Multi-agent cooperation on edge with Spiking Neural Network and reinforcement learning

by

Mohammad Tayefe Ramezanlou

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

Carleton University
Ottawa, Ontario
Spring, 2024

Abstract

Contents

1 Introduction **1**

1.1 Spiking Neural Networks: Models and Learning Algorithms 1

1.1.1 Neuron model 1

1.1.2 Training Algorithms 5

1.1.3 Discussion 7

1.2 Federated Learning for SNNs: Consensus Flying Scenario 8

2 Literature review **9**

3 Control and Navigation of the Leader Drone Over 5G Network **18**

3.1 Introduction 18

3.2 Quadrotor dynamic model 20

3.3 Linear PD Controller 22

3.4 Backstepping Controller 23

3.5 State Estimator 25

3.6 Delay Estimator 27

3.7 Results 28

3.8 Experimental setup 38

3.9 Conclusion 38

4 Modular Learning in SNNs for Optimal Multi-Agent Decision-Making **40**

4.1 Introduction 40

4.2 The ATD problem and SNN-based solution 41
4.3 Learning using R-STDP 42
4.4 Network structure and encoding method 46
4.5 Results 50
4.5.1 Simulation without noise 53
4.5.2 Simulation with noise 56
4.6 Conclusion 58

5 Integration of R-STDP and Federated Learning 59
5.1 Consensus Flying Problem 59
5.2 Proposed Method 61
5.2.1 Network Structure 61
5.2.2 Training algorithm 65
5.2.3 Weight Stabilization using Reward-Modulated Competitive Synaptic Equilibrium (R-CSE) 67
5.2.4 Federated Learning for Consensus Flying 70
5.3 Results and Discussion 74
5.3.1 Simulation without FL 74
5.3.2 Simulation with FL and R-CSE 84
5.4 Conclusion 87

6 Proposed Method 89

List of Figures

3.1 Control framework for quadrotor control using cellular networks 19
3.2 Quadrotor's dynamic model 20
3.3 State estimation error with perfect delay estimator 30
3.4 Control signals with perfect delay estimator 31
3.5 State and delay estimation error for $D = (6, 7, 9, 10)\ ms$ 32
3.6 Control signals for $D = (6, 7, 9, 10)\ ms$ 33
3.7 State and delay estimation error for $D = \{1, 3, 13, 15\}\ ms$ 34
3.8 Control signals for $D = \{1, 3, 13, 15\}\ ms$ 35
3.9 State and delay estimation error when delays and their distribution changed. 36
3.10 Control signals when delays and delay distributions changed 37

4.1 Active target defense game with three agents (LOS angles are shown for both agents). 42
4.2 Two Spiking Neural Network (SNN)s simultaneously play and control the invader and defender. Each agent uses the LOS angle and relative velocities for training. 43
4.3 Relative velocities and LOS angles used in the reward function 45

4.4 Network structure and encoding process for the input layer (Defender). Each neuron is associated with a membership function in Gaussian Receptive Fields (GRF). The Gaussian Receptive Fields (GRF) encodes an input State (S^t) at each time step. There is both a training phase when ($\alpha = 0$) and an operating phase when ($\alpha = 1$). 47

4.5 Spiking Neural Network (SNN)'s performance during training. Hollow circles show the initial positions. 51

4.6 Changes in synaptic weights during training. Only Maximum synaptic weights for both agents are shown. 52

4.7 An illustration of input current to Spiking Neural Network (SNN) and synaptic current to the output neurons after training for a single connection. 54

4.8 Reward, eligibility trace, and weight change during the simulation for W^{3-3}. The $C(t)R(t)$ changes the synaptic weight by considering activation strength and reward value. 54

4.9 Spiking Neural Network (SNN)'s performance after training. The highlighted region shows the defender's dominant region. The purple dot is the optimal capture point. 55

4.10 Spiking Neural Network (SNN)'s performance in noisy conditions. A white Gaussian noise with a variance of 0.01 is added to the measured inputs. . . . 57

5.1 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. 60

5.2 Spiking Neural Network (SNN) structure with encoding and decoding layers. Each sub-layer consists of a fuzzy encoder and a Fuzzy-to-Spiking Converter, with the output layer receiving inputs from synaptic weights and a random action selector. During the training phase, the output layer receives input only from the random action selector, which then shifts to synaptic weight inputs after the training. 62
5.3 The fuzzy encoding principle for the input sub-layer. 64
5.4 The measured distances during the test for evaluating the performance and detecting the collisions. 76
5.5 Distances during the test phase (Reward-Modulated Competitive Synaptic Equilibrium (RCSE) method) 79
5.6 Agents' trajectory during the test phase (Reward-Modulated Competitive Synaptic Equilibrium (RCSE) method) 79
5.7 Distances during the test phase (FACL method) 79
5.8 Agents' trajectory during the test phase (FACL method) 79
5.9 Change in commanded distance during the test phase (Reward-Modulated Competitive Synaptic Equilibrium (RCSE) method) 80
5.10 Change in commanded distance during the test phase (FACL method) 80
5.11 Agents' path after reward change (test phase) 82
5.12 Distances after reward change (test phase) 82
5.13 Synaptic Weights before Reward change 83
5.14 Synaptic Weights after Reward change 83
5.15 Synaptic weights increase after reward change 84
5.16 Synaptic weights decrease after reward change 84
5.17 Distances during the test phase before reward change in the proposed aggregation method. 85
5.18 Distances during the test phase after reward change in the proposed aggregation method. 85

5.20 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. 86

5.19 Frobenius norm of the agents during the learning phase. The reward changes for the Leader after 600 s. 86

List of Tables

3.1 Gains used in PD and backstepping controllers 29
3.2 Quadrotor specifications and state estimator constants 29
3.3 True and estimated transition probabilities 32

4.1 Parameter values for LIF neuron model [90] 50
4.2 Parameter values for Reward-modulated Spike-Timing-Dependent Plasticity (R-STDP) 51

5.2 Simulation Parameters 75
5.1 Parameter values for Leaky Integrate-and-Fire (LIF) neuron model [95] . . . 75
5.3 Simulation Parameters for the FACL Algorithm 78
5.4 Comparative Performance Analysis of the Proposed Aggregation Algorithm+Reward-Modulated Competitive Synaptic Equilibrium (RCSE), Reward-Modulated Competitive Synaptic Equilibrium (RCSE), and FACL 87

Acronyms

FACL Fuzzy Actor-Critic Learning

FL Federated Learning

GRF Gaussian Receptive Fields

LIF Leaky Integrate-and-Fire

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

RCSE Reward-Modulated Competitive Synaptic Equilibrium

SNN Spiking Neural Network

UAV Unmanned Aerial Vehicle

UGV Unmanned Ground Vehicle

Chapter 1

Introduction

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 Spiking Neural Network (SNN), various neuron models have been proposed to simulate the electrical activity of neurons. These models range from simple to complex, aiming to capture the essential features of neuronal dynamics. This section provides an overview of several key neuron models, including their equations and characteristics.

Leaky-Integrate and Fire (LIF) model

The LIF model is a biological model that can be represented as a circuit with a resistor and capacitor and represents a first-order dynamic system [1],

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

where $V_m(t)$ is the neuron's membrane potential, R_m is the membrane resistance, C_m is the

membrane capacitance, E_l is the resting potential, and $I(t)$ is the input current. The neuron spikes when its potential reaches the threshold potential (V_{th}). The potential of the neuron immediately reaches the reset potential (V_{res}) after it spikes.

The spike rate is a parameter that determines how fast the neuron spikes [2].

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

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

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

where τ_m is the membrane time constant.

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

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

or

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

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

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

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

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

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

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

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

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

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

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

Hodgkin-Huxley Model

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

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

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

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

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

where C_m is the membrane capacitance, V is the membrane potential, I_{ext} is the external current, $\bar{g}_i$ and V_i are the maximum conductances and reversal potentials for the K^+, Na^+, and leak (L) currents, and n, m, and h are gating variables. The variables n, m, and h are dimensionless and vary between 0 and 1, representing the proportion of ion channels in a

particular state. The dynamics of these gating variables are critical for the model's ability to replicate the complex temporal patterns of neuronal action potentials.

FitzHugh-Nagumo Model

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

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

$$\frac{dW}{dt} = 0.08(V + 0.7 - 0.8W) \tag{1.15}$$

where V represents the membrane potential, W is a recovery variable, and I_{ext} is the external current.

Izhikevich Model

The Izhikevich model combines the biological plausibility of the Hodgkin-Huxley model with the computational efficiency of integrate-and-fire models:

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

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

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

1.1.2 Training Algorithms

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

The Reward-modulated Spike-Timing-Dependent Plasticity (R-STDP) algorithm is a learning technique inspired by biological processes in the brain, which is believed to be fundamental to certain learning processes [3]. The algorithm's core principle is that when a pre-synaptic neuron activates just before its post-synaptic counterpart, the synapse's strength connecting them should increase, and vice versa if the post-synaptic neuron fires first.

Within the SNN framework, pre-synaptic neurons are the input neurons, and post-synaptic neurons function as the output neurons. The function $STDP(\tau)$ can be defined as the firing timelines of both input and output neurons [4] as,

$$STDP_{kl}(\tau) = \mathcal{A} \exp\left(-\frac{\tau}{\tau_s}\right) \text{ for } \tau \geq 0 \tag{1.18}$$

where $\mathcal{A}$ stands as the exponential function's amplitude, and τ is the difference between the firing time of the input neuron (k) and output neuron (l). Meanwhile, τ_s acts as the time constant, setting the decay rate for the $R - STDP$ function. Should τ_s approach infinity, the exponential function converges to 1, neutralizing time's effect on the $R - STDP$ function.

The adjustment of synaptic weights follows the given equation:

$$\dot{W}_{kl}(t) = STDP_{kl}(\tau)\mathcal{R}(t) \tag{1.19}$$

where $\dot{W}_{kl}(t)$ denotes the rate of change of the synaptic weight that connects neurons k and l. This weight determines the input that the post-synaptic neuron receives upon the spiking of its pre-synaptic neuron, which is quantified as $I(t)$.

Temporal Difference Learning (TD-Learning)

TD-Learning for SNNs employs the concept of temporal difference errors to adjust synaptic weights, facilitating the learning of predictions about future rewards:

$$\Delta W_{kl} = \alpha \cdot (r + \gamma \cdot V(s') - V(s)) \cdot STDP_{kl}(\tau) \tag{1.20}$$

Here, α is the learning rate, r represents the immediate reward, and γ is the discount factor for future rewards. $V(s)$ and $V(s')$ denote the values of the current and subsequent states, respectively.

Deep Q-Learning for Spiking Neural Networks (DQSNN)

DQSNN integrates the principles of Deep Q-Learning with the dynamics of SNNs, allowing for the application of deep RL strategies:

$$\Delta W = \alpha \cdot (r + \gamma \cdot \max_{a'} Q(s', a'; \theta') - Q(s, a; \theta)) \cdot \nabla_\theta Q(s, a; \theta) \tag{1.21}$$

This equation updates synaptic weights ΔW by optimizing the Q-value function $Q(s, a; \theta)$ with respect to the network parameters θ, guided by the prediction error between expected and obtained rewards.

Policy Gradient Methods for SNNs

Policy Gradient Methods directly optimize the policy function $\pi(a|s; \theta)$, encouraging actions that lead to higher rewards:

$$\Delta\theta = \alpha \cdot \mathbb{E}[\nabla_\theta \log \pi(a|s; \theta) \cdot R] \tag{1.22}$$

where R is the reward associated with taking action a in state s, and $\mathbb{E}[]$ denotes the expectation over the distribution of actions.

Spiking Actor-Critic (SAC)

SAC employs an actor-critic architecture, with both components modeled as SNNs. The actor updates its policy based on the critic's value function estimates:

$$\Delta\theta_{actor} = \alpha \cdot \nabla_\theta \log \pi(a|s;\theta) \cdot A(s,a) \tag{1.23}$$

$$\Delta\theta_{critic} = \beta \cdot (r + \gamma \cdot V(s') - V(s)) \cdot \nabla_\theta V(s) \tag{1.24}$$

where $A(s,a)$ is the advantage function, indicating the benefit of taking action a in state s, and β is the learning rate for the critic.

Spiking Proximal Policy Optimization (SPPO)

SPPO adapts the Proximal Policy Optimization algorithm for SNNs, optimizing a clipped surrogate objective to improve learning stability:

$$L(\theta) = \mathbb{E}\left[\min\left(r_t(\theta) \cdot A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) \cdot A_t\right)\right] \tag{1.25}$$

where $r_t(\theta)$ is the probability ratio of the current and old policies, and ϵ is a hyperparameter defining the clipping range.

1.1.3 Discussion

The development of RL-based algorithms for SNNs represents a significant advancement in leveraging the computational properties of spiking neurons for learning complex behaviors. By incorporating reward signals, prediction errors, and policy optimization strategies, these algorithms enable SNNs to adaptively refine their synaptic connections, fostering the emergence of intelligent decision-making capabilities. The continuous evolution of these algorithms promises to further unlock the potential of SNNs in a wide array of applications, from autonomous systems to advanced cognitive modeling.

1.2 Federated Learning for SNNs: Consensus Flying Scenario

Chapter 2

Literature review

During the past years, researchers have focused on quadrotor-type UAVs because of their ability to hover, take off vertically, and land in places that conventional fixed-wing aircraft cannot. They can accomplish a wide range of long-distance missions at low costs [5]. Several linear and nonlinear control algorithms have been developed, and the most common approach is hierarchical control that involves an inner and outer loop [6].

In the linear PID controller, one can design the controller feedback gains based on the linear model to achieve desired performance [7][8]. The Linear-Quadratic Regulator (LQR) has demonstrated acceptable results [9], but finding the required control parameters (weight matrices) is a challenging problem.

The H_∞ approach is widely used [10] for cases with unmodeled dynamics since it eliminates the errors in modelling. However, adjusting the controller parameters is a difficult task that makes the implementation complex to some degree in comparison with the PID and LQR [11].

The quadrotor's control process is a rather complex task because of the nonlinearities and uncertainties (e.g., the moment of inertia of rotors and aerodynamic effects). Sliding Mode Control (SMC) can be used for the flight condition with time-varying disturbances [12] since it quickly compensates for external disturbances and eliminates the effect of uncertainties in modelling [13]. However, chattering and finite-time convergence are always a problem [14].

The backstepping control approach breaks the control law into parts. The control laws are derived using a suitable Lyapunov function [15] to compensate for the problems that exist in the underactuated systems. Although an optimization process can be done to find the optimal gains, it would lead to a time-consuming task if it is done for every drone with different dynamic parameters [16].

All the above-mentioned control algorithms have been developed, considering there is no time delay in the control loop. Drone control in Beyond-Visual-Line-Of-Sight (BVLOS) suffers from many problems caused by random network delays and packet losses [17][18]. Time delay in the feedback loop can make the control system unstable [19].

Control approaches based on the Lyapunov function usually incorporate the time delay into the candidate function [20]. Another approach to compensate for the delay is the Smith predictor, which improves the quadrotors tracking performance by reducing the destabilizing effect of time delay [21]. However, it does not provide any information about the current state of the plant for the remote control station.

In [22], a predictor-based control approach is developed for quadrotor control. The delay was assumed to be constant, and the states of the system were predicted using a discrete-time predictor. A virtual quadrotor model worked in parallel with the actual setup was used to simulate the remote system [23].

Traditional techniques define a model for the time delay or consider it a constant value [24] [25]. Estimating time delay with Machine Learning (ML) algorithms requires high computation capability and sufficient training data, which is unavailable at the beginning of the quadrotor flight [26] [27].

Although previous studies have tried to compensate for time delay by predicting the drone's future states, in the case of 5G networks, the quadrotor's current states should be accessible for the base station to guarantee high communication bandwidth using better beamforming.

Spiking Neural Networks (SNNs) are the third generation of neural networks that have gained attention for their lower power consumption due to their event-driven nature and

sparse representation of data, which is based on real biological neurons [28–30]. The SNNs work based on spikes. Since the spikes are not continuous, only a part of the network is active at a given time. Also, successful hardware implementation of SNNs on neuromorphic chips like TrueNorth [31] and Loihi [32] makes them good candidates for robotic applications.

