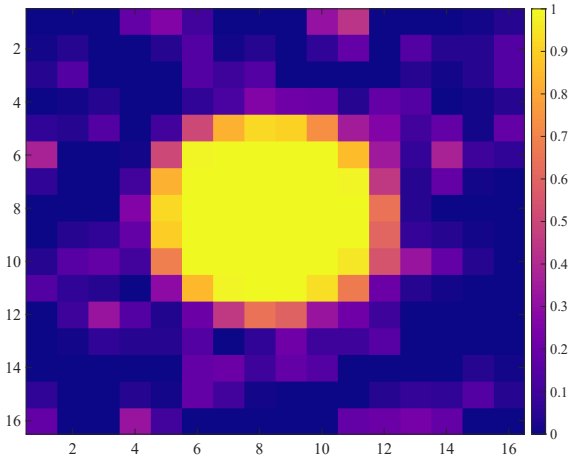
where $k \in \{1, \ldots, K\}$ is the index of the input sites, and $\epsilon$ ensures numerical stability. This formulation emphasizes high-activity regions aligned with the ALM areas and directly provides the per-neuron learning signal. When the hidden layer activity $A_1^{\text{hidden}}$ matches the attention-driven reward, the $r_q \to 0$, stopping the weight updates for that neuron during the iteration.

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, the reward signal for the inhibited repository is effectively multiplied by the zero activity of that repository (plasticity is 0), and no weight updates occur for those synapses. This selective gating ensures that only the active repository receives a weight update.

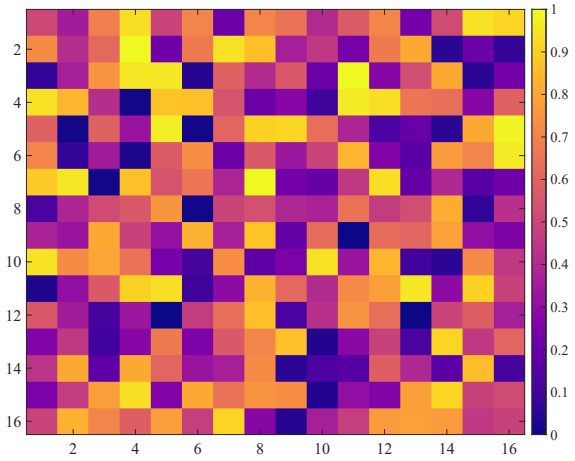
Figure 4.8 shows how the reward function pipeline works for the agent detection process in the first hidden layer. The reward signal provides the value of $r_q$ for each neuron, corresponding to each pixel in the representation.



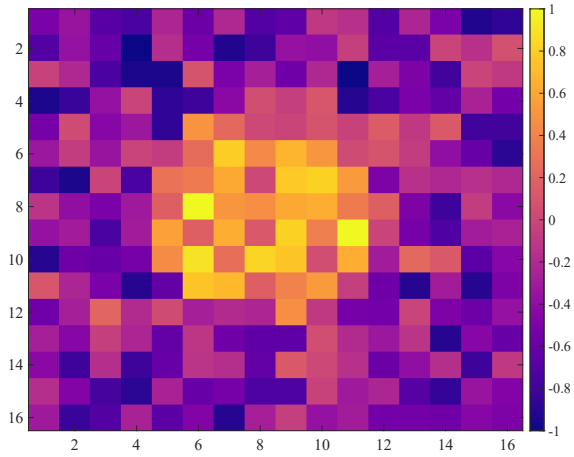
(a) Raw DVS Events of the payload



(b) Attention Map ($\mathcal{A}^{\text{Attention}}$)



(c) Hidden Layer Activity ($A_1^{\text{hidden}}$)



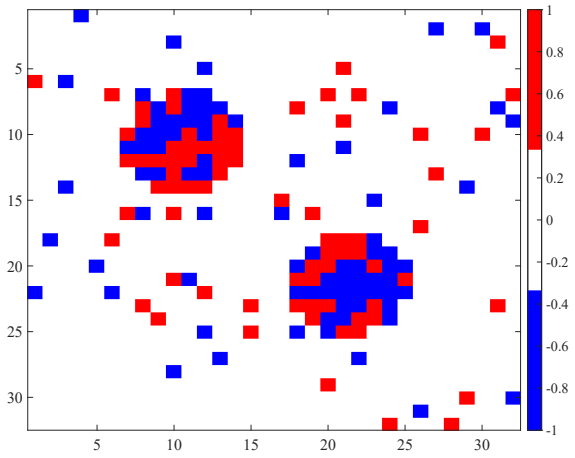
(d) Post-neuron reward ($r$)

Figure 4.8: 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 ($r_q$).

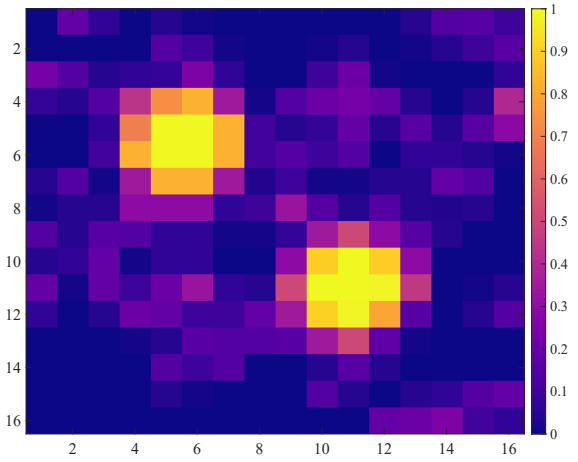
Each grid cell in Figure 4.8(d) shows the reward value assigned to the corresponding neuron in the payload detector repository of the first hidden layer. The spatial arrangement of these neurons follows the same layout as illustrated in Figure 4.7, where each cell represents a specific neuron's position within the repository grid.

Figure 4.9 shows an example of the reward function pipeline for the agent detector repos-

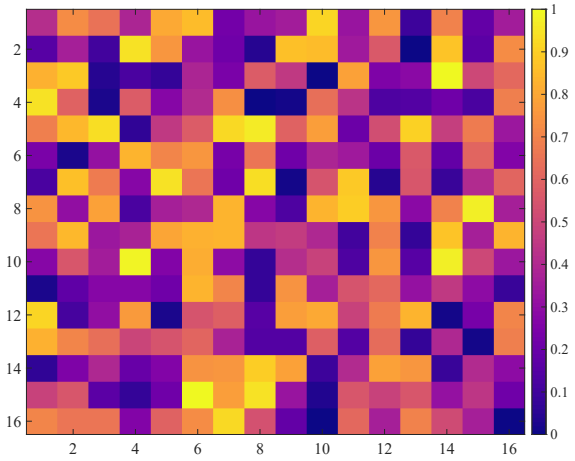
itory in the first hidden layer.



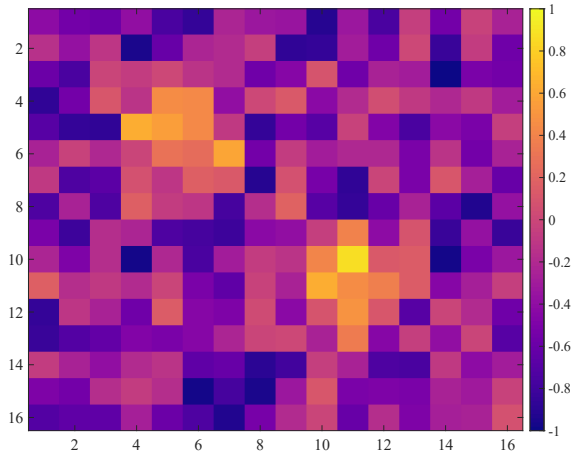
(a) Raw DVS Events of the agents



(b) Attention Map ($\mathcal{A}^{\text{Attention}}$)



(c) Hidden Layer Activity ($A_1^{\text{hidden}}$)



(d) Post-neuron reward ($r$)

Figure 4.9: 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.

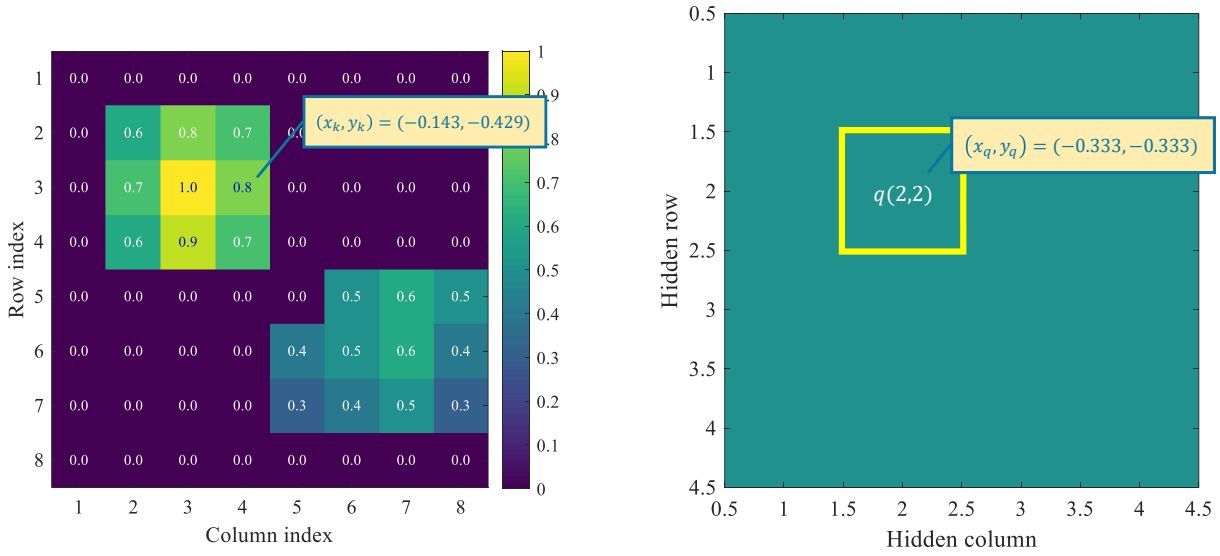
This design enables the first hidden layer to adaptively learn distinct environmental configurations (the spatial positions of the Payload and nearby agents), using event-based attention and ensures stable convergence in layered SNN learning.

**Numerical example (explicit Gaussian attention with $d_{kq}^2$ and $\gamma_{kq}$)** Use an $8 \times 8$ input grid and a $4 \times 4$ hidden grid (same as Figure 4.7). Coordinates follow the code:

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

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

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




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

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

Figure 4.10: Numerical example setup for Gaussian attention computation in the first hidden layer. (a) shows the input activity matrix representing time-averaged firing rates from the input layer. (b) illustrates the spatial arrangement of hidden-layer neurons, each serving as a query point for attention calculation.

*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)

$$g_x(k \mid x_q) = \exp\Big( - \frac{(x_k - x_q)^2}{2\sigma_A^2} \Big), \qquad 2\sigma_A^2 = 0.02.$$

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

$$G_q(r, c) = g_y(r \mid y_q)\, g_x(c \mid x_q), \qquad \gamma_{kq} = G_q(r, c) \;\; \text{with } k \leftrightarrow (r, c).$$

The attention at $q$ is the normalized weighted average

$$\mathcal{A}_q^{\text{Attention}} = \frac{\sum_{r,c} G_q(r,c)\, A^{\text{input}}(r,c)}{\sum_{r,c} G_q(r,c) + \epsilon}.$$

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

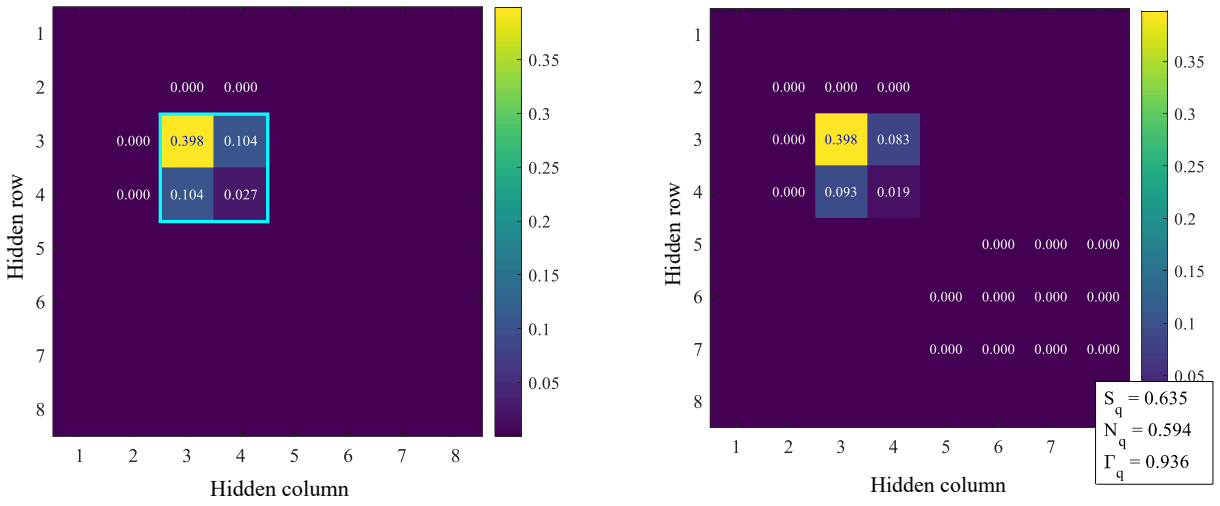
$$g_x(\cdot|-0.333) \approx [0.0000,\ 0.0007,\ 0.6308,\ 0.1645,\ 0.0000,\ 0.0000,\ 0.0000,\ 0.0000],$$

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

Form $G_q = g_y\, g_x^\top$. Non-negligible entries are the $2 \times 2$ block on rows 3:4, columns 3:4 (Figure 4.11(a)):

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

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




(a) Gaussian weight map ($G_q$).

(b) Weighted input activity ($G_q \odot A^{\text{input}}$).

Figure 4.11: 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 $G_q$ centered at the hidden neuron's location. (b) illustrates the element-wise product of the weight map with the input activity matrix, highlighting the contributions to the attention calculation.

The denominator (with $\epsilon$) is

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

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

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

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

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

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

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

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

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

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

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

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:

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

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

The four main weights are

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

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

With $S_q \approx 0.404 + 0.104 + 0.104 + 0.027 + 0.001 = 0.640$ 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

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

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

so $S_q \approx 0.814$. 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* $4 \times 4$ *attention map.* Repeating the above computation for all $q \in \{1, \ldots, 4\}^2$ 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

$$A_{\text{Payload}} = \begin{bmatrix} 0.050 & 0.100 & 0.000 & 0.000 \\ 0.100 & 0.200 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.100 & 0.200 \\ 0.000 & 0.000 & 0.050 & 0.100 \end{bmatrix}, \qquad A_{\text{Agent}} = \begin{bmatrix} 0.000 & 0.020 & 0.000 & 0.000 \\ 0.010 & 0.010 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.010 & 0.010 \\ 0.000 & 0.000 & 0.020 & 0.020 \end{bmatrix}.$$

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 $r_q = A_{1,q}^{\text{hidden}}\big(\mathcal{A}_q^{\text{Attention}} - A_{1,q}^{\text{hidden}}\big)$, the elementwise rewards for the payload detector become

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

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

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

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

the two matrices,

$$r_1^{\text{hidden}} = \begin{bmatrix} \text{rowvec}\big(r_{\text{Payload}}\big) \\ \text{rowvec}\big(r_{\text{Agent}}\big) \end{bmatrix} \in \mathbb{R}^{32\times 1}.$$

Positive entries indicate that the attention-driven input estimate exceeds the neuron's present activity, leading to potentiation 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 efficacy. 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

Let the incoming DVS events be 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 produces two separate detection maps, $P_{\text{pay}}$ and $P_{\text{agt}}$, each of size $N_m \times N_m$ (according to the activity matrix shown in Figure 4.7). These maps indicate how strongly the network believes that the payload or a proximal agent is present at each spatial location.

Because the first hidden layer incorporates a lateral inhibition mechanism, only one of these maps can be active at a time: when the payload detector becomes active, the agent detector is suppressed, and vice versa. For this reason, the current detection state of the network can be expressed simply as

$$P_{\text{det}} = P_{\text{pay}} + P_{\text{agt}}, \tag{4.51}$$

since exactly one of the terms is nonzero at any given moment.

To extract a binary foreground mask from this detection map, a sigmoid function is applied,

$$B(i,j) = \frac{1}{1 + \exp[-\beta\,(P_{\text{det}}(i,j) - \Upsilon)]}, \tag{4.52}$$

where $\Upsilon$ is a soft threshold and $\beta \gg 1$ controls the sharpness of the transition. For large $\beta$,

any nonzero detection value above the threshold is pushed very close to 1, producing a clean binary-like mask while remaining differentiable.

To apply the low–resolution mask $B$ to the full DVS frame, it must be expanded from size $N_m \times N_m$ to the event resolution $N_e \times N_e$. Because the ratio $\kappa = N_e/N_m$ is an integer, this upsampling can be performed by nearest–neighbor replication: each entry of $B$ is copied into a $\kappa \times \kappa$ block in the high–resolution image. Formally,

$$\mathcal{E}(B) = B \otimes \mathbf{1}_{\kappa\times\kappa}, \qquad (\mathcal{E}(B))_{m,n} = B_{\lceil m/\kappa \rceil,\, \lceil n/\kappa \rceil}. \tag{4.53}$$

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

$$E_{\text{det}} = E \odot \mathcal{E}(B), \tag{4.54}$$

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.

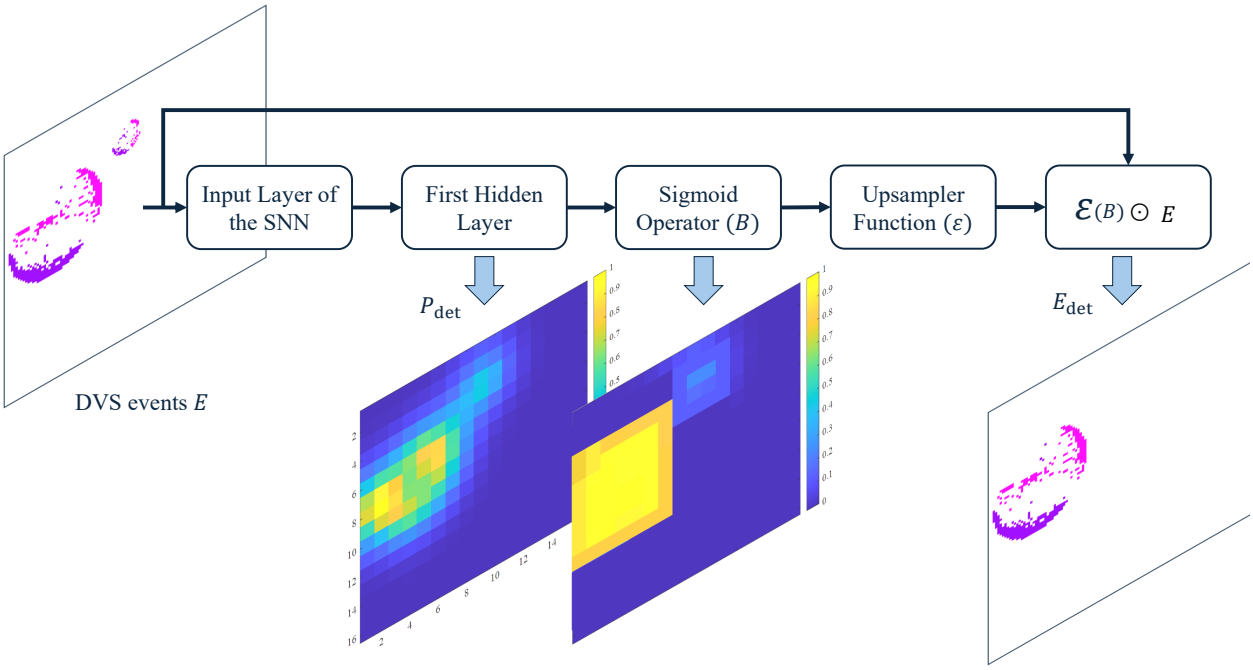



Figure 4.12: The DVS event image is masked by the upsampled binary detection mask derived from the first hidden layer's output, isolating events corresponding to the active object (Payload or agent).