Studies have shown that the SNN can learn particular tasks autonomously to control robotic platforms. In [33], a Reward Modulated Spike-Timing-Dependent Plasticity (R-STDP) algorithm is used to control a mobile robot. Different States of the mobile robot (e.g., acceleration and deceleration) are defined as different frequencies for the input sensory layer. The SNN learns to navigate autonomously in the environment while avoiding obstacles.

There are several successful implementations of SNNs in learning nonlinear dynamics. In [34], a chaotic reservoir of spiking neurons is trained to control a robotic arm adaptively. The network is able to follow the desired trajectory with few neurons. SNN's capability in controlling a robotic arm with three degrees of freedom is also studied in [35], where the SNN has 218 neurons. The results are compared with real behavioural and neural data. The results showed that the SNN can adapt the Jacobian matrix when dynamic properties are changed over time.

In [36], an SNN is trained using the R-STDP algorithm and lateral inhibition mechanism to control a mobile robot in an environment with an obstacle. The lateral inhibition helps SNNs switch between target tracking and obstacle avoidance tasks during and after learning. The results showed that the SNNs have the potential to take control in complex scenarios.

In [37], a method based on the reinforcement learning algorithm is proposed to help an invader minimize its distance from a territory before being captured. In [38], an interceptor is used to defend a target from an attacking intruder. This type of differential game is addressed as Defending an Asset, which is solved using Linear Quadratic formulation. In [39], the problem is solved for multi-player cases but split into two simple two-player games.

Multiple attackers-defenders problems with a stationary target are studied in [40]. However, a simple single attacker-single defender case is solved because of the complexity of the numerical solution. Several studies have tried to propose optimal solutions for the ATD

problem. A set of optimal solutions are proposed in [41] considering reduced State space conditions. Using the geometric method, the attacker played an optimal strategy, capturing the target while maximizing its distance from the defender.

The consensus problem in flying multi-agent systems, often termed the "flocking" or "swarming" challenge, is a critical area of research in aerial robotics. It deals with coordinating and controlling multiple flying agents, ensuring they can work cooperatively without collisions and align towards a common objective. Achieving this consensus is important for different applications, from coordinated surveillance to communication, logistics, and infrastructure monitoring.

Federated Learning (FL) is an emerging paradigm in machine learning that allows for decentralized training of models across multiple agents without centralizing data [42]. Using this approach, robots can benefit from shared experiences while preserving data privacy and reducing communication overheads [43, 44].

The challenge of optimizing communication efficiency in FL is addressed in [44], where a hierarchical approach is introduced, leveraging adaptive staleness control. This method emphasizes that both system-level and data-level heterogeneity in FL must be considered. The complexities of wireless network constraints in FL are explored in another study, where the focus is placed on optimizing the convergence time. This research indicates that the wireless environment introduces a layer of complexity to the FL framework [45].

The practical applicability of FL in real-world scenarios is demonstrated in [46], which focuses on hierarchical trajectory planning for narrow-space automated parking. This study highlights the vast potential of FL in automation and robotics. An interesting direction in which neuromorphic learning and FL converge is presented in [47], suggesting future trends where bio-inspired computing and FL might merge. A holistic view of the advancements, challenges, and future directions in FL is provided in a comprehensive survey, making it clear that FL is poised to reshape the landscape of machine learning [48].

The aforementioned applications demand real-time decision-making and adaptability to dynamic environments. The Reinforcement Learning (RL) algorithms offer a promising

approach to address these challenges by enabling agents to learn optimal policies through interactions with their environment, enabling them to adapt to dynamic scenarios and uncertainties. Specifically, they allow agents to autonomously learn from their experiences, making them well-suited for tasks that require decentralized decision-making and adaptability to unforeseen situations [49].

In addition to RL, the incorporation of FL offers a decentralized approach, further advancing learning and adaptability in dynamic and distributed environments. The integration of FL with RL presents opportunities for drones to learn and update their policies in a distributed manner collaboratively, leveraging the collective intelligence of the swarm [50]. The advent of the FL offers a decentralized training paradigm, allowing agents to learn collaboratively while keeping their data localized.

The opportunity and challenge presented by IoT devices in FL are highlighted in [50], where edge computing and deep reinforcement learning are combined for traffic management in IoT. The evolving nature of FL applications, branching out from traditional use cases, is demonstrated by this work. Regarding the aggregation methods in FL, robust algorithms that provide guarantees against adversarial attacks are introduced, underscoring the security aspect of FL [51].

By integrating RL with FL, researchers envision a new era where flying agents learn and adapt in real-time, making consensus problems more manageable. Fusing these advanced learning techniques can revolutionize consensus mechanisms in flying multi-agent systems. The application of these integrated learning methods in multi-agent systems presents various challenges and opportunities, particularly in formation control and security aspects.

Formation control in multi-agent systems, particularly in cost-constrained environments, presents inherent challenges, including managing limited resources, communication constraints, and ensuring scalability and robustness. To address this, research has proposed a Distributed Deformable Configuration Control (DDCC) for multi-robot systems, allowing adaptive shape maintenance among environmental changes. This method leverages local sensing and communication, demonstrating robustness even on low-cost platforms [52]. In

contrast, the security dimension of multi-agent systems has been explored in the context of cyber threats.

A study presented an edge-based adaptive mechanism to achieve secure consensus in non-linear multi-agent systems, especially when communication links face threats. This methodology adaptively tunes communication link weights and has exhibited resilience against adversarial conditions, with its stability validated using Lyapunov stability theory [53]. Beyond these security and control aspects, multi-agent systems research also explores hierarchical structures, such as leader-follower dynamics, which are essential in certain applications like maritime operations.

Leader-follower consensus mechanisms play a pivotal role in tasks demanding hierarchical structures. Recent work in this domain has introduced finite-time control protocols aiming for rapid leader-follower consensus. Through a specific leader-follower model combined with Lyapunov function techniques, these methods assure stability and convergence within set time frames [54]. Maritime applications, especially concerning the flocking control of Unmanned Surface Vehicles (USVs), have introduced a path-guided model-free flocking control strategy. This approach, devoid of reliance on precise mathematical models, employs Concurrent Learning Extended State Observers (CLESOs) for accurate system state and dynamic estimations, ensuring robust control among maritime uncertainties [55]. In the realm of decision-making within these systems, innovative methodologies like entropy-based consensus are emerging, offering new ways to enhance cooperation and efficiency.

A novel methodology in artificial swarms emphasizes entropy-based local negotiation and preference updating for consensus decision-making. This approach, grounded in swarm intelligence, employs an entropy-based metric to gauge consensus levels among agents, informing their negotiation choices. The method has shown promise with faster convergence, higher success rates, and remarkable scalability, making it suitable for diverse applications [56]. Efficient decision-making in multi-agent systems can also be optimized through event-triggering mechanisms, which streamline communication and computation processes.

According to recent research, event-triggering mechanisms in multi-agent systems can

improve communication and computation. This approach operates with predetermined communication patterns, reducing messages to save resources while ensuring stability and convergence [57]. While optimizing communication in multi-agent systems, emerging methods like SNN offer groundbreaking approaches to robotic learning and control.

SNNs have gained popularity in robotics due to their ability to replicate the structure and functioning of networks. One crucial characteristic of SNNs is their capacity to encode and handle information through spikes, which has been demonstrated as a resource energy-saving approach [58, 59]. The versatility of SNNs is further expanded through their ability to learn and adapt, which is particularly beneficial in complex tasks like movement planning and nonlinear system control.

One of the remarkable features of SNNs is their ability to learn and adapt. Here, R-STDP plays a key role. R-STDP is a bio-inspired learning rule based on the relative timing of pre- and post-synaptic spikes. Several studies have elucidated the constraints on Hebbian and R-STDP learned weights in spiking neurons, revealing the underlying mechanisms that make this learning paradigm so effective [60]. Furthermore, research has demonstrated that networks trained with local rules, such as R-STDP, can exhibit continuous learning, showcasing their potential in lifelong learning scenarios [61]. To fully utilize the potential of SNNs, the development of hybrid models and the exploration of various training methodologies are crucial in enhancing their adaptability and efficiency.

In robotics, the application of SNNs has shown promise in various tasks, including movement planning within confined operational spaces. Such applications leverage the temporal dynamics of spiking neurons to achieve collision-free motion, demonstrating the capability of SNNs to handle complex spatial-temporal challenges [62]. Beyond movement planning, SNNs have also been employed for nonlinear systems control, providing a robust and adaptive control mechanism [63].

The evolution of SNN has seen the emergence of models that aim to blend the advantages of machine-inspired approaches. These hybrid systems have proven to enhance the adaptability of SNNs, making them more suitable for environments encountered in robotics [64].

The advancements in classification capabilities offered by integrate and fire models have also expanded the range of SNN applications in robotics [65]. Notably, Deep Spiking Q Networks, which are trained directly, have shown performance in tasks highlighting the potential of integrating deep learning techniques with SNNs to achieve human-level control [66]. Exploring direct and indirect training methodologies for SNNs is vital, particularly in applications such as autonomous vehicle control, where end-to-end learning is essential.

Direct and indirect training methodologies for SNNs have been extensively explored for achieving end-to-end control of vehicles in tasks like lane keeping [67]. Furthermore, incorporating meta neurons into SNNs has further improved their effectiveness in spatial learning tasks [68]. The adaptability of SNNs, enhanced through reinforcement and evolutionary learning, opens new avenues in robotics, particularly motor control and changing operational needs. The exploration of learning dynamics in neural networks, particularly in developing neural units that can learn rules and robot dynamics, represents a significant advancement in the capabilities of SNNs.

In line with research on learning dynamics, there is a growing interest in developing networks consisting of neural units. When combined together, these units possess the ability to learn both learning rules and spiking dynamics, thereby enhancing the capabilities exhibited by SNN [69].

The application of reinforcement and evolutionary learning to train SNNs for motor control has opened new avenues in the field of robotics. Such training methods use SNNs' adaptability for the changing needs of robots [70–73]. The integration of FL with SNNs presents the synergy between decentralized training strategies and advanced neural network architectures, offering collaborative and privacy-preserving learning opportunities.

Another frontier in the application of SNNs in robotics is the integration of FL. The FL has been combined with SNNs to achieve collaborative learning across multiple agents while harnessing the efficiency of SNNs in distributed settings [74].

In the exploration of integrating SNNs with FL, several challenges emerge. The transmission of SNN-specific parameters, such as spike timings, introduces considerable communica-

tion overhead in the FL setup [75]. Additionally, the aggregation of SNN models from diverse devices in a federated context is non-trivial, often leading to challenges in achieving effective global learning. The unique dynamics of spiking neurons, coupled with the distributed nature of FL, can also result in training instabilities. Lastly, the inherent complexity and potential size of SNNs raise concerns about memory requirements and the feasibility of model aggregation in distributed scenarios [76].

Chapter 3

Control and Navigation of the Leader Drone Over 5G Network

3.1 Introduction

In this chapter, we perform a comparative analysis to examine how linear and nonlinear controllers impact quadrotor performance when operated via cellular or 5G networks. This chapter is part of the Ericsson 5G Lab's project in autonomous vehicle technology, encompassing both simulation and implementation of Unmanned Aerial Vehicle (UAV) and Unmanned Ground Vehicle (UGV) navigating over a 5G network. The chapter is divided into two phases. The first phase focuses on studying the control mechanisms of autonomous vehicles, taking into account delay constraints. The second phase involves developing a practical testbed for evaluating the designed algorithms.

In this chapter, we consider a scenario where a single UAV (leader drone) is controlled over a 5G network by a user from a base station. This section outlines the challenges encountered and presents solution to address these challenges. Regarding the simulation aspect, we choose to run the controller at the Ground Control Station due to the following reasons:

- *Integrated control system*: The controller uses information provided by the delay esti-

mator and state estimator. All these parts should be run as an integrated system on the Ground Control Station to avoid the interconnection delay among the controller, state estimator, and delay estimator.

- *Low computation time*: Running complex control systems needs high computation capability that the servers can provide on the Ground Control Station.
- *Beamforming*: The beamforming technology needs the user's current position to find the most efficient data delivery route. In the studied control framework, the Ground Control Station provides the drone's position to the base station.
- *Path planning*: The controller executes the commands that are provided by a path planning algorithm. The proposed framework reduces the transmission delay between the path planning algorithm and the control system.

Fig. 3.1 shows the control framework. The controller sends control commands $U(t)$ to the quadrotor using the cellular networks. The quadrotor receives control commands and sends back position and orientation $S(t)$. The recorded delay history is used for time delay estimation ($\hat{\tau}_B$).

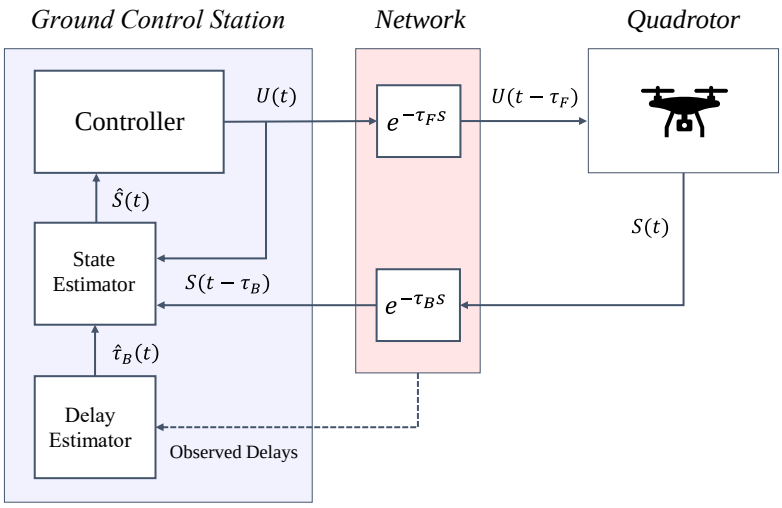

Figure 3.1: Control framework for quadrotor control using cellular networks

The state estimator estimates the quadrotors current position and orientation $\hat{S}(t)$ using delay information, control commands, and received data from the quadrotor.

3.2 Quadrotor dynamic model

In this section, the dynamic model of the quadrotor is presented. The quadrotor is a six Degree-of-Freedom (DOF) system controlled by four motors, as shown in Fig. 3.2. It is an underactuated system that makes the control process a challenging task.

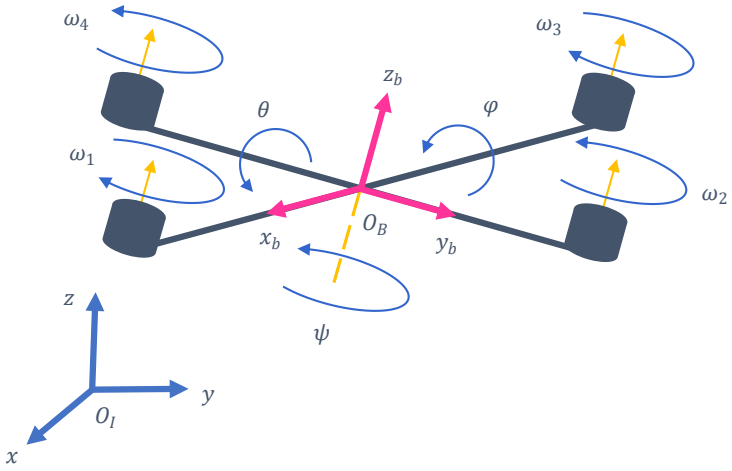

Figure 3.2: Quadrotor's dynamic model

An inertial $\{O_I, x, y, z\}$ and a body-fixed $\{O_B, x_b, y_b, z_b\}$ reference frame are used to define the rotational and transitional motions [77]. According to rigid body dynamics, the rotational motion can be represented as,

$$\ddot{\phi} = \frac{(I_y - I_z)}{I_x}\dot{\theta}\dot{\psi} + \frac{l}{I_x}U_2 \tag{3.1}$$

$$\ddot{\theta} = \frac{(I_z - I_x)}{I_y}\dot{\phi}\dot{\psi} + \frac{l}{I_y}U_3 \tag{3.2}$$

$$\ddot{\psi} = \frac{(I_x - I_y)}{I_z}\dot{\theta}\dot{\phi} + \frac{l}{I_z}U_4 \tag{3.3}$$

where, $\{\phi, \theta, \psi\}$ are the Euler angles (pitch, roll, and yaw), I_x, I_y, and I_z are the moment of inertia along x, y, and z axes, respectively. The U_is are the control inputs, and l is the distance between the quadrotor's geometric center and the rotors.