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 $T_{\text{ALM}}$, with oscillation frequency and angular frequency

$$f_{\text{osc}} = \frac{1}{T_{\text{ALM}}}, \qquad \omega = 2\pi f_{\text{osc}}. \tag{4.55}$$

Let $G^{(e)} \in \mathbb{R}_{+}^{N_e \times N_e}$ be the Gaussian kernel sampled at the DVS resolution $N_e$, centered at the Payload, with standard deviation $\sigma_g$. The scalar value that detects the motion direction (whether the object is getting closer to the center of the Gaussian envelope or not) is computed as

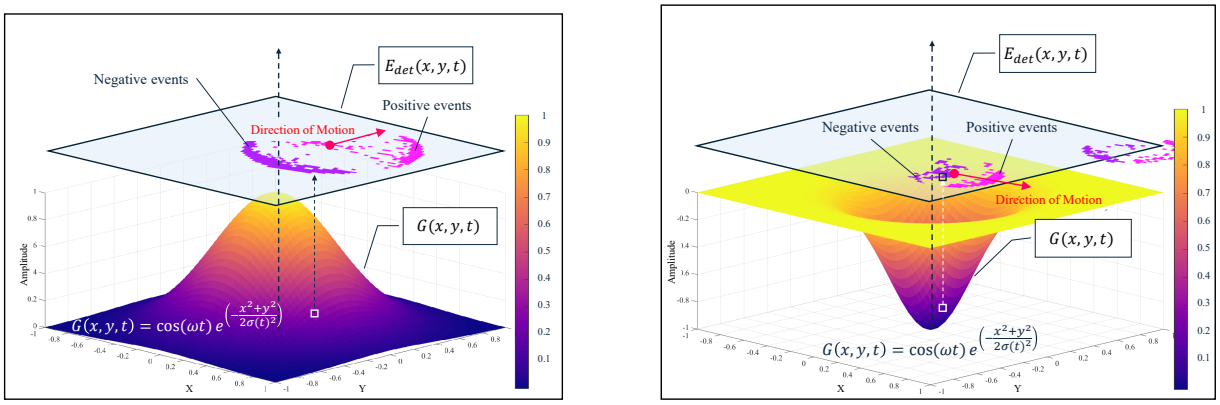
$$\mathcal{H} = \frac{1}{Z} \sum_{m=1}^{N_e} \sum_{n=1}^{N_e} E_{\text{det}}^{(e)}(m,n)\, G^{(e)}(m,n,t) \in \mathbb{R}. \tag{4.56}$$

The time-varying width and amplitude are

$$\sigma(t) = \sigma_0 + 0.5\sigma_0 \sin(\omega t), \qquad \alpha(t) = \cos(\omega t), \tag{4.57}$$

yielding the oscillatory Gaussian envelope

$$G(x,y,t) = \alpha(t) \exp\left(-\frac{x^2+y^2}{2\sigma(t)^2}\right), \quad (x,y) \in [-1,1]^2. \tag{4.58}$$




(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 4.13: The oscillatory Gaussian envelope used to compute the motion direction signal $\mathcal{H}$ for reward modulation in the second hidden layer.

Figure 4.13 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. When the amplitude is positive, the envelope concentrates on the Payload; when negative, it shifts focus to the agents. This dynamic modulation allows the network to adaptively learn from

the relevant object based on the current task context. Getting close to the center of the Gaussian envelope can produce negative or positive $\mathcal{H}$ values depending on the phase of the oscillation, thereby influencing the reward signal accordingly.

Each agent is equipped with a planar Inertial Measurement Unit (IMU) measuring the planar velocity $\mathbf{V} = [V_x, V_y]^\top$. Heading and normalized speed are

$$\varphi = \operatorname{atan2}(V_y, V_x), \qquad \bar{\mathcal{V}} = \frac{\|\mathbf{V}\|}{V_{\max}}. \tag{4.59}$$

The velocity is encoded into an $N_g \times N_g$ grid with cell centers $(x_i, y_j)$ (a point whose $x$- and $y$-coordinates both range from $-1$ to 1), normalized polar coordinates

$$R_{ij} = \frac{\sqrt{x_i^2 + y_j^2}}{\sqrt{2}}, \quad \Theta_{ij} = \operatorname{atan2}(y_j, x_i), \tag{4.60}$$

and wrapped angular difference

$$\Delta\theta_{ij} = \operatorname{atan2}\big(\sin(\Theta_{ij} - \varphi), \cos(\Theta_{ij} - \varphi)\big). \tag{4.61}$$

The separable membership is

$$\mu_r(i,j) = \exp\left(-\frac{(R_{ij} - \bar{\mathcal{V}})^2}{2\sigma_r^2}\right), \quad \mu_\theta(i,j) = \exp\left(-\frac{\Delta\theta_{ij}^2}{2\sigma_\theta^2}\right), \tag{4.62}$$

$$M_{ij} = \frac{\mu_r(i,j)\,\mu_\theta(i,j)}{\max_{a,b}\,\mu_r(a,b)\,\mu_\theta(a,b)}, \tag{4.63}$$

with opposite-direction map $\widetilde{M} = M(\varphi + \pi)$.

Let $A \in [0,1]$ be the normalized activity vector of neurons in the second hidden layer (spikes per $\Delta T$ divided by a maximum). The reward is decomposed into positive and negative parts

$$\mathcal{H}_+ = \frac{\mathcal{H} + |\mathcal{H}|}{2} \geq 0, \quad \mathcal{H}_- = \frac{|\mathcal{H}| - \mathcal{H}}{2} \geq 0, \tag{4.64}$$

yielding the per-neuron reward

$$L_i = \mathcal{H}_+ M_i + \mathcal{H}_- \widetilde{M}_i. \tag{4.65}$$

The modulation signal for each neuron in this layer is

$$r^{(2)} = ((v_x \ = 0)||(v_y \ = 0)) \times (\mathcal{H}_+(M - A) + \mathcal{H}_-(\widetilde{M} - A)), \tag{4.66}$$

which pushes activity toward $M$ for $\mathcal{H} > 0$ and toward $\widetilde{M}$ for $\mathcal{H} < 0$, and vanishes for $\mathcal{H} = 0$.

**Numerical example (second hidden layer with the provided data)** We set $N_e = 8$, $N_m = 4$, hence the upsampling factor is $\kappa = N_e/N_m = 2$. At the current instant the oscillatory envelope has amplitude $\alpha(t) = 1$ (as given). Unless noted, all numbers are rounded to 3 decimals.

*Inputs.*

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

Thus $P_{\text{det}} = P_{\text{pay}}$.

*Binarization and upsampling.* With threshold $\Upsilon = 0.3$ and steepness $\beta = 10$,

$$B(i,j) = \frac{1}{1 + \exp\big(-\beta(P_{\text{det}}(i,j) - \Upsilon)\big)}.$$

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

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

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

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

*Masked events.*

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

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

each axis and width $\sigma_g = 0.6$:

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

This yields

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

The scalar reward is

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

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

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

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

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

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

On a $3\times3$ grid with $(x_i, y_j) \in \{-1, 0, 1\}^2$, define $R_{ij} = \sqrt{x_i^2 + y_j^2}/\sqrt{2}$ and $\Theta_{ij} = \operatorname{atan2}(y_j, x_i)$ (set $\Theta_{2,2} = \varphi$ at the origin). With $\sigma_r = 0.3$, $\sigma_\theta = 0.6$,

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

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

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

*Per–neuron reward and signed credit.* Let the normalized second–layer activities be

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

Since $\mathcal{H} < 0$,

$$r^{(2)} = \mathcal{H}_+(M - A) + \mathcal{H}_-(\widetilde{M} - A) = 0.011\,(\widetilde{M} - A),$$

so

$$r^{(2)} = \begin{bmatrix} 0.003 & -0.001 & -0.003 \\ 0.006 & -0.005 & -0.002 \\ 0.000 & -0.003 & -0.006 \end{bmatrix}.$$

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* reward ($r = -0.011$), so learning pushes activity toward the *opposite* velocity map $\widetilde{M}$: cells aligned with $\varphi + \pi$ are encouraged (positive entries in $L$), while currently overactive cells are suppressed by the negative entries of $\dot{W}$.

### Output Layer Reward Computation Based on Directional Error

The output layer comprises four neurons actuating along the $\pm X$ and $\pm Y$ axes. The reward is constructed to align the output action with the commanded direction decoded from the preceding layer.

Let the second hidden layer provide a directional activity over $N$ evenly spaced angular bins with centers

$$\mu_k \;=\; \frac{360^\circ}{N}\,k, \qquad k \in \{0, \ldots, N-1\},$$

and nonnegative weights $w_k$ given by 10 $ms$ binned spike counts. The commanded direction is decoded by the circular mean

$$C \;=\; \sum_{k=0}^{N-1} w_k \cos(\mu_k), \qquad S \;=\; \sum_{k=0}^{N-1} w_k \sin(\mu_k), \qquad \theta_{\text{cmd}} \;=\; \text{atan2}(S,\, C) \in [0^\circ, 360^\circ).$$

Denote by $L, R, D, U$ the 50 $ms$ averaged spike counts of the four output neurons corresponding to $-X, +X, -Y, +Y$, respectively. The instantaneous action vector is formed as

$$\mathbf{a} \;=\; (L - R,\; D - U), \qquad \theta_{\text{act}} \;=\; \text{atan2}(D - U,\; L - R) \in [0^\circ, 360^\circ),$$

so that the cosine and sine of $\theta_{\text{act}}$ are obtained directly from the normalized components of $\mathbf{a}$. Writing the trigonometric discrepancies

$$\Delta_c \;=\; \cos(\theta_{\text{cmd}}) - \cos(\theta_{\text{act}}), \qquad \Delta_s \;=\; \sin(\theta_{\text{cmd}}) - \sin(\theta_{\text{act}}),$$

the four per-neuron rewards are

$$r_{X^-} = \Delta_c, \qquad r_{X^+} = -\Delta_c, \qquad r_{Y^-} = \Delta_s, \qquad r_{Y^+} = -\Delta_s,$$

which encourage thrust along the commanded direction and suppress the opposite directions.

**Weight Update using R-STDP**

To ensure consistent hierarchical learning, the synaptic plasticity of each layer is modulated by the convergence state of its preceding layer. Let $r^\ell$ denote the instantaneous reward of the layer $\ell$. The effective reward of the layer $\ell$ is adaptively damped according to the stability of the previous layer as

$$r^\ell = r^\ell \exp\big(-a_L \max(r^{\ell-1})\big), \tag{4.67}$$

where $a_L > 0$ is a tunable damping coefficient that controls how strongly the upstream plasticity suppresses downstream adaptation. This hierarchical modulation ensures that the second hidden layer begins significant adaptation only after the first hidden layer produces stable and reliable feature representations, and the output layer adapts only when the directional encoding in the second hidden layer becomes consistent. As a result, synaptic updates propagate through the network in a layer-wise manner, reducing interference between layers and promoting coherent temporal convergence of the entire spiking hierarchy.

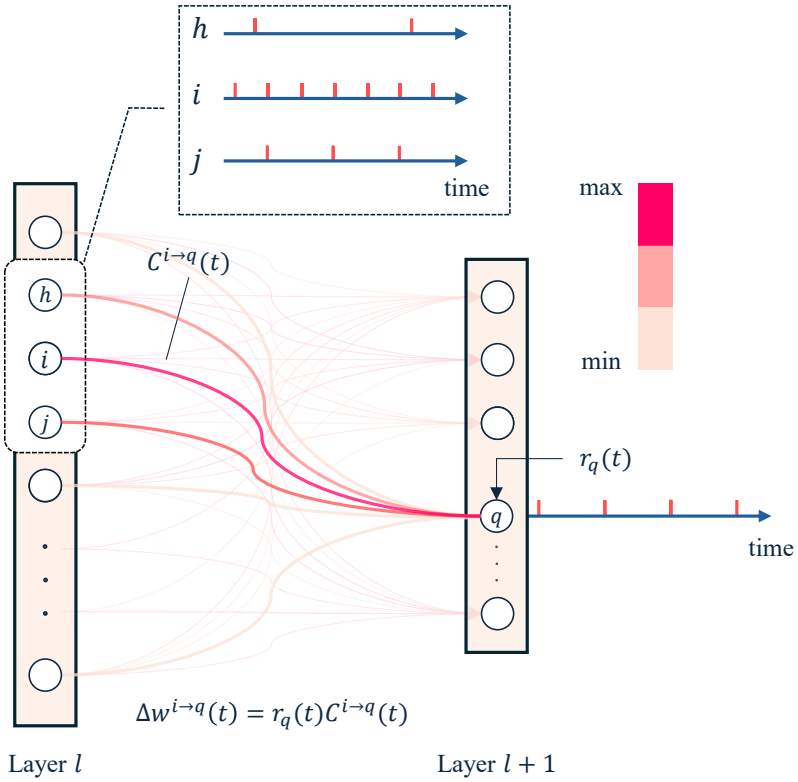



Figure 4.14: Schematic of the per-layer R-STDP weight update. The eligibility trace is calculated based on the firing rate of the neurons in layer $\ell$ and $\ell + 1$. The per-neuron reward ($r_q$) is computed based on the layer-specific reward functions. The weight update is obtained by multiplying the eligibility trace by the per-neuron reward.

The network is partitioned into layers using index bounds $\{\ell = 1, \ldots, L-1\}$. For each layer $\ell$, let $\mathcal{J}_\ell$ be the set of presynaptic indices and $\mathcal{I}_{\ell+1}$ the set of postsynaptic indices. 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

$$\dot{\mathbf{C}}^l(t) = -\frac{1}{\tau_c}\mathbf{C}^l(t) + \mathbf{STDP}(\tau)\delta(t - \mathbf{T}^l) \tag{4.68}$$

where, $\tau_c$ is the constant for the decay of $\mathbf{C}_{\text{post, pre}}(t)$, the $\tau$ is the spike time difference between pre and post neurons, and $\mathbf{T}^l$ 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 are computed on the postsynaptic side of each layer. Let $\boldsymbol{r}^{(\ell)}(t) = \left[r_i^{(\ell)}(t)\right]_{i\in\mathcal{I}_{\ell+1}} \in \mathbb{R}^{|\mathcal{I}_{\ell+1}|}$ collect the layer–specific reward vector, respectively. The instantaneous, gated R-STDP update for layer $\ell$ is then

$$\Delta\mathbf{W}^{(\ell)}(t) \;=\; \boldsymbol{r}^{(\ell)}(t) \circ \mathbf{C}^{(\ell)}(t) \;\in\; \mathbb{R}^{|\mathcal{I}_{\ell+1}|\times|\mathcal{J}_\ell|}, \tag{4.69}$$

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 [118].

| **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, $c_{\text{damp}}$ | 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) [119], 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 PAD and 20 N2AD attention 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 FFM, 45 CDM, 20 DPD, and 20 FPD. Finally, the output layer consists of 4 Bistabilitiy 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 | 512 Visual neurons<br>20 Payload Attention Dominance (PAD)<br>20 Neighboring Agent Attention Dominance (N2AD) |
| Hidden 2 | Class 2 | 130 | 45 Formation Flying Mission (FFM)<br>45 Collaborative Docking Mission (CDM)<br>20 Docking Phase Dominance (DPD)<br>20 Formation Phase Dominance (FPD) |
| Output | Bistability | 4 | Generates PWM thrust commands for $X$–$Y$ axes |

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

Table 4.3 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.

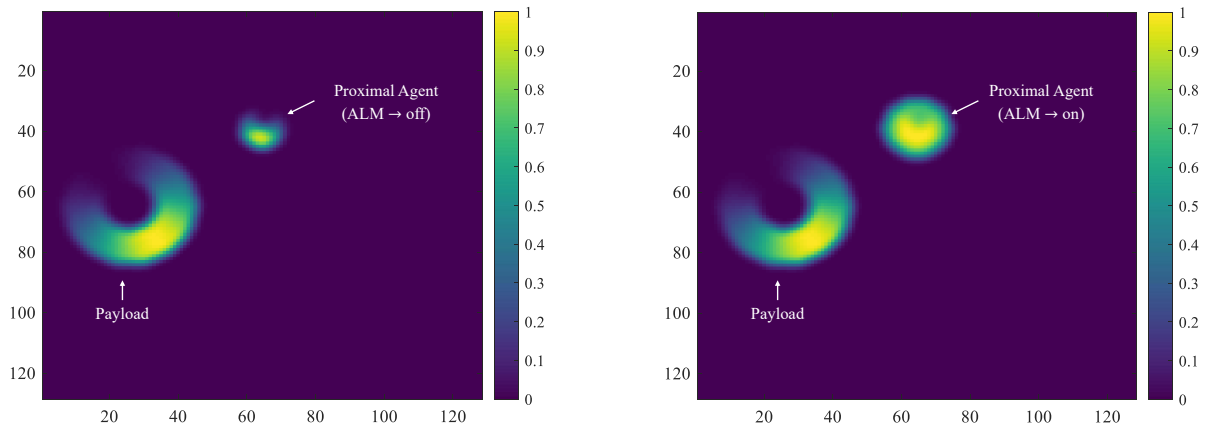
To ensure robust inhibitory control and prevent phase-mixing, each of the FPD, DPD, and ALM repositories is implemented with multiple neurons rather than a single unit. By deploying more than one neuron per repository, we ensure that at least one neuron will fire at each sample time, maintaining a continuous inhibitory or gating signal even under variable spiking conditions. Furthermore, we introduce small random perturbations to the initial

Table 4.3: Layer-wise Learning Parameters

| **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 |

membrane potentials and Izhikevich parameters across neurons within each repository. This heterogeneity breaks up synchrony, if all neurons in a repository were identical and initialized identically, they would tend to fire simultaneously, weakening phase separation. Instead, randomized initialization ensures staggered spiking, which improves the temporal coverage of inhibition and strengthens the decoupling of formation-flying and docking learning phases.

Figure 4.15 illustrates the effect of activating the ALM on the agents' visual perception. When the ALM is turned off (Figure 4.15a), the proximal agent produces relatively weak brightness variations compared to the payload, resulting in limited event generation in the DVS. As a consequence, the visual perception layer has difficulty distinguishing the smaller agent from the background. In contrast, when the ALM is turned on (Figure 4.15b), 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 attention mechanism, allowing the network 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.

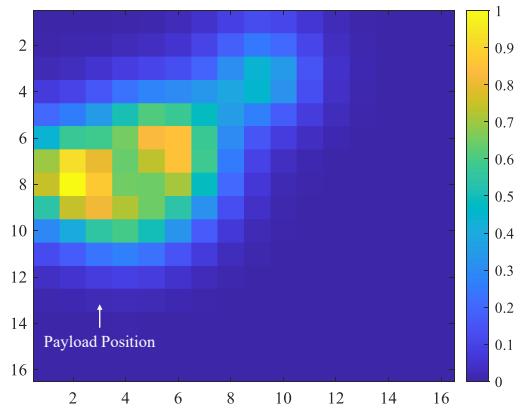
(b) Simulation of the environment with Agents' ALM turned on, illustrating enhanced visibility and separation of each agent.

Figure 4.15: 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.

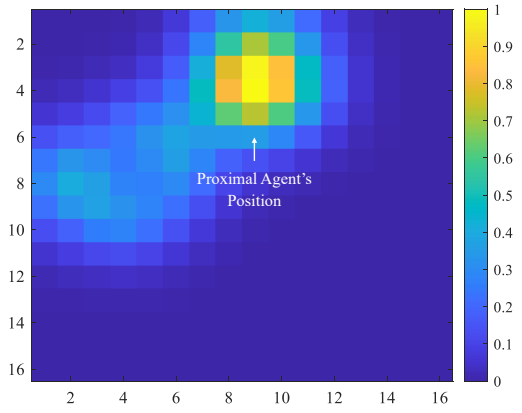
To visualize the activity patterns of the first hidden layer, the spiking responses of each repository are mapped onto a two-dimensional grid. Each repository consists of 256 neurons, which are spatially organized into a 16×16 grid such that the activity of each neuron is represented by a corresponding 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 4.16 demonstrates how the attention mechanism is modulated by the ALM. When the ALM is inactive (Figure 4.16a), 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 4.16b), the frequency-modulated blinking of the ALM redirects the attention toward the proximal agent. This shift occurs because the periodic light emission generates distinct bursts of DVS events, which, when synchronized with the inhibitory gating

connections from ALM neurons in the first hidden layer, suppress competing channels and allow the network to isolate the agent's position. In this way, the ALM not only enhances object visibility but also works in conjunction with the attention and inhibitory mechanisms to dynamically alternate focus between the payload and proximal agents.




(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 4.16: 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 under the Gaussian-weighted mapping.

Figure 4.17 illustrates the structural configuration of the SNN's first hidden layer under two conditions: with the agents' ALM turned off (Figure 4.17a) and with the ALM turned on (Figure 4.17b). In both scenarios, the first hidden layer contains two distinct repositories: one dedicated to payload detection and the other to 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, allowing the network to focus on learning representations related to the payload. Conversely, when the ALM is activated, it inhibits the payload detector repository, thereby releasing the proximal

agent detector repository from suppression.