The transitional motion in the inertial frame is given by,

$$\ddot{x} = (\cos\phi \sin\theta \cos\psi + \sin\phi \sin\psi)\frac{U_1}{m} \tag{3.4}$$

$$\ddot{y} = (\cos\phi \sin\theta \sin\psi - \sin\phi \cos\psi)\frac{U_1}{m} \tag{3.5}$$

$$\ddot{z} = -g + (\cos\phi \cos\theta)\frac{U_1}{m} \tag{3.6}$$

where, m is the mass, and g is the acceleration due to gravity. The generated force by the motors is considered to be proportional to the square of the angular velocities of the rotors. Therefore, the control inputs can be represented as,

$$U_1 = b\left(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2\right) \tag{3.7}$$

$$U_2 = b\left(\omega_4^2 - \omega_2^2\right) \tag{3.8}$$

$$U_3 = b\left(\omega_3^2 - \omega_1^2\right) \tag{3.9}$$

$$U_4 = d\left(\omega_2^2 + \omega_4^2 - \omega_1^2 - \omega_3^2\right) \tag{3.10}$$

where, ω_i is the angular velocity of the ith rotor, and b and d are constants.

3.3 Linear PD Controller

The PID controller is a widely used linear controller and is defined as,

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{3.11}$$

where, $e(t)$ is the error between the desired state and the current state, and K_p, K_i, and K_d are PID gains. In this chapter, a PD controller is designed to control the rotational and transitional motions. The controller is designed based on the linearized model of the quadrotor.

To simplify the linearization of the system, we assume that the yaw angle is held constant at zero, and as such, the motion along the x-axis is controlled only by the pitch angle θ, and the motion along the y-axis is controlled only by the roll angle ϕ. We linearized the dynamics for our PD controller design by assuming that the roll and pitch angles are small, which is a reasonable assumption since quadrotors rarely rotate more than 30 degrees during flight. Also, the angular accelerations of roll and pitch are equal to the torque about the appropriate axis, as such, we write the linear dynamics as $\ddot{\phi} = T_x/Ixx$ and $\ddot{\theta} = T_y/Iyy$, where T_x and T_y are the torques about the x and y-axis.

There are two control loops: inner and outer loop. The outer loop controls the desired pitch and roll angles. Since the motions along x and y axes depend on the rotation along ϕ and θ, the outer loop uses the current position in x and y and desired positions to calculate the desired rotations along x and y axes as follows [78],

$$\theta_d = K_p\,(x - x_d) + K_d\,(\dot{x} - \dot{x}_d) \tag{3.12}$$

$$\phi_d = K_p\,(y - y_d) + K_d\,(\dot{y} - \dot{y}_d) \tag{3.13}$$

where, ϕ_d and θ_d are desired pitch and roll angles, and x_d and y_d are desired positions on x and y directions. The K_p and K_d control the pitch and roll angles. The inner loop controls

the quadrotor's rotation along the x and y axes.

According to the PD control law, the control signals can be represented as [79],

$$U_1 = mg + K_{zp}\left(z_d - z\right) + K_{zd}\left(\dot{z}_d - \dot{z}\right) \tag{3.14}$$

$$U_2 = K_{lp}\left(\phi_d - \phi\right) + K_{ld}\left(\dot{\phi}_d - \dot{\phi}\right) \tag{3.15}$$

$$U_3 = K_{mp}\left(\theta_d - \theta\right) + K_{md}\left(\dot{\theta}_d - \dot{\theta}\right) \tag{3.16}$$

$$U_4 = K_{np}\left(\psi_d - \psi\right) + K_{nd}\left(\dot{\psi}_d - \dot{\psi}\right) \tag{3.17}$$

where, K_{zp}, K_{zd}, K_{lp}, K_{ld}, K_{mp}, K_{md}, K_{np}, and K_{nd} are PD gains for the inner loop. The z_d is the desired position along the z-axis, and ψ_d is the desired yaw angle.

The linearized PD controller can be designed based on the second-order dynamic model, where the bandwidth and damping ratio of the controller are adjustable. Therefore, gains can be calculated based on the inner and outer loop bandwidth.

3.4 Backstepping Controller

In this section, a nonlinear backstepping control law is designed to be compared with the linear PD controller. As mentioned before, the backstepping controller breaks the control law into several subsystems [80]. There are three subsystems consisting of the altitude (z), heading (ψ), and horizontal subsystems. The general Lyapunov function for every subsystem based on error variables is defined as [24],

$$V_i = \frac{1}{2}\eta_i^2 \tag{3.18}$$

where, η_i is the error variable for the ith subsystem.

We define error variables for the z subsystem as $\eta_1 = z - z_d$, $\eta_2 = \dot{z} - c_1\eta_1 - \dot{z}_d$. Then, the control law for altitude control becomes:

$$U_1 = \frac{m}{\cos\phi\cos\theta}\left(g - \eta_1 - c_2\eta_2 - c_1\dot{z} + c_1\dot{z}_d + \ddot{z}_d\right) \tag{3.19}$$

where, c_1 and c_2 are constant values that control how fast the quadrotor responds to the input commands in the z-direction.

For the ϕ subsystem, the error variables are $\eta_9 = \phi - \phi_d$ and $\eta_{10} = \dot{\phi} - c_9\eta_9 - \dot{\phi}_d$. Therefore, the control law is given by:

$$U_2 = \frac{I_x}{l}\left(-\alpha\dot{\theta}\dot{\psi} - c_9\dot{\phi} + c_9\dot{\phi}_d - \eta_9 - c_{10}\eta_{10} + \ddot{\phi}_d\right) \tag{3.20}$$

where, $\alpha = \frac{I_y - I_z}{I_x}$, and c_9 and c_{10} are the constant values that control the response time of the ϕ subsystem [81].

For the θ subsystem, the error variables are $\eta_{11} = \theta - \theta_d$ and $\eta_{12} = \dot{\theta} - c_{11}\eta_{11} - \dot{\theta}_d$. Then, the control law for the θ subsystem becomes,

$$U_3 = \frac{I_y}{l}\left(-\beta\dot{\phi}\dot{\psi} - c_{11}\dot{\theta} + c_{11}\dot{\theta}_d - \eta_{11} - c_{12}\eta_{12} + \ddot{\theta}_d\right) \tag{3.21}$$

In the above equation, the $\beta = \frac{I_z - I_x}{I_y}$, and c_{11} and c_{12} are the constant values that control the response time of the θ subsystem.

The error variables for the ψ subsystem are $\eta_3 = \psi - \psi_d$ and $\eta_4 = \dot{\psi} - c_3\eta_3 - \dot{\psi}_d$. The control law is given by:

$$U_4 = I_z\left(-\gamma\dot{\phi}\dot{\theta} - c_3\dot{\psi} + c_3\dot{\psi}_d - \eta_3 - c_4\eta_4 + \ddot{\psi}_d\right) \tag{3.22}$$

where, $\gamma = \frac{I_x - Iy}{I_z}$, and c_3 and c_4 control the response time of the ψ subsystem.

Since the dynamic model is underactuated and the U_1 is incorporated in x and y subsystems, the virtual control signals in x and y directions can be calculated based on the mentioned Lyapunov function as follows,

$$u_y = \frac{m}{U_1}\left(-c_5\dot{y} + c_5\dot{y}_d + \ddot{y}_d - c_6\eta_6 - \eta_5\right) \tag{3.23}$$

$$u_x = \frac{m}{U_1}\left(-c_7\dot{x} + c_7\dot{x}_d + \ddot{x}_d - c_8\eta_8 - \eta_7\right) \tag{3.24}$$

where, c_5 to c_7 are constant coefficients, $\eta_5 = y - y_d$, $\eta_6 = \dot{y} + c_5\eta_5 - \dot{y}_d$, $\eta_7 = x - x_d$, and $\eta_8 = \dot{x} + c_7\eta_7 - \dot{x}_d$ are error values for position and velocity.

The desired roll and pitch angles considering yaw angle are $\phi_d = sin^{-1}[v(2)]$ and $\theta_d = sin^{-1}\left[\frac{v(1)}{cos\phi_d}\right]$ where,

$$v = \begin{bmatrix} cos\psi & sin\psi \\ sin\psi & -cos\psi \end{bmatrix}^{-1} \begin{bmatrix} u_x & u_y \end{bmatrix}^T \tag{3.25}$$

The v is designed to find the desired orientation along the x and y axes that make the quadrotor follow the desired position, velocity, and acceleration in the horizontal subsystem.

3.5 State Estimator

Practical networked control systems with feedback loops are prone to face anomalies due to communication imperfections such as time delay. The feedback loop provides information about the plant's current states, and the controller calculates the new commands based on the received information.

When the system experiences a time delay, the controller may become unstable. In order to design a suitable control system, one should use a state estimator that estimates the current states based on the received data from the plant. In this section, the state estimator is given as follows,

$$\dot{\hat{\phi}} = \bar{q}_\phi - k_1\left(\hat{\phi}(t - \hat{\tau}_B) - \phi(t - \tau_B)\right) \tag{3.26}$$

$$\dot{\bar{q}}_{\phi} = \left(\frac{I_y - I_z}{I_x}\right)\bar{q}_{\theta}\bar{q}_{\psi} + \frac{U_2\left(t\right)}{I_x} - k_2\left(\bar{q}_{\phi}(t-\hat{\tau}_B) - q_{\phi}(t-\tau_B)\right) \tag{3.27}$$

$$\dot{\hat{\theta}} = \bar{q}_{\theta} - k_3\left(\hat{\theta}(t-\hat{\tau}_B) - \theta(t-\tau_B)\right) \tag{3.28}$$

$$\dot{\bar{q}}_{\theta} = \left(\frac{I_z - I_x}{I_y}\right)\bar{q}_{\phi}\bar{q}_{\psi} + \frac{U_3\left(t\right)}{I_y} - k_4\left(\bar{q}_{\theta}(t-\hat{\tau}_B) - q_{\theta}(t-\tau_B)\right) \tag{3.29}$$

$$\dot{\hat{\psi}} = \bar{q}_{\psi} - k_5\left(\hat{\psi}(t-\hat{\tau}_B) - \psi(t-\tau_B)\right) \tag{3.30}$$

$$\dot{\bar{q}}_{\psi} = \left(\frac{I_x - I_y}{I_z}\right)\bar{q}_{\phi}\bar{q}_{\theta} + \frac{U_4\left(t\right)}{I_z} - k_6\left(\bar{q}_{\psi}(t-\hat{\tau}_B) - q_{\psi}(t-\tau_B)\right) \tag{3.31}$$

$$\dot{\hat{x}} = \bar{q}_x - k_7\left(\hat{x}\left(t-\hat{\tau}_B\right) - x\left(t-\tau_B\right)\right) \tag{3.32}$$

$$\dot{\bar{q}}_x = \left(\cos\hat{\phi}\sin\hat{\theta}\cos\hat{\psi} + \sin\hat{\phi}\sin\hat{\psi}\right)\frac{U_1(t)}{m} - k_8\left(\bar{q}_x\left(t-\hat{\tau}_B\right) - q_x\left(t-\tau_B\right)\right) \tag{3.33}$$

$$\dot{\hat{y}} = \bar{q}_y - k_9\left(\hat{y}\left(t-\hat{\tau}_B\right) - y\left(t-\tau_B\right)\right) \tag{3.34}$$

$$\dot{\bar{q}}_y = \left(cos\hat{\phi}sin\hat{\theta}sin\hat{\psi} - sin\hat{\phi}cos\hat{\psi}\right)\frac{U_1\left(t\right)}{m} - k_{10}\left(\bar{q}_y\left(t-\hat{\tau}_B\right) - q_y\left(t-\tau_B\right)\right) \tag{3.35}$$

$$\dot{\hat{z}} = \bar{q}_z - k_{11}\left(\hat{z}\left(t-\hat{\tau}_B\right) - z\left(t-\tau_B\right)\right) \tag{3.36}$$

$$\dot{\bar{q}}_z = -g + \left(cos\hat{\phi}\ cos\hat{\theta}\right)\frac{U_1(t)}{m} - k_{12}\left(\bar{q}_z\left(t-\hat{\tau}_B\right) - q_z\left(t-\tau_B\right)\right) \tag{3.37}$$

According to (3.26) to (3.37), the state estimation is based on the quadrotor's dynamic model and the control signal. Therefore, we need to run the controller on the GCS to have access to control signals for state estimation. As mentioned before, the state estimation results are necessary information for beamforming. The Due to space restrictions, details are left to the reader [24]. In (3.26) to (3.37), the τ_B and $\hat{\tau}_B$ are the true and estimated backward delays (the backward and forward delays are assumed to be equal). The k_is are the estimator gains, and $\hat{\phi}$, $\hat{\theta}$, $\hat{\psi}$, $\hat{x}$, $\hat{y}$, and $\hat{z}$ are the estimated orientations and positions.

3.6 Delay Estimator

Network load, node competition, and congestion affect the communication quality. Therefore, considering the network delay as a constant parameter is not a proper assumption. Whenever a packet is sent over the network, the network state changes. Since the delay distribution follows the Markov model [82], Markov-based models are suitable for delay estimation.

In this chapter, the delay is assumed to be a discrete-time, discrete-space stochastic process. In order to simulate the network delays during simulation, a fixed predefined transition matrix can be used as follows,

$$p_{ij}(n) = P(\tau_{n+1} = j | \tau_n = i) \tag{3.38}$$

where, τ_n and τ_{n+1} are current and next time delays, respectively, and the i and j are indices corresponding with the time delays. The delays estimator algorithm aims to find the true transition probabilities.

The delays are generated based on the fixed predefined transition matrix p. Delay values are assumed to be known ($D = \tau_1, \tau_2, ..., \tau_\lambda$), but we do not know which one will happen in the current time step. Therefore, the challenge is to find the transition probabilities matrix

Algorithm 1 Time delay estimation using Markov model

input $D_{1\times\lambda}$, delay values
input $M_{\lambda\times\lambda}$, auxiliary matrix
input $\hat{p}_{\lambda\times\lambda}$, estimated probability matrix
for $n = 1$ to *simulation time* **do**
    read $\tau(n-1)$
    $i \leftarrow$ index of $\tau(n-1)$ in D
    $j \leftarrow$ index of $\tau(n-2)$ in D
    $M[j,i] \leftarrow M[j,i] + 1$
    for $r = 1$ to λ **do**
        $\hat{p}[r, 1:\lambda] \leftarrow$ Normalize r^{th} row of the M
    end for
    $ind \leftarrow$ find the index of $\max(\hat{p}[i, 1:\lambda])$
    $\hat{\tau}[n] \leftarrow D[1, ind]$
    $DEE(n-1) \leftarrow \sqrt{\frac{\sum_{i=0}^{n-1}(\tau(n-1)-\hat{\tau}(n-1))^2}{n-1}}$
end for

p. The delay estimator gets the previous delay $\tau(n-1)$ as the initial value and calculates the transition probabilities matrix using the observed delays' history. Finally, the Delay Estimation Error (DEE) is calculated based on the delay history.

3.7 Results

The performance of the controllers was evaluated using a numerical simulation. The simulation was done in MATLAB R2021 and the total simulation time was 10 seconds with 0.001 second sampling time. The initial condition for orientation, angular velocities, position, and velocities were taken to be $\mathbf{0}_{12\times1}$. The desired positions in the x, y, and z directions were 10, 20, and 30 m.