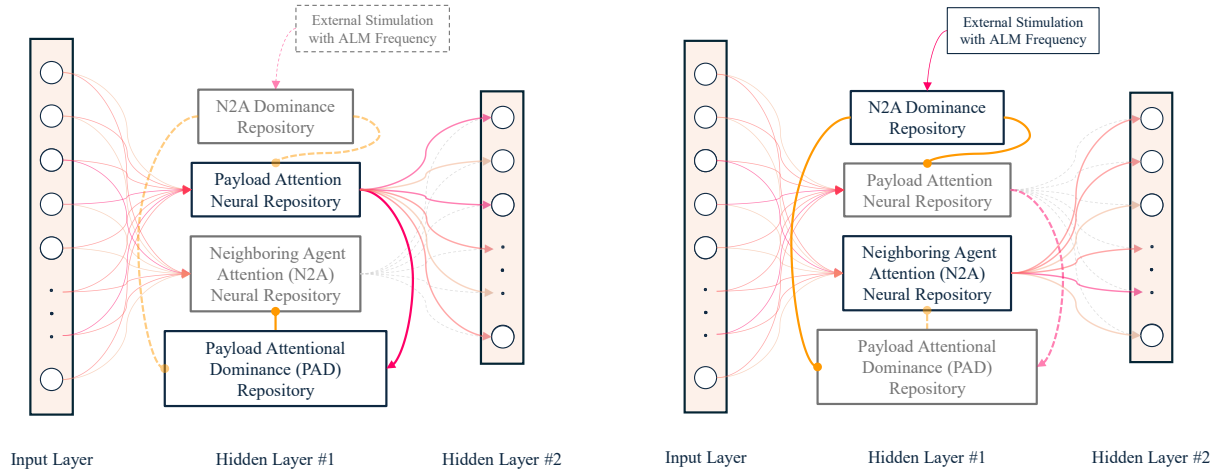
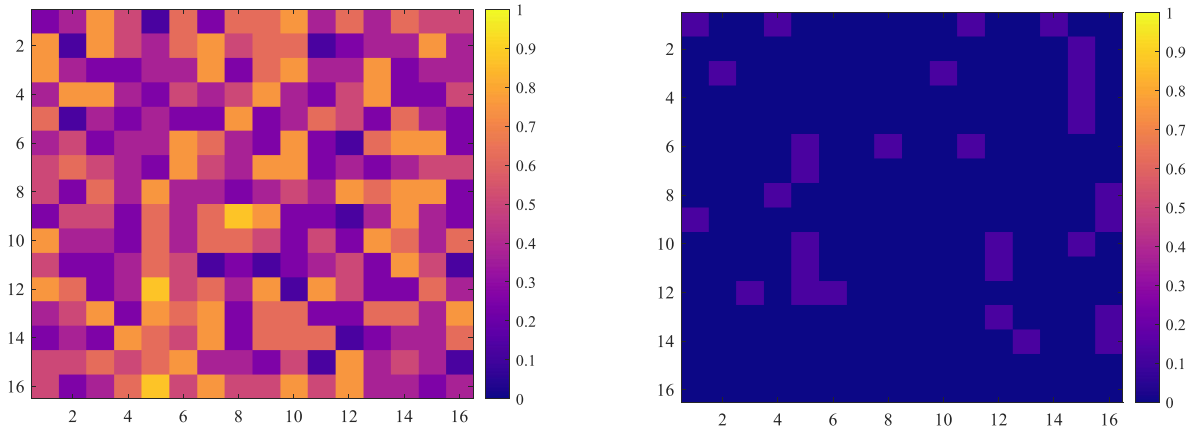



Figure 4.17: SNN structure in the first hidden layer: (a) with Agents' ALM turned off, focusing on payload detection; (b) with Agents' ALM turned on, emphasizing proximal agent detection. Inactive neuron repositories and the connections are shown in gray. The dotted lines are also inactive inhibitory connections.

Figure 4.18 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 **??**. As illustrated in Figure 4.18a, 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 4.18b). Its activity is suppressed by the inhibitory signals originating from the payload detector repository, which dominates the competition when the ALM is turned 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.

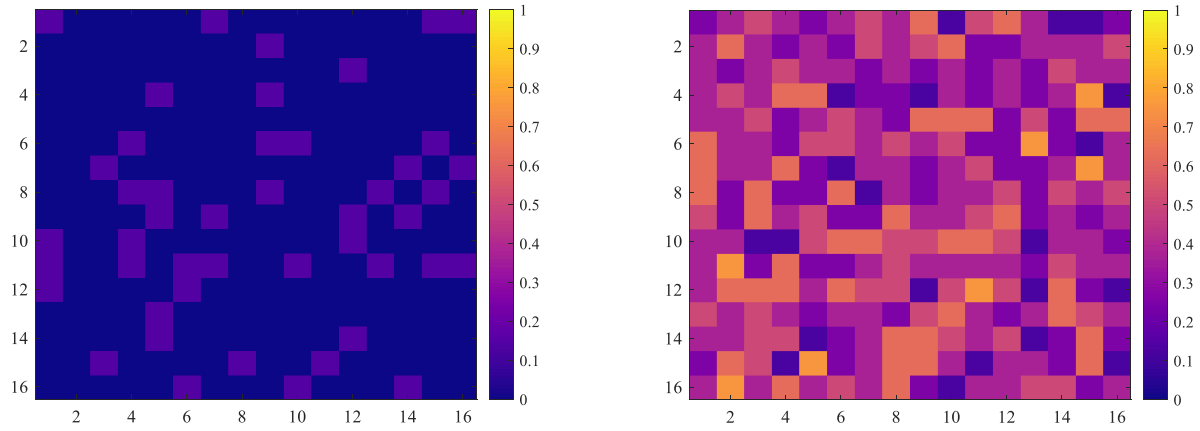


(a) Activity of the Payload detector repository during initial training phases with the Agents' ALM turned off.

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

Figure 4.18: Detector repository activity during initial training with ALM off. The payload detector remains active, while the proximal agent detector is suppressed due to inhibitory competition.

Figure 4.19 shows the complementary case to the previous condition, where the agents' ALM is active. As illustrated in Figure 4.19a, the payload detector repository becomes largely suppressed due to the inhibitory influence exerted by the ALM neurons. This inhibition effectively releases the proximal agent detector repository, which in turn exhibits strong activity (Figure 4.19b). In this phase, the R-STDP mechanism focuses synaptic plasticity on the connections between the input layer and the proximal agent repository, allowing the network to learn representations specific to the agents while preventing interference from the payload channel.

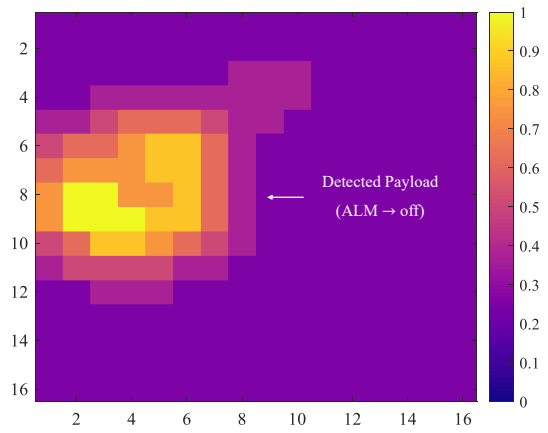


(a) Activity of the Payload detector repository during initial training phases with the Agents' ALM turned on.

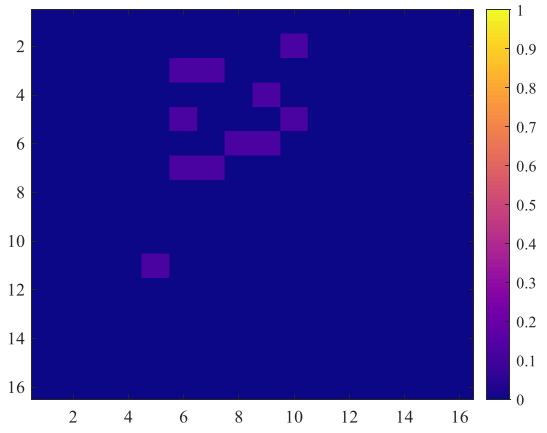
(b) Activity of the proximal agent detector repository during initial training phases with the Agents' ALM turned on.

Figure 4.19: 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 4.20 shows the activity of the first hidden layer repositories after convergence when the agents' ALM is turned off. As seen in Figure 4.20a, the payload detector repository has successfully learned to represent the payload, producing a dense and localized activation pattern that clearly highlights the detected object. In contrast, the proximal agent detector repository (Figure 4.20b) 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 is able to converge to a stable representation of the payload in the absence of ALM stimulation that shows the ability of the first hidden layer to encode and isolate salient objects once learning stabilizes.



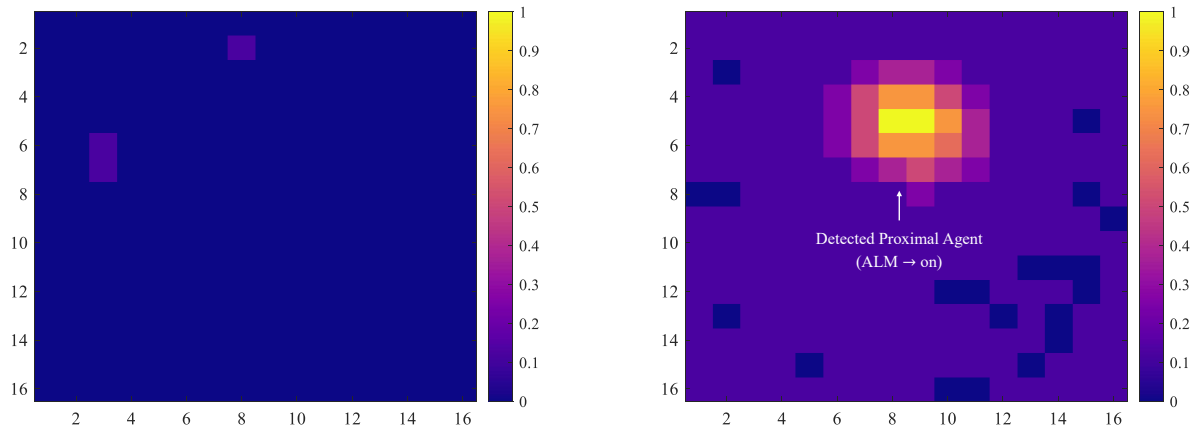

(a) Activity of the Payload detector repository after learning convergence with the Agents' ALM turned off.



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

Figure 4.20: Detector repository activity after learning convergence with ALM off. The payload detector shows strong, localized activity, while the proximal agent detector remains inactive.

Figure 4.21 illustrates the activity of the first hidden layer repositories after learning convergence when the agents' ALM is turned on. As shown in Figure 4.21a, the payload detector repository remains almost completely inactive due to inhibitory gating, while the proximal agent detector repository (Figure 4.21b) exhibits a clear, localized activation corresponding to the detected agent. This demonstrates that, once training converges, the network can reliably isolate and represent the proximal agent by leveraging the ALM-induced event bursts. The results highlight how the ALM not only enhances visibility but also enables the first hidden layer to switch attention between payload and agent detection, ultimately providing stable and distinct object representations.

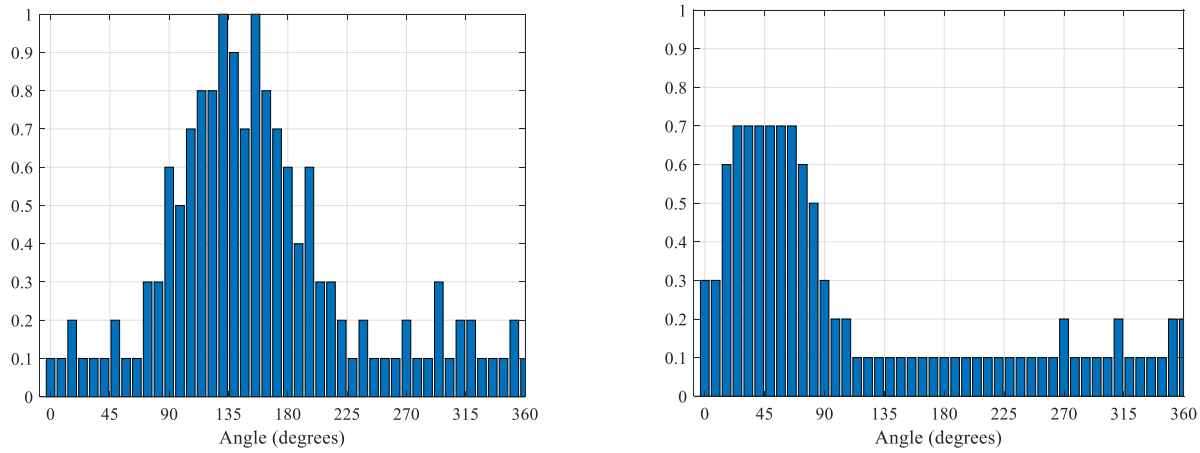



(a) Activity of the Payload detector repository after learning convergence with the Agents' ALM turned on.

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

Figure 4.21: Detector repository activity after learning convergence with ALM on. The proximal agent detector shows strong, localized activity, while the payload detector remains inactive.

Figure 4.22 presents the activity patterns of the second hidden layer repositories during both the formation and docking phases for Agent 1. In Figure 4.22a, during the formation phase, the FFM repository exhibits a pronounced activity peak around 135°, which aligns with the line-of-sight (LOS) direction from Agent 1 to the payload. This indicates that the FFM has effectively learned to orient its activity towards the payload, facilitating stable formation flying. Conversely, in Figure 4.22b, during the docking phase, the CDM repository shows a significant activity peak around 45°, corresponding to the LOS direction from Agent 1 to the docking point. This shift in activity demonstrates that the CDM has adapted its representation to focus on guiding the payload towards the docking location, highlighting the dynamic reconfiguration of repository activity in response to task demands.

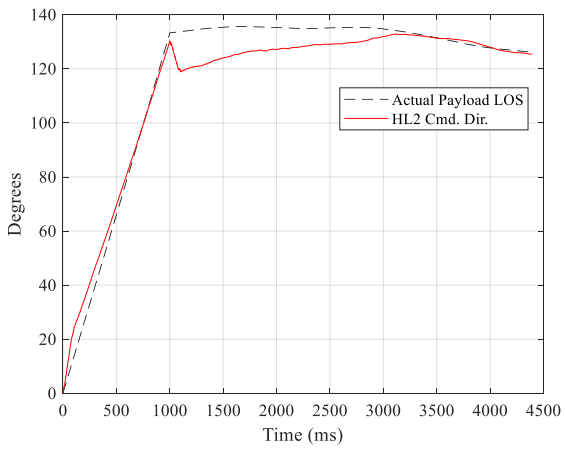



(a) Initial iterations of the formation phase.

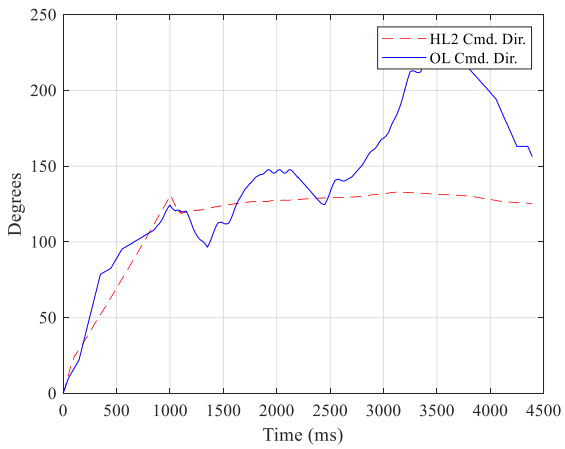
(b) Initial iterations of the docking phase

Figure 4.22: Second hidden layer activity for the formation and docking phases for Agent 1: (a) FFM activity 135 degree that matches the LOS of the Payload regarding the Agent 1 and (b) CDM activity shows 45 degree LOS toward the docking point.

Figures 4.23 and 4.24 display the activity of the second hidden layer and output layer during the formation and docking phases, respectively. In Figure 4.23a, during the formation phase, the hidden layer 2 activity closely tracks the actual payload trajectory. The corresponding output layer activity in Figure 4.23b 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 formation flying.




(a) Commanded velocity generated by the Hidden layer 2 activity during the formation phase vs the actual agent's LOS regarding the payload.



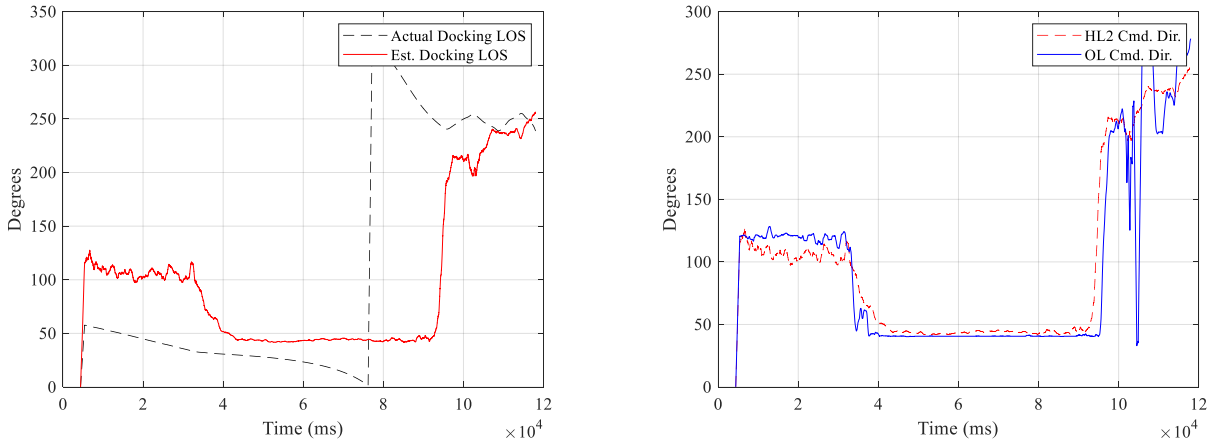

(b) Output layer activity during formation phase

Figure 4.23: Hidden layer 2 and output layer activity during the formation phase.

Figures 4.23(b) shows the commanded velocity generated by the output layer activity during the formation 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 formation with the payload. There is a lag at the end of the formation phase that is because the system is still learning to adjust the output layer weights to match the hidden layer 2 activity.

Similarly, in Figure 4.24a, 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 wrap-around, the second hidden layer learns to anticipate the change within the first 10 seconds of docking. The output layer activity in Figure 4.24b shows that it adapts itself with the hidden layer 2 activity, indicating successful learning of the docking maneuvers. The lag at the end of the mission is because of the system is still refining the output layer weights to align with the hidden layer 2 activity and the output layer tries to

adjust itself to the new docking LOS after the angle wrapping.

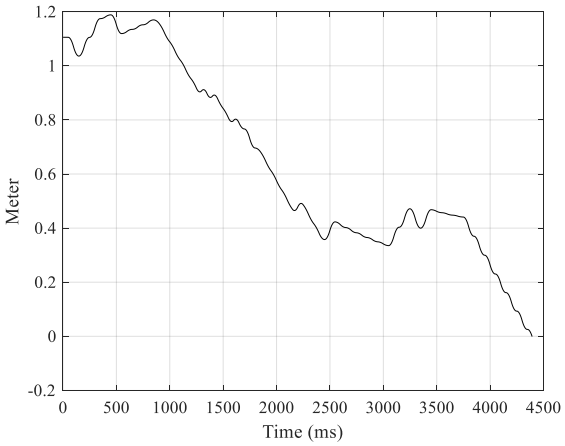



(a) Hidden layer 2 activity during docking phase

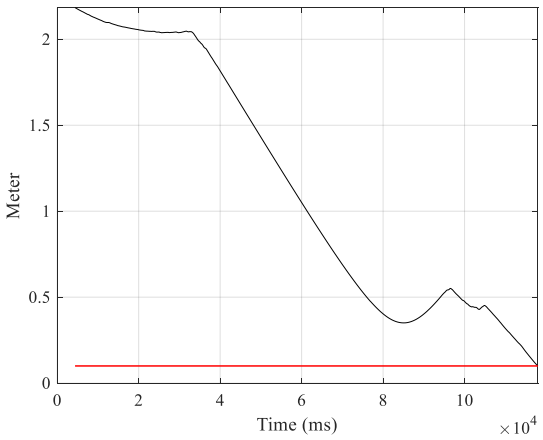
(b) Output layer activity during docking phase

Figure 4.24: Hidden layer 2 and output layer activity during the docking phase.

Figures 4.25 illustrates the distances during the formation and docking phases. In Figure 4.25a, Agent 1's distance from the payload decreases steadily during the formation phase, indicating successful approach towards the payload. Figure 4.25b shows the payload's distance from the docking center during the docking phase, demonstrating effective guidance towards the target location.



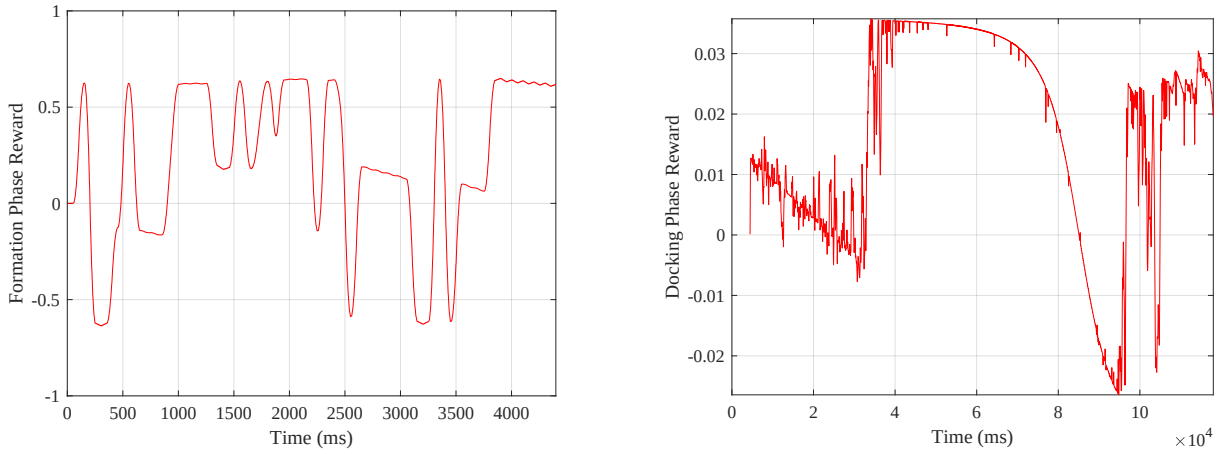

(a) Agent 1's distance from the payload during formation phase




(b) Payload's distance from the docking center

Figure 4.25: Distances during time.

Figure 4.26 presents the reward profiles during both the formation and docking phases. In Figure 4.26a, during the formation phase of the 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 collision with Agent 2 because it is getting closer to it at the same time. In Figure 4.26b, 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 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 4.24b.