The PD controller was designed using the linearized model of quadrotor dynamics. The gains were calculated using the second-order system model, and the bandwidth for inner and outer loops were 70 rad/s and 5 rad/s, respectively. High bandwidths for controllers can be used because of the small delay values in the 5G network. The c_is in the backstepping controller were tuned so that the response time of the controllers are approximately equivalent. The control gains are shown in Table 3.1.

Table 3.1: Gains used in PD and backstepping controllers

	Coefficient	**Value**	**Coefficient**	**Value**
PD	K_p	2.5	K_d	1
	K_{lp}	20.09	K_{ld}	0.574
	K_{mp}	20.09	K_{md}	0.574
	K_{np}	43.12	K_{nd}	1.232
	K_{zp}	2293.2	K_{zd}	65.52
Backstepping	c_1	20	c_2	25
	c_3	3	c_4	6
	c_5	3	c_6	6
	c_7	3	c_8	6
	c_9	15	c_{10}	30
	c_{11}	15	c_{12}	30

Table 3.2: Quadrotor specifications and state estimator constants

Parameter	**Value**	**Unit**
m	0.468	kg
I_x	0.0041	$kg.m^2$
I_y	0.0041	$kg.m^2$
I_z	0.0088	$kg.m^2$
l	0.225	m
k_i {i even}	4	-
k_i {i odd}	8	-

The nonlinear quadrotor model was used for the simulation. The dynamic parameters and state estimator coefficients are shown in Table 3.2.

In order to simulate the delay in the 5G network, a model based on the Markov chain was used to generate delays using the transition matrix. The transition probabilities were set randomly, and the delay values were changed to evaluate the effect of the delay changes on controllers. There were four delays in the experiment $D = \{6, 7, 9, 10\}$ ms with random probabilities. The delay estimator calculated the transition matrix and estimated the time delay based on the maximum probability.

Exact knowledge of delay In this case, the delay values and their probabilities were known. Fig. 3.3 shows the state estimator error in estimating the current states in x, y, and z directions. After three seconds, the state estimator has converged to zero.

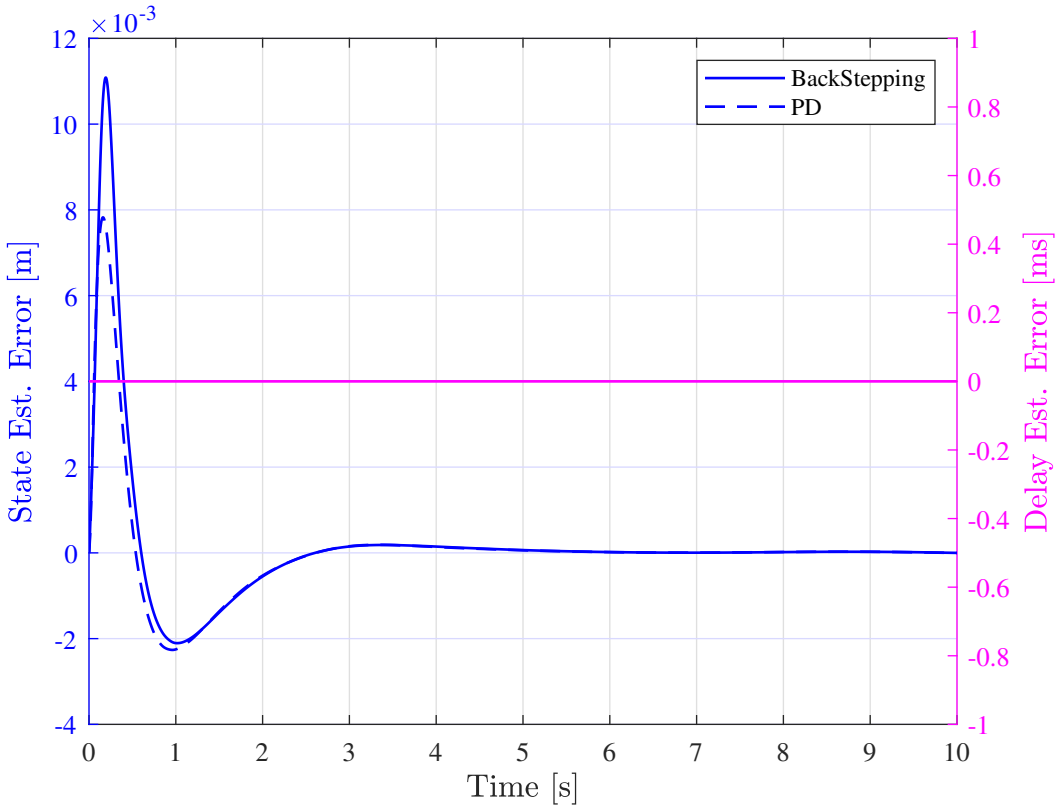

Figure 3.3: State estimation error with perfect delay estimator

Fig. 3.4 shows the control signals calculated on the Ground Control Station according to the estimated quadrotor states. The control signal for both controllers are similar, and as it is shown in Fig. 3.3, the control framework that works based on the PD controller estimates the states similar to the state estimator that works based on the backstepping.

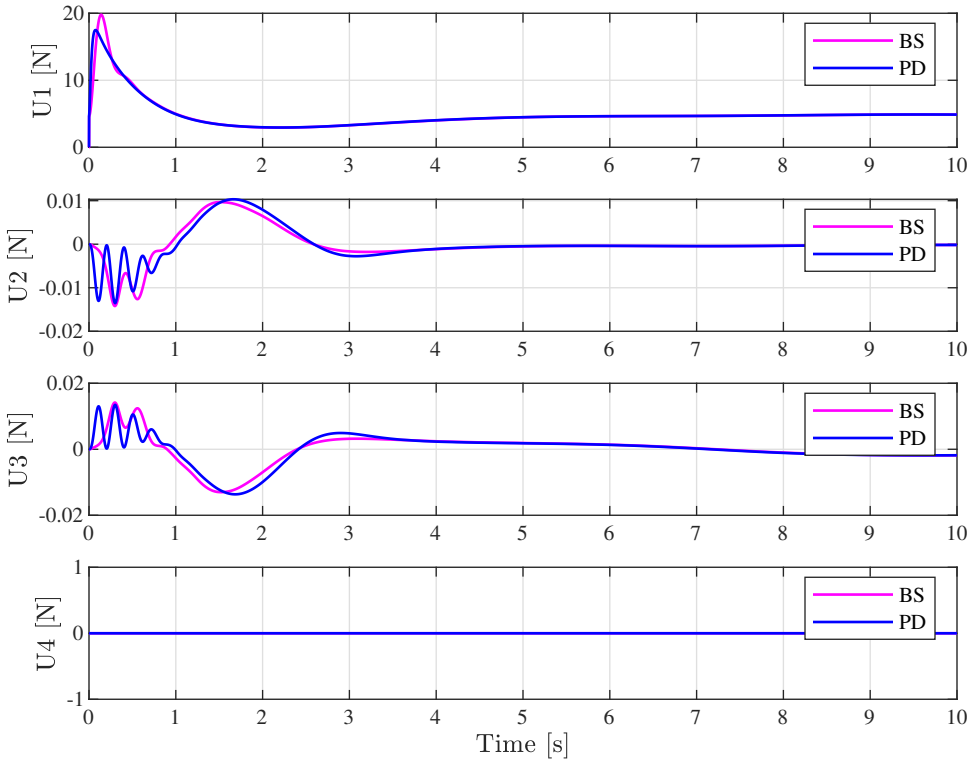

Figure 3.4: Control signals with perfect delay estimator

Delay estimation based on the Markov model In this case, delays were changed based on the predefined transition matrix. According to Fig. 3.5, the estimation error for the PD and backstepping controllers did not change significantly due to small delay values. The true and estimated transition probability matrices are shown in Table 3.3.

Table 3.3: True and estimated transition probabilities

		6 ms	7 ms	9 ms	10 ms
6 ms	True	0.7736	0.0250	0.1643	0.0372
	Estimate	0.7631	0.0259	0.1705	0.0405
7 ms	True	0.0224	0.1972	0.0723	0.7080
	Estimate	0.0349	0.2170	0.0574	0.6908
9 ms	True	0.0360	0.1797	0.0310	0.7533
	Estimate	0.0333	0.1849	0.0258	0.7559
10 ms	True	0.2097	0.0075	0.0305	0.7523
	Estimate	0.2174	0.0072	0.0357	0.7397

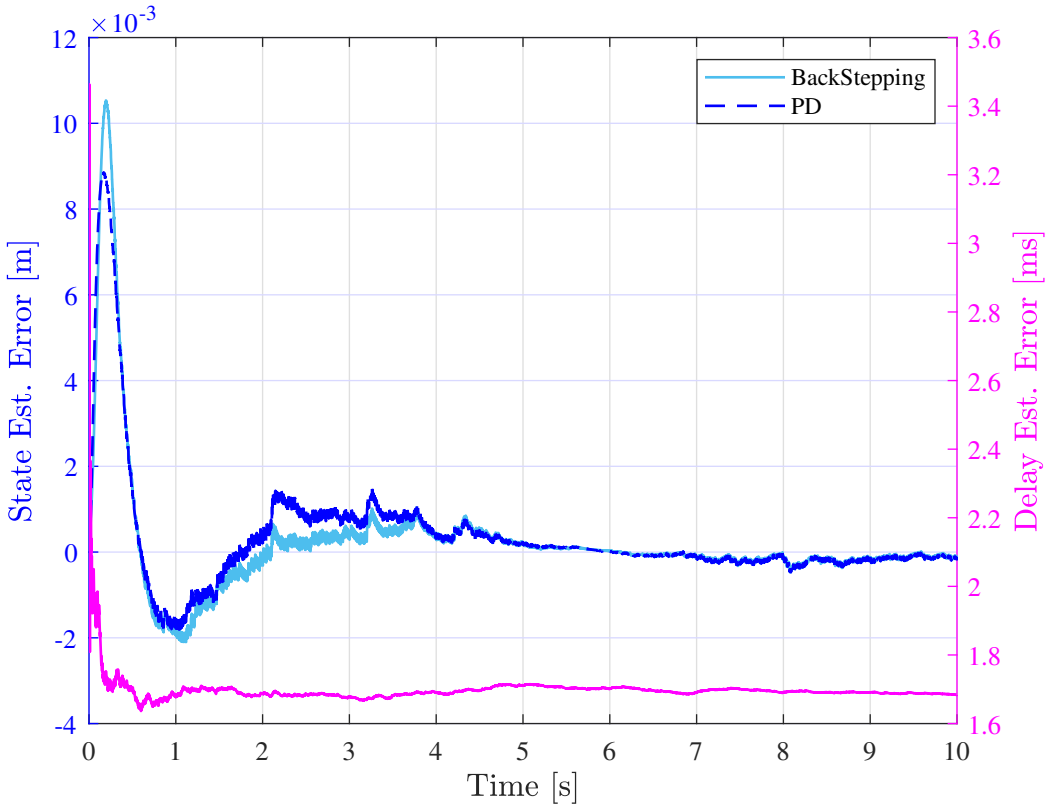

Figure 3.5: State and delay estimation error for $D = (6, 7, 9, 10)$ ms

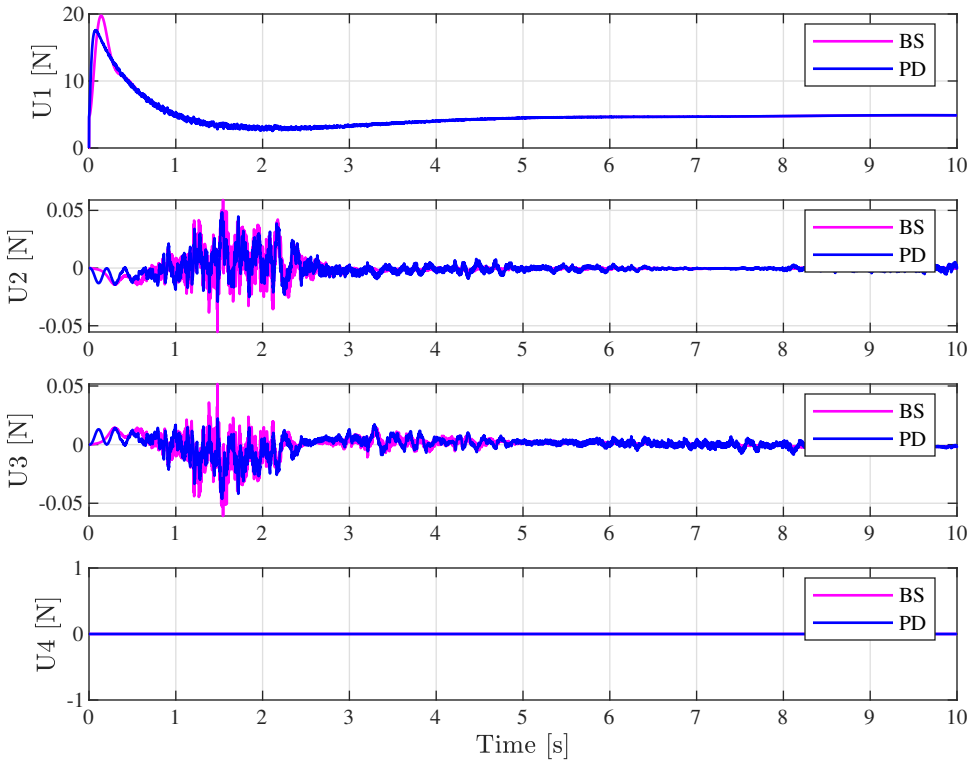

Figure 3.6: Control signals for $D = (6, 7, 9, 10)\ ms$

As illustrated in Fig. 3.6, even very small unknown delays affect controller performance. Because of the time delay in the system, tiny oscillations happen in the quadrotor's orientation. For example, since the bandwidth of the PD controller is relatively high (large gains in the control signal), small feedback errors are multiplied by the large gains and generate oscillating control signals.

In order to study the effect of the large changes in delay on the system, the time delays were changed to $D = \{1, 3, 13, 15\}\ ms$. The simulation was done with the true transition probabilities presented in Table 3.3. Fig. 3.7 shows how a slight change in the delay variation increases the estimation error. It also shows that we cannot use the mean value for the delay because large variations in delay negatively affect the response. It takes longer for the state estimator to estimate the current position.

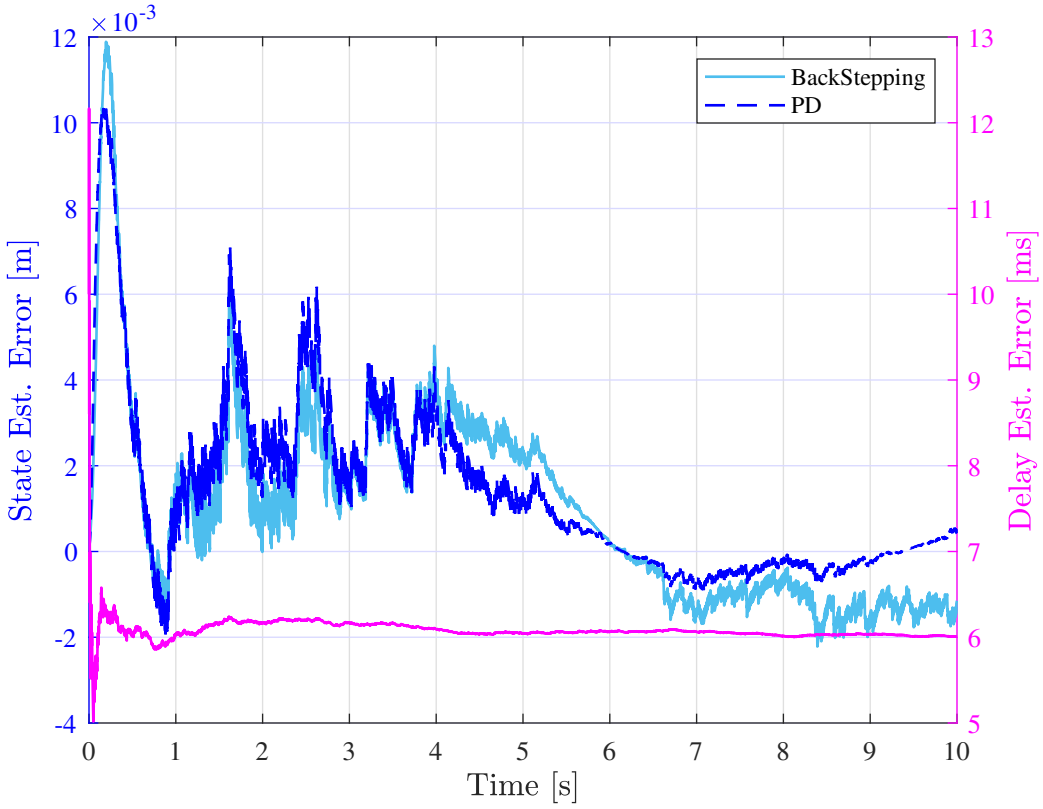

Figure 3.7: State and delay estimation error for $D = \{1, 3, 13, 15\}$ ms

According to Fig. 3.8, large variations in state estimation error affect the backstepping controller so that it generates control signals with higher frequency and amplitude than the PD controller. It should be noted that although one way to solve this problem is to reduce the backstepping controller's coefficients, lower values for coefficients increase the response time and path RMSE.