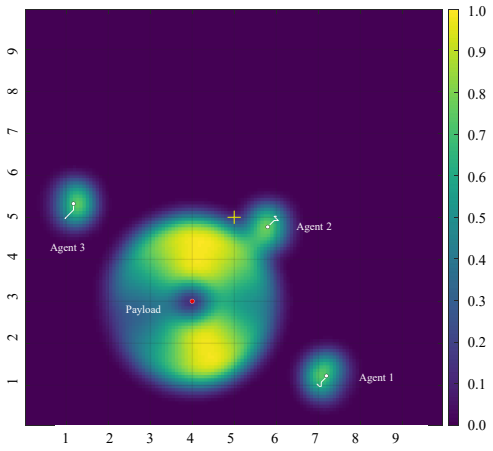



(a) Formation phase

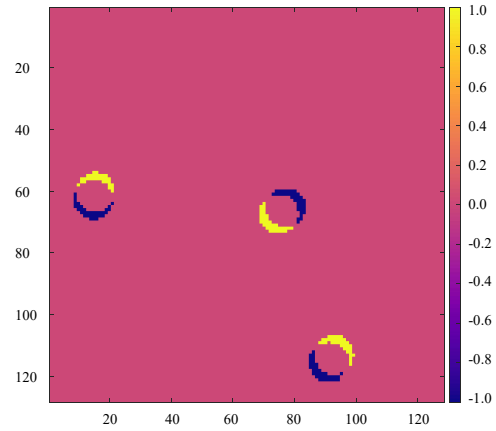
(b) Docking phase

Figure 4.26: Reward profile during the docking phase

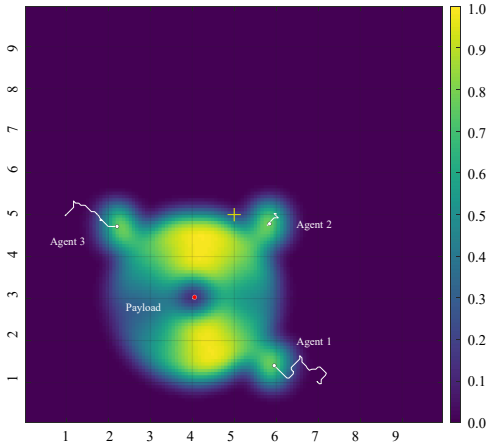
The cooperative docking sequence can be divided into three main phases: formation, docking approach, and final soft capture. Figures 4.27–4.29 illustrate these phases, showing both the spatial configuration of the agents and the corresponding event-based visual responses. The yellow crosshair in each frame indicates the center of the hub, providing a reference point for assessing alignment and positioning throughout the maneuver.



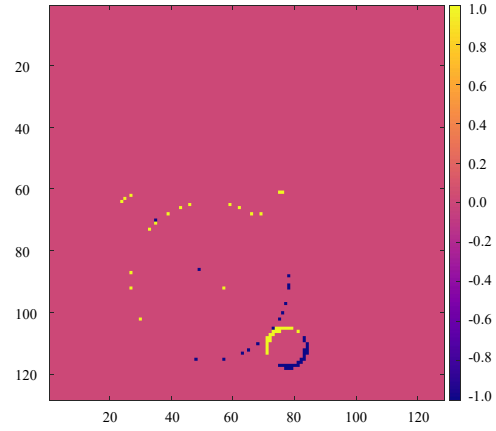

(a) View From the Hub Center at 5 seconds



(b) DVS View From the Hub Center at 5 seconds



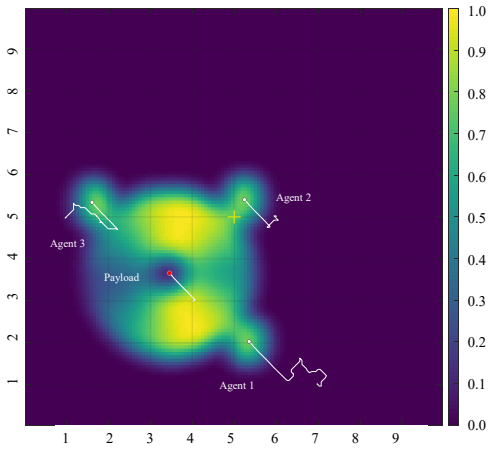

(c) View From the Hub Center at 15 seconds



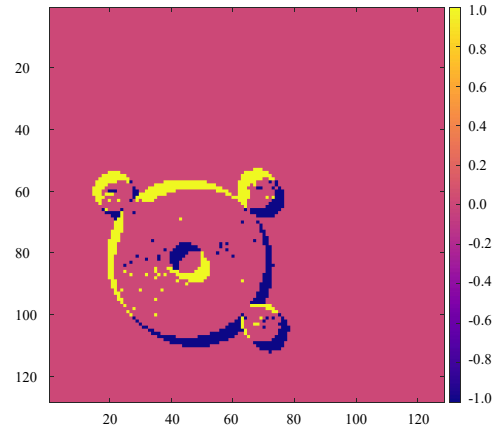
(d) DVS View From the Hub Center at 15 seconds

Figure 4.27: Views during the formation phase at different time steps.

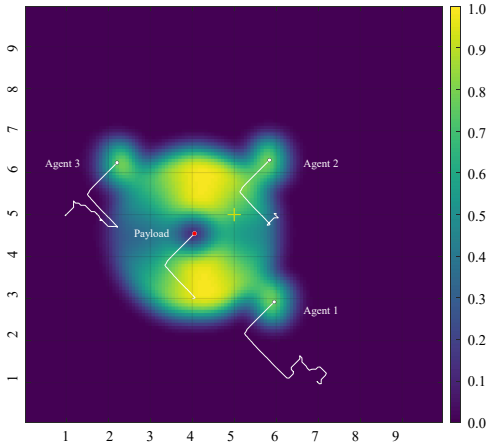
During the formation phase (Figure 4.27), the agents converge toward the payload and establish a formation around it. At 5 s and 15 s, the agents can be observed gradually aligning with the payload while reducing relative displacement. The corresponding DVS View From the Hub Centers captures the events generated by the agents' motion. These asynchronous events encode temporal luminance changes as positive and negative polarities that represent motion direction and relative displacement.



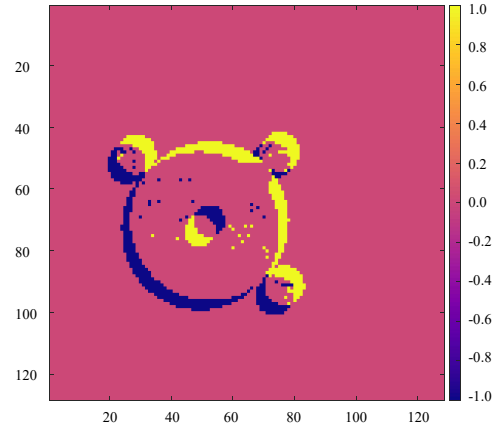

(a) View From the Hub Center at 30 seconds



(b) DVS View From the Hub Center at 30 seconds



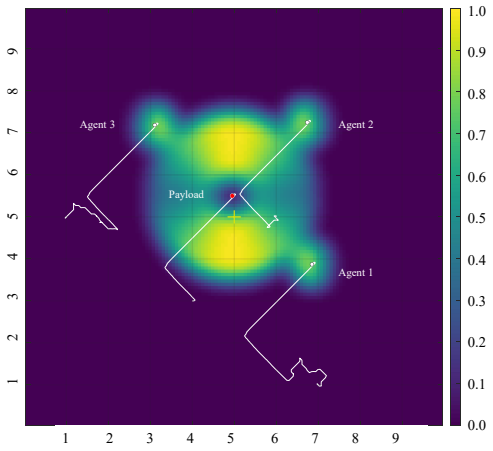

(c) View From the Hub Center at 60 seconds



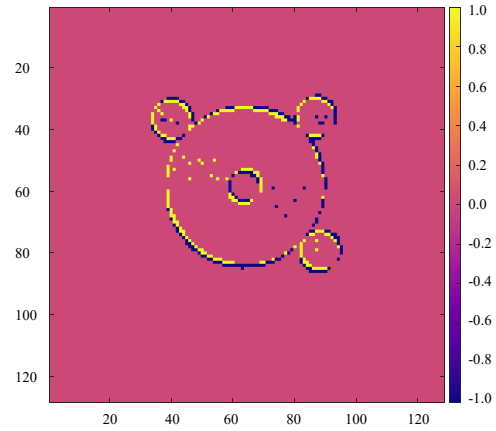
(d) DVS View From the Hub Center at 60 seconds

Figure 4.28: Views during the docking phase at different time steps.

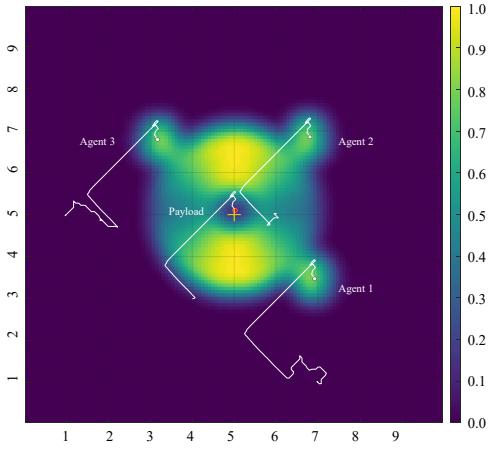
In Figure 4.28(a), the formation stabilizes and the payload starts moving by the agents. The agents begin moving the payload toward the docking center. The agents maintain synchronized thrust while compensating for small misalignments. However, after 30 seconds, the direction of movement produces a negative reward signal, making the agents adjust their thrust direction as shown in Figure 4.28(c).



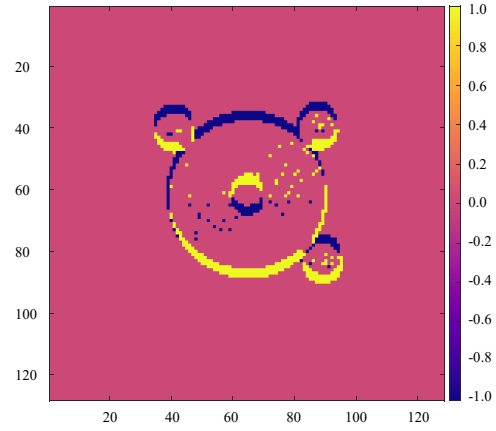

(a) View From the Hub Center at 99 seconds



(b) DVS View From the Hub Center at 99 seconds




(c) View From the Hub Center at docking completion



(d) DVS View From the Hub Center at docking completion

Figure 4.29: Views during the final approach and soft capture.

During the final phase of the maneuver (see Figure 4.29 (c)), the agents complete the docking sequence. At 99 seconds (Figure 4.29 (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 is less than 0.1 meters and the lateral velocity is under 0.04 meters per second. Overall,

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 work presents a novel approach to multi-agent cooperative docking using SNNs enhanced with adaptive light modulation. By integrating an attention mechanism with inhibitory gating, the network effectively isolates and represents both payload and proximal agents that enables dynamic switching of focus based on task demands. The use of R-STDP allows the network to learn robust representations for formation flying and docking maneuvers. Simulation results demonstrate the efficacy of the proposed architecture in achieving precise docking without direct state measurement (distances and relative velocities) under challenging lighting conditions, which highlights its potential for real-world applications in autonomous space operations.