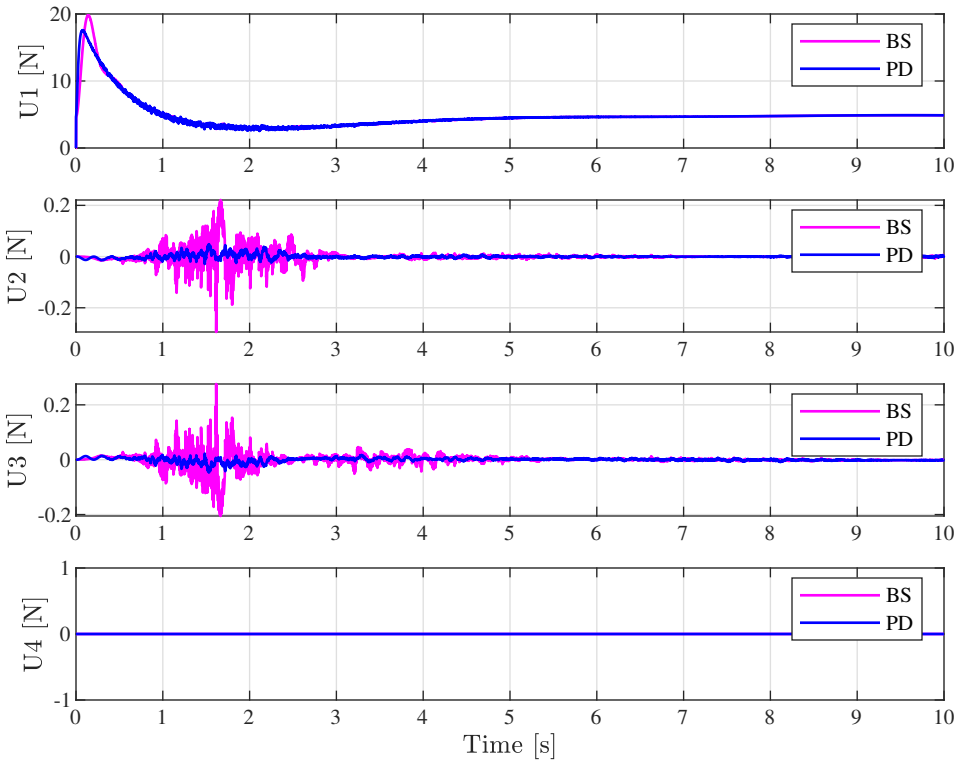

Figure 3.8: Control signals for $D = \{1, 3, 13, 15\}$ ms

Change in delay values and distributions While the cellular networks provide higher communication bandwidth with lower time delays, the variation in network load in crowded areas like urban environments changes the network properties. During the simulation, the time delays (D) and their distributions p were changed to simulate the changes in network states. The simulation was done with delays between $1 - 10$ ms, and the time delay values and their transition probabilities were changed six times.

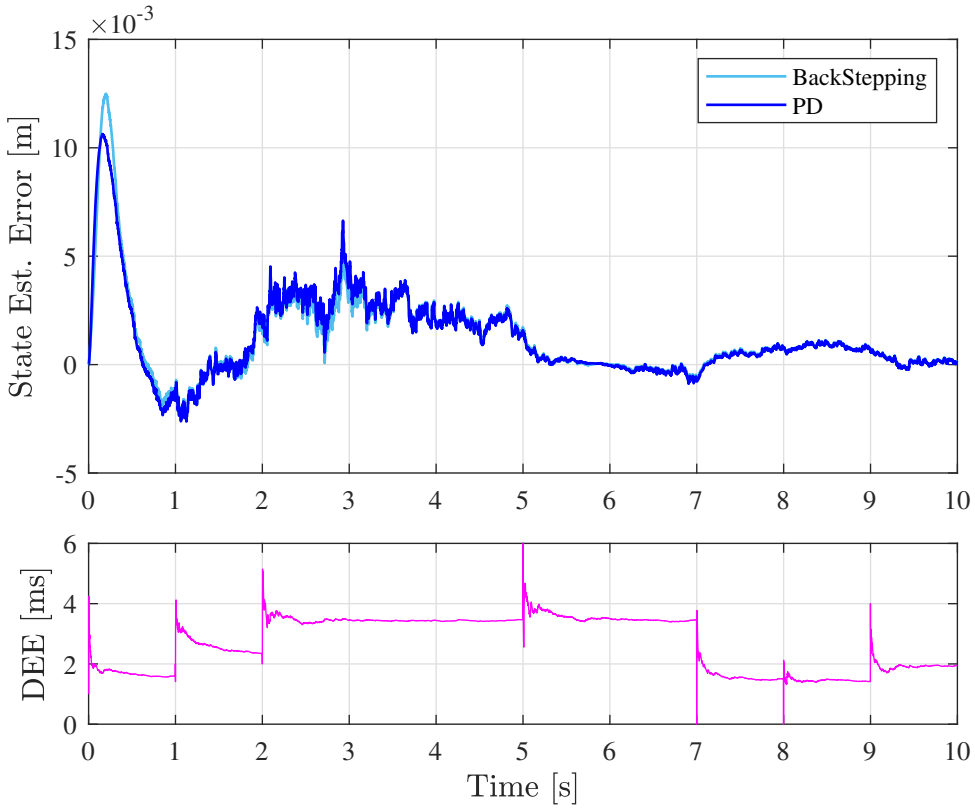

Figure 3.9: State and delay estimation error when delays and their distribution changed.

Fig. 3.9 shows the delay estimation error during the simulation. Large variations in delay estimation shows the points where the delay values and the transition probabilities are changed. According to the figure, change in delay values and distributions has affected the delay estimation. However, small delay changes between 1 to 10 ms did not decrease the state estimator's performance. It should be noted that after every change in delay, the DEE was reset, and error was calculated for new time delays.

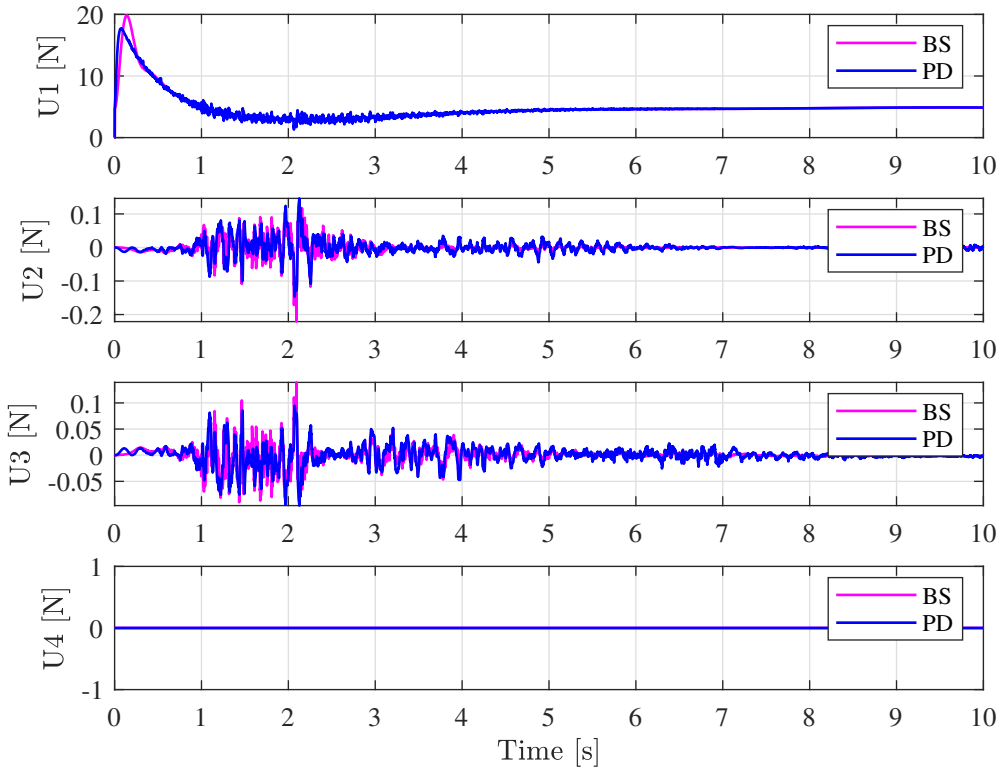

Figure 3.10: Control signals when delays and delay distributions changed

As shown in Fig. 3.10, errors in estimating time delay caused high-frequency oscillations in the control signal. Although the maximum delay value is not changed, changing delay values between 1 to 10 ms negatively affected the control signal.

According to the results, PD and backstepping controllers have similar performance in quadrotor control. However, in the case of the large variation in time delay, the PD controller generates better control signals. The design process of the backstepping controller is more complicated than the PD controller.

In this chapter, the PD gains and backstepping coefficients were set so that the trajectory tracking performance was almost the same. RMSE of the path in x, y, and z directions were 1.046, 1.138, and 0.131 m for the PD controller, and 1.224, 1.329, and 0.19 m for the backstepping controller. In this case, the forward delay is the major reason for high RMSE

and makes quadrotor follow the path with delay.

In the linear PD controller, the gains can be tuned using inner and outer loop bandwidths. Also, using the theoretical stability analysis based on the linearized model, one can find an equation between time delay and controller's bandwidth. This issue helps the control framework adjust the bandwidth based on the observed time delay.

The bandwidth adaptation in the backstepping controller is a complex task since 12 parameters should be tuned. On the other hand, developing an equation between coefficients and time delay is much more complex than the linear PD controller.

Cellular networks are state-of-the-art communication tools for precise swarm control of drones. The presented control framework can control multiple drones from the GCS while the state estimator provides precise information about each drone's current position and orientation (fully observable system) for real-time centralized multi-agent-based collision avoidance, cooperation, and path planning tasks.

3.8 Experimental setup

3.9 Conclusion

In this chapter, a comparative study was done to evaluate the performance of the linear PD and nonlinear backstepping controller in controlling the quadrotor through 5G or other cellular networks. Considering the channel behavior in producing the time delays, the delay estimator estimated the time delays using the Markov model. The estimated time delays were then used to estimate the quadrotor's correct position. The state estimator received the current control signal from the controller. It estimated the correct position using time delay estimation and the measured quadrotor position.

The controllers' performance in three different scenarios was evaluated. Both controllers generated a smooth control signal when the delay was known. There were oscillations in the control signal when the delay was unknown, but the position of the quadrotor was estimated with a very low error. Results showed that increasing the change in delay negatively affects

both controllers. Also, when delay values and their distributions changed, the oscillations in control signals increased.

According to the results, there is not a significant difference in controllers' performance. However, large variations in time delay are a considerable problem in the backstepping controller.

Chapter 4

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

4.1 Introduction

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

Expanding on what we learned, this chapter looks at two main things: how well SNNs handle noise, which is important because drones operate in real-world conditions with lots of disturbances, and how they can handle complex scenarios like optimal decision-making. This detailed study shows us the potential of SNNs in managing drones' complex behaviors even when there are outside disturbances and their capability to separate the reward function for the neural network's different parts.

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

problem [83].

In the ATD, a defender tries to protect a target, while a superior invader with higher velocity than the defender and target tries to reach the target and escape the defender at the same time. In defense strategy, they aid in analyzing movements and determining optimal strategies for aircraft. Law enforcement benefits from these games by minimizing escape possibilities. They are a fundamental component of game theory, providing insights into strategic decision-making across disciplines like economics and biology. Additionally, these games find applications in Cybersecurity for modeling attacker-defender interactions. [84].

There are three distinct outcomes to the ATD, characterized by two or three termination sets [85]. The first outcome happens when the invader reaches the target while the defender is far from it. The second outcome happens when the defender reaches the target while the invader's distance from the target is larger than the defender's distance. Finally, the third outcome is realized when the invader and defender reach the target at the same time.

4.2 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 [85]. 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 4.1 shows LOS angles used as the input for the SNNs [86].

The R-STDP algorithm is used to train two separate SNNs simultaneously that control the invader and defender. The target in this chapter moves in the environment, and the SNNs receive the LOS angles (e.g., the defender receives the LOS angle to both the invader

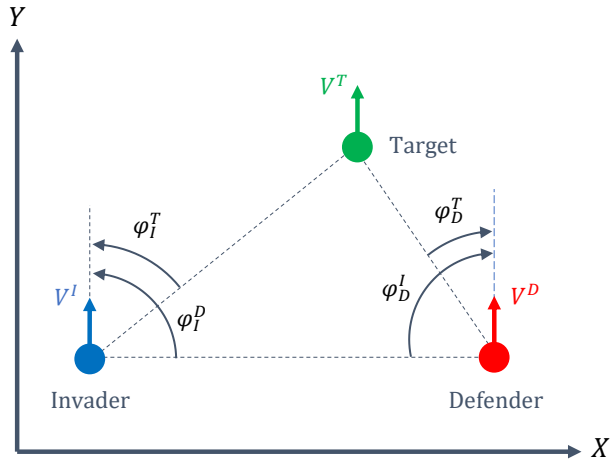

Figure 4.1: Active target defense game with three agents (LOS angles are shown for both agents).

and the target) and calculate the steering angle for the agent (Figure 4.2). In this figure, the ϕ_I^T represents the LOS angle to the target relative to the invader.

4.3 Learning using R-STDP

R-STDP is a biological learning algorithm that is believed to underlie certain learning mechanisms in the brain [87]. The R-STDP algorithm is based on the idea that if a pre-synaptic neuron fires just before a post-synaptic neuron, the strength of the synapse between the two neurons should be increased. Conversely, if the post-synaptic neuron fires just before the pre-synaptic neuron, the strength of the synapse should be decreased.