The event-triggered mission phase switching in the second hidden layer repositories allows the agents to adaptively transition between formation and docking behaviors. The multi-scale synaptic time constant design further enhances the network's ability to capture temporal dynamics relevant to each phase. It lets SNNs capture high-frequency visual cues during formation while integrating longer-term information during actuation. 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 in controlling complex actuators like 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 in the system during missions and more complex docking scenarios will

further explore 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 Dynamic Vision Sensor (DVS) event streams and deep SNN controllers, achieving successful cooperative docking of a substantial payload within stringent performance criteria.
- Comprehensive evaluation of the proposed systems, demonstrating their effectiveness in real-world cooperative tasks and their resilience to operational constraints.

The results obtained from the simulations indicate that SNNs can significantly enhance the performance of multiagent systems in edge computing 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), transfer learning, and evolutionary optimization algorithms to optimize the global model.
- **Real-World Deployment:** Deveolping 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. This work aims to improve the efficiency of bio-inspired neural network models in edge computing and ensure the integrity and privacy of data in decentralized networks.

# 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] S. Dora and N. Kasabov, "Spiking neural networks for computational intelligence: an overview," *Big Data and Cognitive Computing*, vol. 5, no. 4, 2021.

[6] K. Xie, Z. Zhang, B. Li, J. Kang, D. Niyato, S. Xie, and Y. Wu, "Efficient federated learning with spike neural networks for traffic sign recognition," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 9, pp. 9980–9992, 2022.

[7] W. J. Yun, Y. Kwak, H. Baek, S. Jung, M. Ji, M. Bennis, J. Park, and J. Kim, "Slimfl: Federated learning with superposition coding over slimmable neural networks," *IEEE/ACM Transactions on Networking*, 2023.

[8] 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.

[9] 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.

[10] 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.

[11] L. Long and G. Fang, "A review of biologically plausible neuron models for spiking neural networks," *AIAA Infotech@ Aerospace 2010*, 2010.

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

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

[14] A. Triche, A. S. Maida, and A. Kumar, "Exploration in neo-hebbian reinforcement learning: Computational approaches to the exploration–exploitation balance with bio-inspired neural networks," *Neural Networks*, vol. 151, pp. 16–33, 2022.

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

[16] D. L. Young and C.-S. Poon, "Hebbian covariance learning: a nexus for respiratory variability, memory, and optimization?," *Advances in Modeling and Control of Ventilation*, pp. 73–83, 1998.

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

[18] T. Toyoizumi, J.-P. Pfister, K. Aihara, and W. Gerstner, "Generalized bienenstock–cooper–munro rule for spiking neurons that maximizes information transmission," *Proceedings of the National Academy of Sciences*, vol. 102, no. 14, pp. 5239–5244, 2005.

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

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

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

[22] A. L. Hodgkin and A. F. Huxley, "Currents carried by sodium and potassium ions through the membrane of the giant axon of loligo," *The Journal of physiology*, vol. 116, no. 4, 1952.

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

[24] S. Song, K. D. Miller, and L. F. Abbott, "Competitive hebbian learning through spike-timing-dependent synaptic plasticity," *Nature Neuroscience*, vol. 3, no. 9, pp. 919–926, 2000.

[25] H. Z. Shouval, S. S.-H. Wang, and G. M. Wittenberg, "Spike timing dependent plasticity: a consequence of more fundamental learning rules," *Frontiers in computational neuroscience*, vol. 4, 2010.

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

[27] V. Vassiliades, A. Cleanthous, and C. Christodoulou, "Multiagent reinforcement learning: spiking and nonspiking agents in the iterated prisoner's dilemma," *IEEE transactions on neural networks*, vol. 22, no. 4, pp. 639–653, 2011.

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

[29] D. Probst, M. A. Petrovici, I. Bytschok, J. Bill, D. Pecevski, J. Schemmel, and K. Meier, "Probabilistic inference in discrete spaces can be implemented into networks of lif neurons," *Frontiers in computational neuroscience*, vol. 9, 2015.

[30] U. Markowska-Kaczmar and M. Koldowski, "Spiking neural network vs multilayer perceptron: who is the winner in the racing car computer game," *Soft Computing*, vol. 19, pp. 3465–3478, 2015.

[31] S. Guo, Z. Yu, F. Deng, X. Hu, and F. Chen, "Hierarchical bayesian inference and learning in spiking neural networks," *IEEE transactions on cybernetics*, vol. 49, no. 1, pp. 133–145, 2017.

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

[33] Z. Bing, C. Meschede, G. Chen, A. Knoll, and K. Huang, "Indirect and direct training of spiking neural networks for end-to-end control of a lane-keeping vehicle," *Neural Networks*, vol. 121, pp. 21–36, 2020.

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

[35] L. Salt, D. Howard, G. Indiveri, and Y. Sandamirskaya, "Parameter optimization and learning in a spiking neural network for uav obstacle avoidance targeting neuromorphic

processors," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 9, pp. 3305–3318, 2019.

[36] J. Liu, H. Lu, Y. Luo, and S. Yang, "Spiking neural network-based multi-task autonomous learning for mobile robots," *Engineering Applications of Artificial Intelligence*, vol. 104, 2021.

[37] H. Lu, J. Liu, Y. Luo, Y. Hua, S. Qiu, and Y. Huang, "An autonomous learning mobile robot using biological reward modulate stdp," *Neurocomputing*, vol. 458, pp. 308–318, 2021.

[38] Q. Zhan, G. Liu, X. Xie, G. Sun, and H. Tang, "Effective transfer learning algorithm in spiking neural networks," *IEEE Transactions on Cybernetics*, vol. 52, no. 12, pp. 13323–13335, 2021.

[39] D. Chu and H. Le Nguyen, "Constraints on hebbian and stdp learned weights of a spiking neuron," *Neural Networks*, vol. 135, pp. 192–200, 2021.

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

[41] B. Golosio, G. Tiddia, C. De Luca, E. Pastorelli, F. Simula, and P. S. Paolucci, "Fast simulations of highly-connected spiking cortical models using gpus," *Frontiers in Computational Neuroscience*, vol. 15, 2021.

[42] X. 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] H. Liang, G. Liu, H. Zhang, and T. Huang, "Neural-network-based event-triggered adaptive control of nonaffine nonlinear multiagent systems with dynamic uncertainties," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 5, pp. 2239–2250, 2021.

[49] Q. Shen, P. Shi, J. Zhu, S. Wang, and Y. Shi, "Neural networks-based distributed adaptive control of nonlinear multiagent systems," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 3, pp. 1010–1021, 2020.

[50] 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.

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] 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.

[56] E. Narváez, A. A. Ravankar, A. Ravankar, T. Emaru, and Y. Kobayashi, "Autonomous vtol-uav docking system for heterogeneous multirobot team," *IEEE Robotics and Automation Letters*, 2021.

[57] 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.

[58] 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.

[59] F. Jia, M. Afaq, B. Ripka, Q. Huda, and R. Ahmad, "Vision- and lidar-based autonomous docking and recharging of a mobile robot for machine tending in autonomous manufacturing environments," *IEEE Transactions on Industrial Informatics*, 2023.

[60] J. S. Oh and M. Y. Kim, "Regression-based docking system for autonomous mobile robots using a monocular camera and aruco markers," *IEEE Transactions on Automation Science and Engineering*, 2025.

[61] G. Tang, N. Kumar, R. Yoo, and K. Michmizos, "Deep reinforcement learning with population-coded spiking neural network for continuous control," in *Conference on robot learning*, pp. 2016–2029, PMLR, 2021.

[62] 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.

[63] 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.

[64] 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.

[65] 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.

[66] 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.

[67] 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.

[68] 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.

[69] Y. Yang, R. M. Voyles, H. H. Zhang, and R. A. Nawrocki, "Fractional-order spike-timing-dependent gradient descent for multi-layer spiking neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 36, no. 2, pp. 1120–1133, 2025.

[70] 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.

[71] Z. Huang, J. Ding, Z. Pan, H. Li, Y. Fang, Z. Yu, and J. K. Liu, "Converting high-performance and low-latency snns through explicit modeling of residual error in anns," *Neural Networks*, vol. 161, pp. 345–358, 2025.

[72] W. Xu, Y. Li, G. Chen, and L. Shi, "Diet-snn: Deep-learning-inspired energy-efficient training of spiking neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 10, pp. 4567–4579, 2021.

[73] 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.

[74] 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.

[75] 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.

[76] 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.

[77] 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.

[78] 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.

[79] 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.

[80] A. Vicente-Sola, D. L. Manna, P. Kirkland, G. D. Caterina, and T. Bihl, "Spiking neural networks for event-based action recognition: A new task to understand their advantage," *Neurocomputing*, vol. 611, 2024.

[81] 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.

[82] 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.

[83] 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.

[84] 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.

[85] 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.

[86] 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.

[87] 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.

[88] 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.

[89] 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.

[90] 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.

[91] 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.

[92] 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.

[93] 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.

[94] 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.

[95] 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.

[96] A. B. Mansour, G. Carenini, A. Duplessis, and D. Naccache, "Federated learning aggregation: New robust algorithms with guarantees," in *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 721–726, IEEE, 2022.

[97] 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.

[98] 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.

[99] 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.

[100] G. Xia, J. Chen, C. Yu, and J. Ma, "Poisoning attacks in federated learning: A survey," *IEEE Access*, vol. 11, pp. 10708–10722, 2023.

[101] 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.

[102] 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.

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

[104] 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.

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

[106] 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.

[107] 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.

[108] 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.

[109] 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.

[110] 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.

[111] 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.

[112] 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.

[113] 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.

[114] R. Graca and T. Delbruck, "Towards a physically realistic computationally efficient dvs pixel model," *arXiv preprint arXiv:2505.07386*, 2025.

[115] A. Botelho, P. Rosa, and J. M. Lemos, "Explicit spacecraft thruster control allocation with minimum impulse bit," *IEEE Transactions on Control Systems Technology*, 2024.

[116] Y. Mei, X. Zhou, and X. Tan, "Modeling and mixed-integer nonlinear mpc of positive-negative pressure pneumatic systems," *arXiv preprint arXiv:2510.00433*, 2025.

[117] E. M. Izhikevich, "Which model to use for cortical spiking neurons?," *IEEE transactions on neural networks*, vol. 15, no. 5, pp. 1063–1070, 2004.

[118] SpaceX, "Dragon." Online: Available at https://www.spacex.com/vehicles/dragon/, 2025. Accessed: 27-Oct-2025.

[119] 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/.