In the SNN, the pre-synaptic neurons are input neurons, and post-synaptic neurons are output neurons. The synaptic plasticity is referred to as an eligibility trace (C), which is calculated based on the following equation [88],

$$\dot{C}^{i-j} = -C^{i-j}/\tau_C + STDP^{i-j}(\tau)\delta(t - t_{pre/post}) \tag{4.1}$$

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

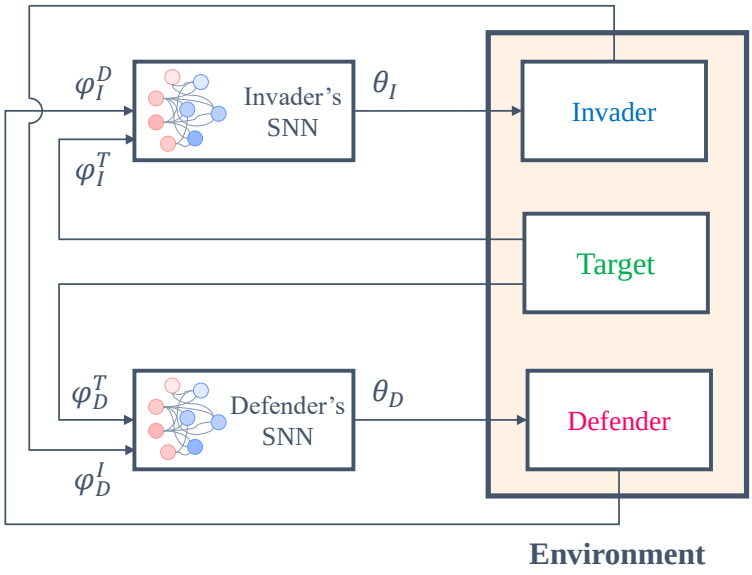

Figure 4.2: Two SNNs simultaneously play and control the invader and defender. Each agent uses the LOS angle and relative velocities for training.

constant for the synaptic plasticity, δ is the Dirac function, $t_{pre/post}$ is the firing time of the pre or post-synaptic neuron, and $STDP(\tau)$ is a function of the firing time of the input and output neurons as follow,

$$STDP^{i-j}(\tau) = \begin{cases} A_+ exp\left(\frac{-\tau}{\tau_s}\right) & \text{if } \tau > 0 \\ A_- exp\left(\frac{\tau}{\tau_s}\right) & \text{if } \tau < 0 \end{cases} \tag{4.2}$$

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

The synaptic weights are changed according to the following equation,

$$\dot{W}^{i-j}(t) = C^{i-j}(t)R(t) \tag{4.3}$$

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 [87],

$$\overrightarrow{W}^n(t) = \overrightarrow{W}^n(t-1)\left(\frac{I^{max}}{\Sigma_{\iota=1}^{N} W^{\iota}(t)}\right) \tag{4.4}$$

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

The reward for the invader and defender is defined based on the projection of the velocities along the LOS direction. Figure 4.3 shows the projected velocities for the invader and defender, where both agents measure the relative velocities from each other. It means that the invader receives a negative reward when it moves toward the defender and receives a positive reward when it moves toward the target. The defender receives a positive reward when it moves toward the target and invader. Since the velocities are constant, the reward value depends only on the headings that cause the change in relative velocities. Therefore, the LOS toward the other agents determines the reward value.

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

$$R_I^T(t) = \eta_I^T\left(V^I + V^T\right) cos\left(\phi_I^T\right) \tag{4.5}$$

and

$$R_I^D(t) = \eta_I^D\left(V^D - V^I\right) cos\left(\phi_I^D\right) \tag{4.6}$$

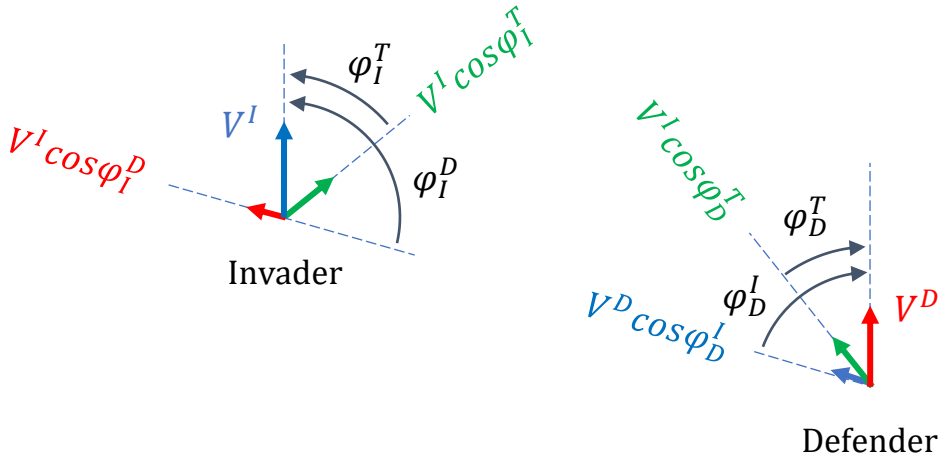

Figure 4.3: Relative velocities and LOS angles used in the reward function

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

$$R_D^T(t) = \eta_D^T \left(V^D + V^T\right) cos\left(\phi_D^T\right) \tag{4.7}$$

and

$$R_D^I(t) = \eta_D^I \left(V^D - V^I\right) cos\left(\phi_D^I\right) \tag{4.8}$$

In (4.5)-(4.8), η_I^T, η_I^D, η_D^T, and η_D^I are constant coefficients. Adjusting these values changes the agent's attention to other agents. For example, in the invader case, increasing the η_I^T and decreasing the η_I^D increases the effect of the target on the output and reduces the defender's effect. The invader then places more importance on getting to the target than evading the defender.

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

$$\mathbf{W}_I^T(t) = \mathbf{W}_I^T(t-1) + \mathbf{C}_I^T(t) R_I^T(t) \tag{4.9}$$

and

$$\mathbf{W}_I^D(t) = \mathbf{W}_I^D(t-1) + \mathbf{C}_I^D(t) R_I^D(t) \tag{4.10}$$

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.

4.4 Network structure and encoding method

Figure 4.4 shows the defender's network structure and encoding process. The invader has the same network with different inputs. The network receives the LOS angles and converts the inputs to Fuzzy Membership Values (FMV) using Gaussian Receptive Fields (GRF) [89]. Since the fuzzy membership values are used just for encoding data into the SNN, the type of the membership function does not affect the computation complexity.

There are q input neurons, and a membership function is assigned to each neuron. Therefore, there are q membership functions. The acquired fuzzy membership values are real numbers between 0 and 1 and should be converted to the input currents. This can be done using a linear function and the minimum and maximum inputs.

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

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

or

$$I_\sigma = \frac{\tau_m \left(V_{th} - V_{res}\right)}{\Delta t R_m} FMV_\sigma + \frac{V_{th} - E_l}{R_m} \tag{4.11}$$

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

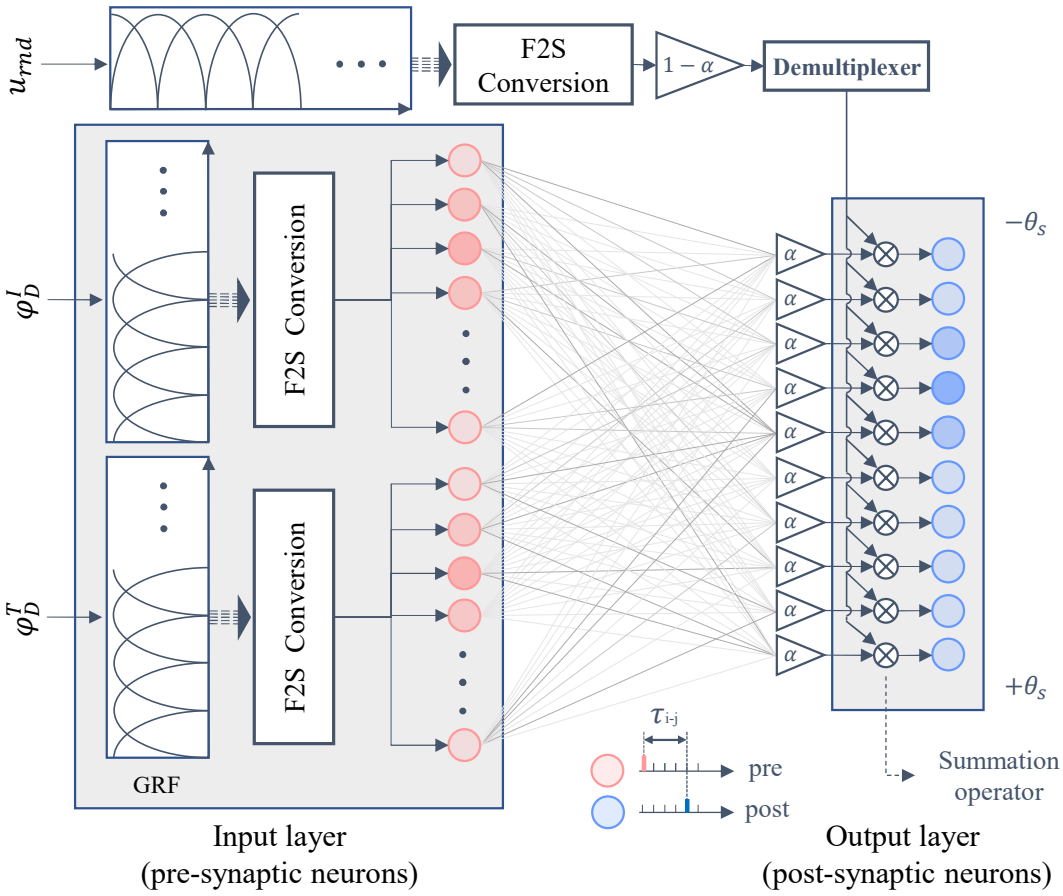

Figure 4.4: Network structure and encoding process for the input layer (Defender). Each neuron is associated with a membership function in GRF. The GRF encodes an input State (S^t) at each time step. There is both a training phase when ($\alpha = 0$) and an operating phase when ($\alpha = 1$).

value in GRF.

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

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

$$I_{out} = \alpha I_{syn} + (1 - \alpha)\, I_{rnd} \tag{4.12}$$

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

During training, the input State (S^t) is encoded into the network, and a random steering angle (u_{rnd}) is encoded as a random action using the F2S process into the output layer. These two encoding currents for the input and output layer make input and output neurons fire independently. The R-STDP adapts the weights for the fired neurons. Since α is 0 during the training, the I_{syn} does not affect the SNN's output (Equation 4.12).

In the training process, the output neurons and input neurons are excited separately. The input neurons are fired based on the agent's current state, whereas the agent's steering angle is randomly assigned based on the random input to the output neurons, as shown in Figure 4.8. The training algorithm then evaluates the reward for that given random steering angle. If the reward is positive, then the weight associated with the input neurons to output neurons that fired for that State is strengthened, and if the reward is negative, then the

Algorithm 2 Weight training algorithm

for $t = 0 : \Delta t : t_{final}$ **do**
  input ϕ_I^T and ϕ_I^D
  $\zeta \leftarrow$ number of input neurons
  $\kappa \leftarrow$ number of output neurons
  for i = $0{:}2\pi/\zeta{:}2\pi$ **do**
    $Z_T(i,1) = exp\left(-0.5(\phi_I^T - i)/\sigma\right)$
    $Z_D(i,1) = exp\left(-0.5(\phi_I^D - i)/\sigma\right)$
  end for
  $I^T = (I^{max} - I^{min})\, Z_T + I^{min}$
  $I^D = (I^{max} - I^{min})\, Z_D + I^{min}$
  Generate a uniform random number for exploration $u_{rnd} \in [-\theta_s, \theta_s]$
  $j = 1$
  for i = $-\theta_s{:}2\theta_s/\kappa{:}\theta_s$ **do**
    $Z_{rnd}(j,1) = exp\left(-0.5(u_{rnd} - i)/\sigma\right)$
    $j \leftarrow j + 1$
  end for
  $I_{rnd} = (I^{max} - I^{min})\, Z_{rnd} + I^{min}$
  for j = 1 to κ **do**
    $I_{syn}(j,1) = \sum_{i=1}^{\zeta} W^{i-j}\delta(t - t_i)$ ▷ t_i is the firing time of the i^{th} input neuron
  end for
  $I_{in} = \begin{bmatrix} I^T \\ I^D \end{bmatrix}$ Input current
  $I_{out} = \alpha I_{syn} + (1 - \alpha) I_{rnd}$ ($\alpha = 0$ in training phase)
  $\mathbf{V_m}(t + \Delta t) = (1 - \frac{\Delta t}{\tau_m})\mathbf{V_m}(t) + \frac{\Delta t}{\tau_m}\left(\mathbf{E_l} + R_m \begin{bmatrix} I_{in} \\ I_{out} \end{bmatrix}\right)$ {$\mathbf{V_m}$ and $\mathbf{E_l}$ are $[\zeta + \kappa] \times 1$}
  Find fired neurons in input and output layer
  Calculate the $\boldsymbol{\tau}$ matrix that shows the difference in firing time between the fired input neurons and fired output neurons (Figure 4.4)
  Calculate $\mathbf{R - STDP}$ for all connection using (4.2)
  Calculate $\mathbf{C}$ matrix using (4.1) for all the connections
  Calculate $\mathbf{W}$ matrix using (4.3) for all the connections considering reward from (4.5) to (4.8)
  for i = 1 to κ **do**
    if sum of the input weights to ith neuron $\geq I^{max}$ **then**
      Normalize input weights of ith neuron using (4.4)
    end if
  end for
  Set voltage of the fired neurons to reset voltage (V_{res})
end for

Table 4.1: Parameter values for LIF neuron model [90]

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

weight for the input to output neurons in (4.2) is weakened. Future research will include an inhibitory effect where the weights can become negative.

After training, the α changes to 1 and eliminates the effect of random output, and the SNN's output is calculated based on the synaptic currents. Algorithm 2 shows the training process.

4.5 Results

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

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

Table 4.2: Parameter values for R-STDP

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

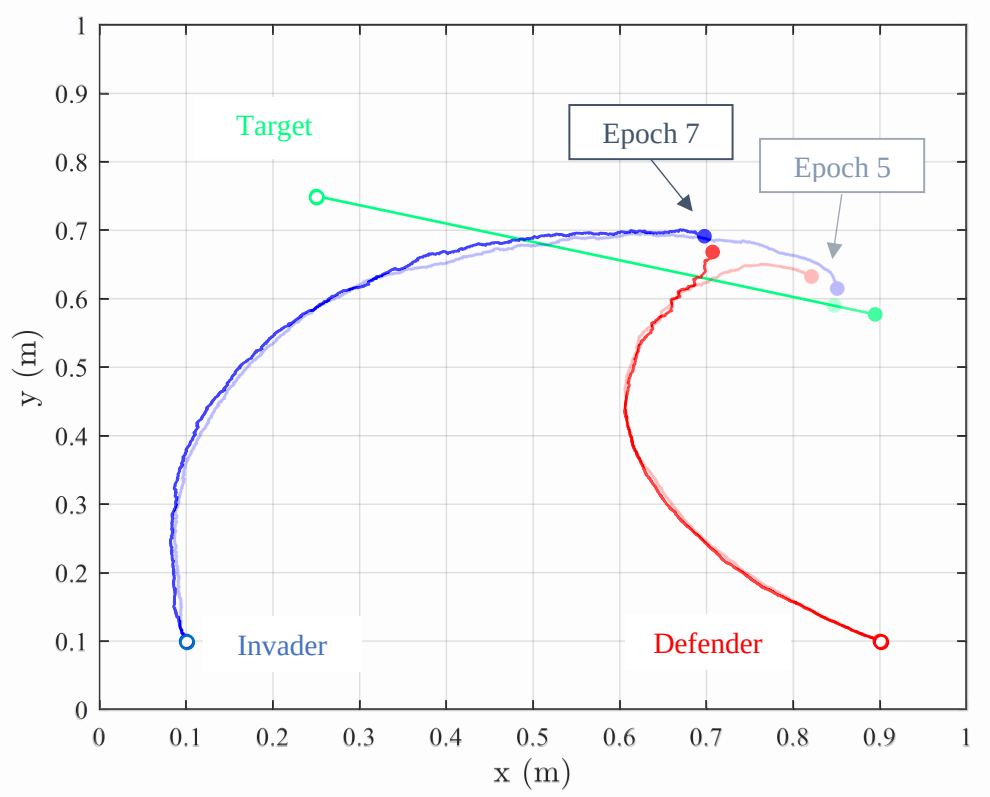

Figure 4.5: SNN's performance during training. Hollow circles show the initial positions.

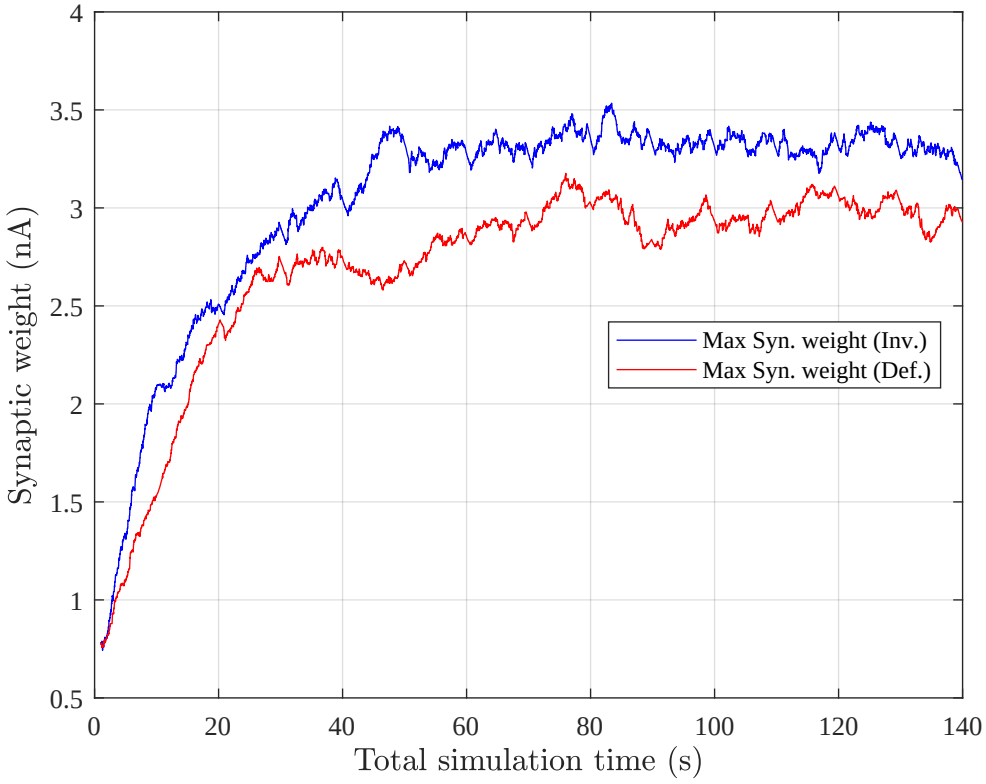

Figure 4.6: Changes in synaptic weights during training. Only Maximum synaptic weights for both agents are shown.

layer has 20 Gaussian membership functions.

The output of the activation of each membership function is the input to a neuron associated with that specific membership function. There are 10 input neurons for the 10 membership functions related to each input. Furthermore, no more than 2 membership functions fire for any given input. Therefore, at most, only two neurons are excited and generate an impulse sequence for a given input.

The target's velocity is 0.15 m/s. The invader's velocity is 0.3 m/s. The γ is 0.75, so the defender's velocity is 0.225 m/s. The defender's capture radius (ρ) is set to 0.025 m. The maximum simulation time for each epoch is 10 seconds. The θ_s for invader and defender is $\pi/4$, and the σ is set to 1.25. The parameter values for R-STDP, shown in Table II, are set through several simulations.

The reward coefficients are set manually. The τ_s in Table II defines the decaying rate of C in (4.1) that determines the R-STDP sensitivity to prior firings. According to (4.2), higher τ_s means that the R-STDP takes into account the activity of the two neurons that have fired in the relatively larger time window. Different studies have considered different values for this parameter.

As mentioned in Section II, the optimal capture point is the closest point from the reachable region to the target position. Figure 4.5 shows the agents during the training process. In epoch 5, the defender is not able to capture the invader, and the invader reaches the target. In epoch 7, the defender learns how to block and capture the invader. The defender has won the game. However, the invader should learn to reach the minimum distance from the target.

4.5.1 Simulation without noise

Figure 4.9 shows the performance after training. Each epoch has a maximum time of 10 seconds. After 14 epochs, the defender learned to capture the target, while the invader learned to reduce its distance from the target. According to the CO, the target is inside the defender's dominant region. Therefore, although the invader's velocity is higher than the

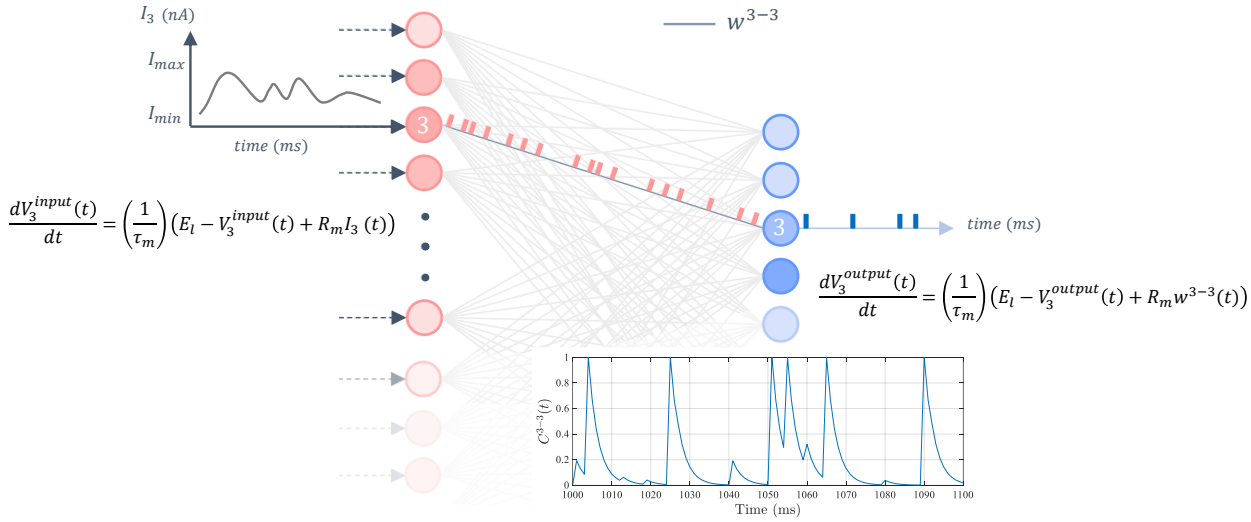

Figure 4.7: An illustration of input current to SNN and synaptic current to the output neurons after training for a single connection.

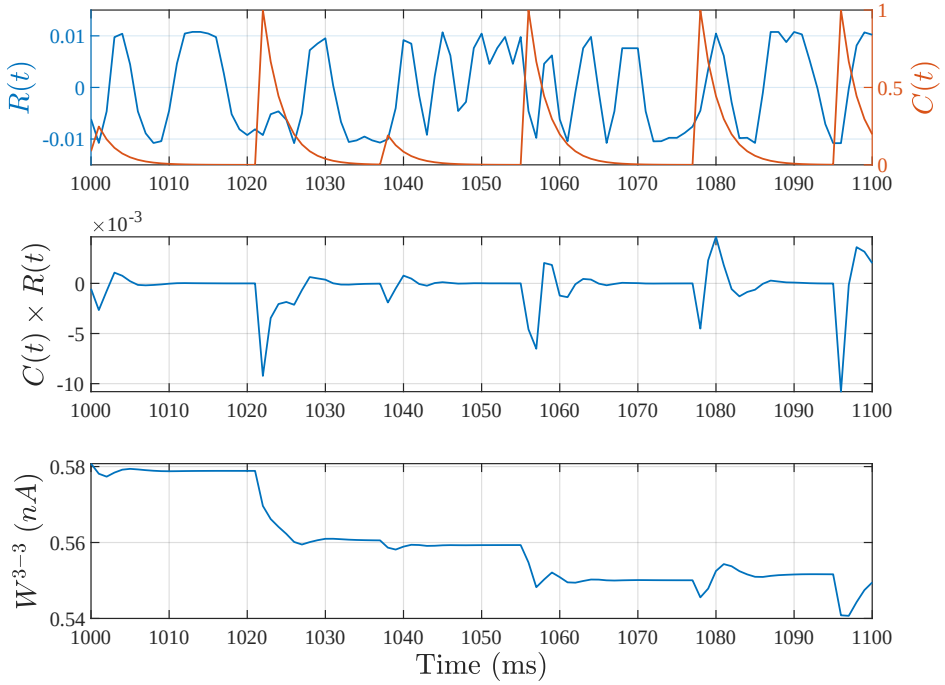

Figure 4.8: Reward, eligibility trace, and weight change during the simulation for W^{3-3}. The $C(t)R(t)$ changes the synaptic weight by considering activation strength and reward value.

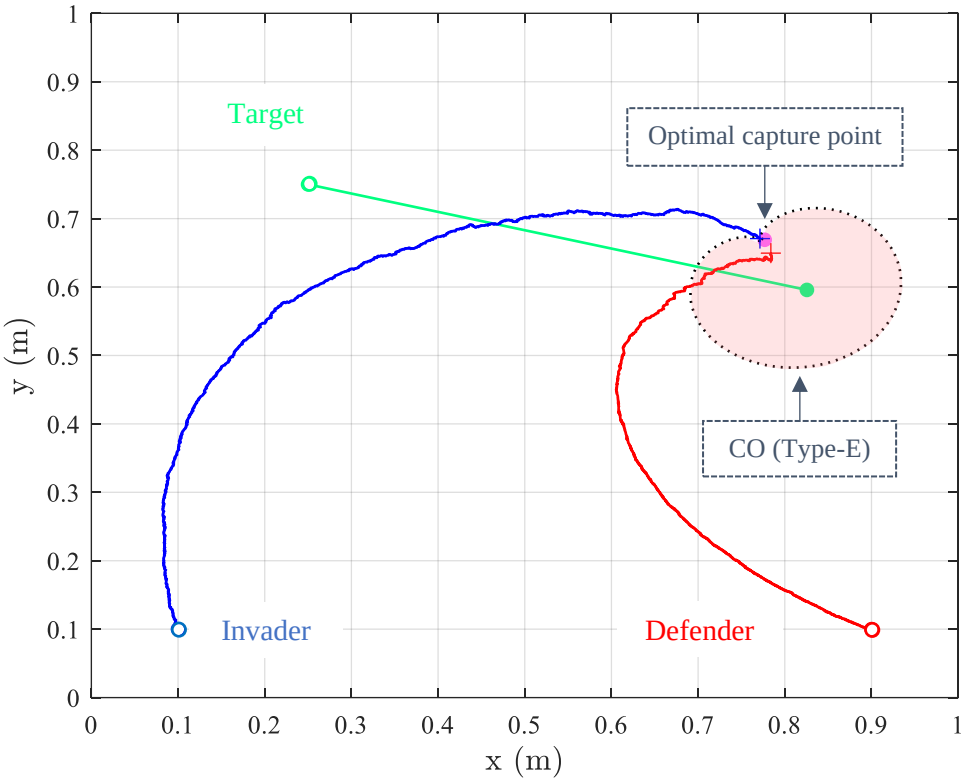

Figure 4.9: SNN's performance after training. The highlighted region shows the defender's dominant region. The purple dot is the optimal capture point.

defender's velocity, it cannot reach the target. In this situation, the optimal policy for the invader is to minimize its distance from the target.

According to Figure 4.9, the invader's SNN can find the optimal [85] 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 4.6 shows the changes in synaptic weights. Only the synaptic weights with maximum values are shown in this figure because the network has 200 synaptic connections. The minimum value is limited to zero because negative synaptic weights inhibit the post-synaptic neurons. This chapter does not consider the inhibition process. According to the figure, after almost 100 seconds of simulation time, the MSN process causes the synaptic weights to converge.

4.5.2 Simulation with noise

In Figure 4.10, we observe the performance of two methods, namely the SNN and the Cartesian Oval (CO) method, in the presence of noise. The noise in this experiment is introduced as white Gaussian noise, characterized by a mean of zero and a variance of 0.01. Both SNN and CO receive position data that has been corrupted by this noise.

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

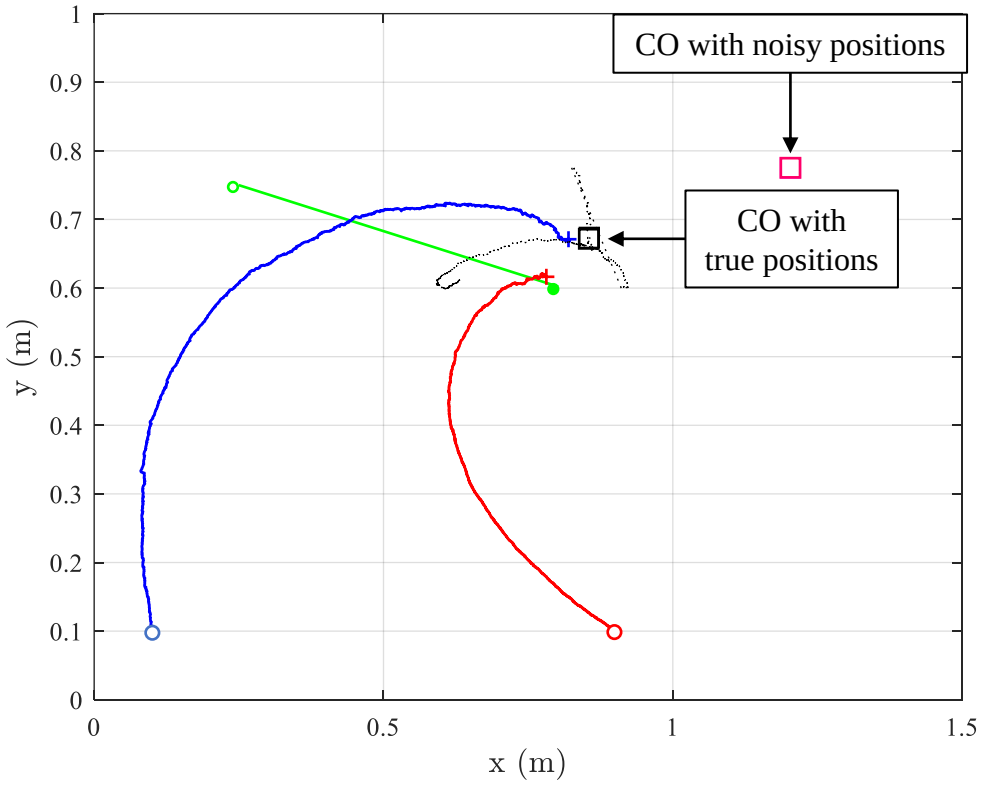

Figure 4.10: SNN's performance in noisy conditions. A white Gaussian noise with a variance of 0.01 is added to the measured inputs.

4.6 Conclusion

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

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

Chapter 5

Integration of R-STDP and Federated Learning

5.1 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 5.1, a swarm of agents (follower drones) flies around a leader. The leader is controlled from a remote base station, and the swarm agents should learn to fly safely with the leader. The leader sends its position to all agents, and each agent only sees two neighboring agents. The swarm aims to learn how to keep a commanded distance from each other and the leader. The commanded distance is provided from the leader. Each agent uses the onboard sensors to find the distance and line of sight from neighboring agents.

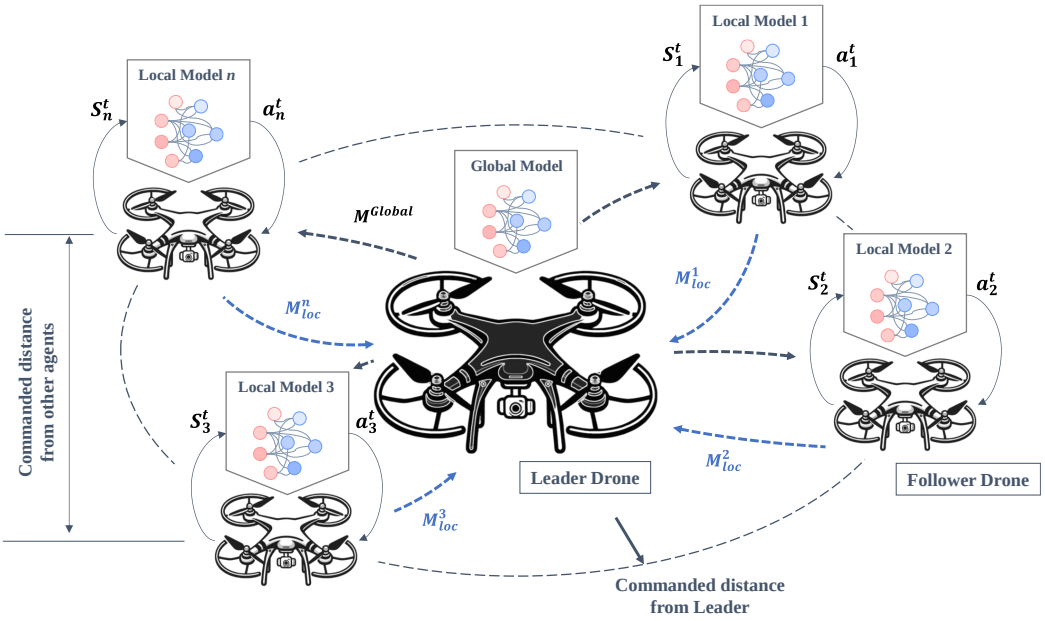

Figure 5.1: The central server (the leader) and the surrounding follower agents (white drones). The follower agents learn to fly in a formation to maintain the commanded distance. The Local models trained individually by follower agents are sent to the leader. The leader aggregates the models and sends back the global model for another round of training of the follower agents.

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

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

- The chapter presents a comprehensive method for stabilizing and enhancing the learning process in SNN. This method focuses on controlling the unbounded growth of synaptic weights in SNNs, utilizing a strategy that dynamically adapts to changes in reward conditions and coefficients. It introduces an innovative decay rate adjustment based on the status of synaptic weights. This approach not only enhances the responsiveness to weight changes but also preserves the shape of the Gaussian function formed by fuzzy encoding.

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

5.2 Proposed Method

5.2.1 Network Structure

This chapter assumes that each agent detects only two neighboring agents in addition to the leader. The information obtained from other agents includes the line-of-sight angle and the distance. Each agent's neural network consists of three sub-layers in the input layer, as shown in Figure 5.2. 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 utilizes the difference between the current distance and the commanded distance (r_{cmd}) to stimulate the input neurons.

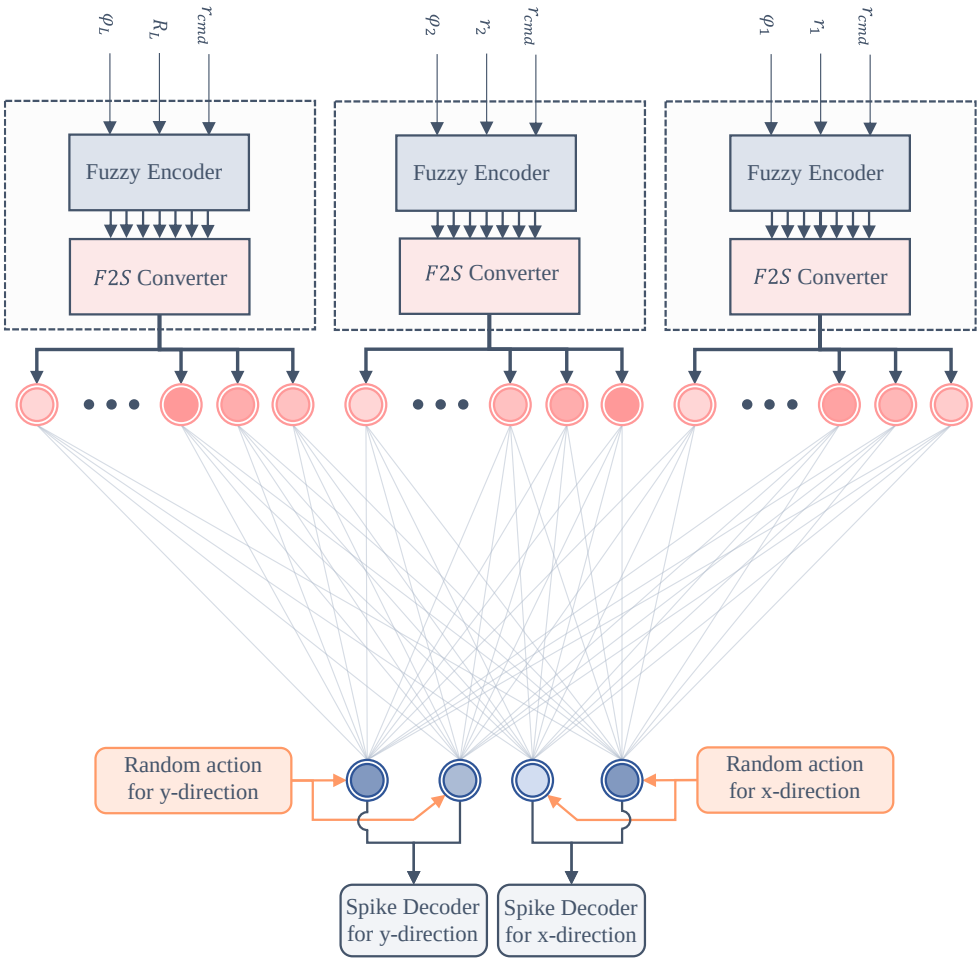

Figure 5.2: SNN structure with encoding and decoding layers. Each sub-layer consists of a fuzzy encoder and a Fuzzy-to-Spiking Converter, with the output layer receiving inputs from synaptic weights and a random action selector. During the training phase, the output layer receives input only from the random action selector, which then shifts to synaptic weight inputs after the training.

Every input sub-layer is split into two parts. The first part deals with distances greater than the commanded distance, while the second focuses on the space between the agent and the commanded distance. Within each part, the Line-of-Sight angle is encoded with fuzzy membership functions. The difference between the current and commanded distance is represented as the error that changes the maximum amplitude of the fuzzy membership

functions. We transform this difference into an amplitude value using the *tanh* function so that it is bounded between 0 and 1. An error of zero leads to an amplitude of zero, and as the error increases towards infinity, the amplitude approaches one. Consequently, the encoding function for the input layer is expressed as follows,

$$\mu_I(\phi_i, r_i) = |tanh(r_{cmd} - r_i)| \cdot \exp\left(-\frac{(\phi_i - \zeta)^2}{2\sigma^2}\right) \tag{5.1}$$

where ζ and σ are the Gaussian membership functions' center and standard deviation. The r_i is the distance from the corresponding agent, ϕ_i is the line-of-sight angle, and μ_I is the vector of the membership degrees. The firing strengths from fuzzy encoders are then converted to the spiking input based on the neuron model as follows [91],

$$I_{sub-layer} = \left(I^{max} - I^{min}\right) \mu_I(\phi_i, r_i) + I^{min} \tag{5.2}$$

or

$$I_{sub-layer} = \frac{\tau_m \left(V_{th} - V_{res}\right)}{\Delta t R_m} \mu_I(\phi_i, r_i) + \frac{V_{th} - E_l}{R_m} \tag{5.3}$$

The encoding process is shown in Figure 5.3. The Fuzzy to Spiking (F2S) block uses (5.3) to calculate the inputs for the associated sub-layer.

The output layer has two sub-layers, and each sub-layer has two neurons. The first sub-layer determines the Δx, and the second one determines Δy. The first neuron of the sub-layers is for negative values, and the second one is for positive values. Each neuron is associated with the output sign, and the magnitude of the Δx and Δy is encoded into the output sub-layers based on the minimum and maximum synaptic weights. Equation (5.3) is used to encode the magnitude of the random action into the output sub-layers. The only difference is that a function called (μ_O) is used to normalize the maximum step between 0 and 1 as follows,

$$\mu_x = \frac{\Delta x}{\Delta X_{max}} \tag{5.4}$$

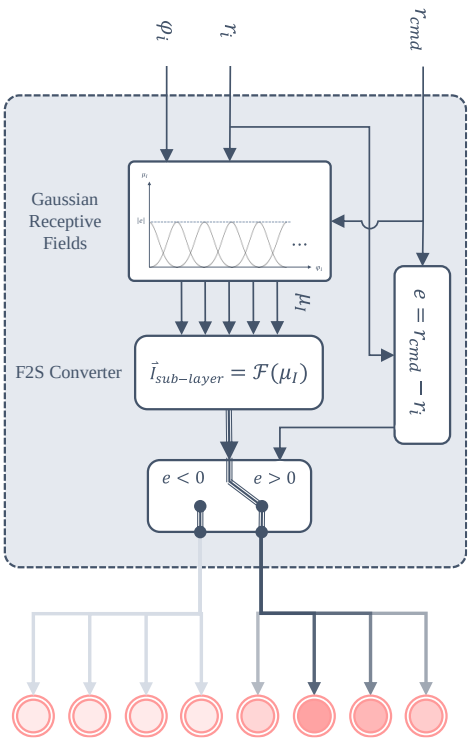

Figure 5.3: The fuzzy encoding principle for the input sub-layer.

$$\mu_y = \frac{\Delta y}{\Delta Y_{max}} \tag{5.5}$$

where Δx and Δy are selected actions, and ΔX_{max} and ΔY_{max} are maximum steps (displacements) in X and Y directions. Therefore, two random actions, one for Δx and one for Δy, are generated for the training process.

The decoding of the spiking output is determined by the difference in the firing rates of the output neurons within each sub-layer. Let us denote $f^{x+}(t)$ and $f^{x-}(t)$ as the firing rates of the first and second output neurons, respectively, in the x-direction sub-layer at time t. The equation for decoding this activity can be expressed as:

$$\Delta x_{decoded} = \left[\sum_{i=t-\Delta T}^{t} \left(f^{x+}(i) - f^{x-}(i) \right) \right] \Delta X_{max} \tag{5.6}$$

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

$$\Delta y_{decoded} = \left[\sum_{i=t-\Delta T}^{t} \left(f^{y+}(i) - f^{y-}(i) \right) \right] \Delta Y_{max} \tag{5.7}$$

The dynamic model of agents can be represented as,

$$\dot{X}(t) = \frac{\Delta x_{decoded}}{\Delta T} \tag{5.8}$$

$$\dot{Y}(t) = \frac{\Delta y_{decoded}}{\Delta T} \tag{5.9}$$

where ΔT is the time window the network updates weights.

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

$$\amalg_t = \gamma \cdot \amalg_{t-1} + (1 - \gamma) \cdot \epsilon_t \tag{5.10}$$

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

5.2.2 Training algorithm

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

$$\mathcal{R}_{Fi}^{Fj}(t) = \mathcal{C}_{Fi}^{Fj} \left[r_{Fi}^{Fj}(t-1) - r_{Fi}^{Fj}(t) \right] tanh(r_{Fi}^{Fj}(t) - r_{cmd}^{Fi}) \tag{5.11}$$

$$\mathcal{R}^{L}_{Fi}(t) = \mathcal{C}^{L}_{Fi}\left[r^{L}_{Fi}(t-1) - r^{L}_{Fi}(t)\right] tanh(r^{L}_{Fi}(t) - r^{L}_{cmd}) \tag{5.12}$$

where, $\mathcal{R}^{Fj}_{Fi}$, r^{Fj}_{Fi}, and r^{Fi}_{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^{L}_{cmd} represent the reward, distance, and the commanded distance between the follower agent i and the Leader (L), respectively. The $\mathcal{C}^{Fj}_{Fi}$ and $\mathcal{C}^{L}_{Fi}$ show the reward coefficients and the $tanh(r^{Fj}_{Fi}(t) - r^{Fi}_{cmd})$ and $tanh(r^{L}_{Fi}(t) - r^{L}_{cmd})$ 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.

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

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

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

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

$$\boldsymbol{\alpha}^{\mathcal{S}}(t) = \frac{1}{1 + \exp\left(|W^{\mathcal{S}}_{max}(t)| - \Psi^{\mathcal{S}}\right)}, \quad \left(\Psi^{\mathcal{S}} = \frac{\mathcal{R}^{\mathcal{S}}_{max}}{\mathcal{R}^{G}_{max}} I^{max}\right) \tag{5.13}$$

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

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

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

This chapter defines the decay rate as a function of the maximum synaptic weight among neurons in the set $\mathcal{S}$. This approach is designed to address a critical aspect: when the maximum synaptic weight in $\mathcal{S}$ reaches its peak ($|W^{\mathcal{S}}_{max}(t)| = \Psi^{\mathcal{S}}$), it is essential that the learning rate remains above zero. This condition is necessary to allow weight change and prevent the learning rate from stagnating at zero. Simultaneously, the learning rate must not exceed the maximum acceptable weight change, which is $\mathcal{A} \times \mathcal{R}^{G}_{max}$. With these considerations, we

propose that the decay rate should be set to $\mathcal{A}/\lambda \times \mathcal{R}^{G}_{max}$ when $|W^{\mathcal{S}}_{max}(t)| = \Psi^{\mathcal{S}}$ and increase to $\lambda\mathcal{A} \times \mathcal{R}^{G}_{max}$ when $|W^{\mathcal{S}}_{max}(t)| = 2\Psi^{\mathcal{S}}$, where λ is a coefficient that controls the rate of decay when $|W^{\mathcal{S}}_{max}(t)| > \Psi^{\mathcal{S}}$. By applying the mentioned condition and solving for the SoftPlus function, the decay rate function can be obtained as follows,

$$\Theta^{\mathcal{S}} = \left(\frac{\eta}{\beta}\right) \log\left(1 + \exp\left[\beta\left(|W^{\mathcal{S}}_{max}(t)| - \Psi^{\mathcal{S}}\right)\right]\right) \tag{5.14}$$

where $\eta = \frac{\mathcal{A}\mathcal{R}^{G}_{max} \ln\left(2^{\lambda^2}-1\right)}{\lambda\Psi^{\mathcal{S}} \log(2)}$ and $\beta = \frac{\ln\left(2^{\lambda^2}-1\right)}{\Psi^{\mathcal{S}}}$. Therefore, (5.14) can be represented as,

$$\Theta^{\mathcal{S}} = \left(\frac{\mathcal{A}\mathcal{R}^{G}_{max}}{\lambda \log(2)}\right) \log\left(1 + \exp\left[\left(\frac{\ln\left(2^{\lambda^2}-1\right)}{\Psi^{\mathcal{S}}}\right)\left(|W^{\mathcal{S}}_{max}(t)| - \Psi^{\mathcal{S}}\right)\right]\right) \tag{5.15}$$

The choice of setting the decay rate to $\lambda\mathcal{A} \times \mathcal{R}^{G}_{max}$ when $|W^{\mathcal{S}}_{max}(t)| = 2\Psi^{\mathcal{S}}$ is based on the feature of reward coefficients. Specifically, when the reward coefficient increases, leading to $|W^{\mathcal{S}}_{max}(t)| < \Psi^{\mathcal{S}}$, the $|W^{\mathcal{S}}_{max}(t)|$ is allowed to increase. Conversely, a decrease in the reward coefficient, resulting in $|W^{\mathcal{S}}_{max}(t)| > \Psi^{\mathcal{S}}$, necessitates a higher decay rate to reduce the maximum synaptic weight back to $\Psi^{\mathcal{S}}$. Incorporating the RCSE method into (1.19), the enhanced version of the R-STDP method is expressed as follows,

$$\dot{\boldsymbol{W}}(t) = \boldsymbol{\alpha} \odot \mathbf{STDP}(\tau) \odot \mathcal{R}(t) - \boldsymbol{\Theta}_{ij} \odot \operatorname{sgn}(\boldsymbol{W}) \tag{5.16}$$

where $\odot$ is the Hadamard product.

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

The RCSE method addresses the challenge of unbounded growth in synaptic weights within SNNs. By integrating a dynamic decay rate governed by the SoftPlus function, RCSE ensures that synaptic weight adjustments remain responsive to learning stages and reward changes. This method not only preserves the growth pattern induced by fuzzy input firing strengths but also provides a balanced approach to weight adjustment, preventing

saturation and facilitating continual learning adaptability. Moving forward, we will explore the integration of FL in the learning process.

5.2.4 Federated Learning for Consensus Flying

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

Asynchronous FL (AFL) has emerged as a significant advancement in federated systems, particularly in response to the challenges posed by device heterogeneity. This approach allows clients to upload their local model updates at different times rather than synchronously [92]. Such a method is particularly beneficial in reducing the negative impacts of device heterogeneity, which can include varying computational capacities and network connectivity among devices. In traditional FL setups, delays caused by poor network signals or unexpected client crashes can significantly prolong the time the server takes to receive updates from all clients. By adopting asynchronous aggregation, the server processes and aggregates models as they are received without synchronizing with all clients. This strategy accelerates the training process, making FL more efficient and adaptable to diverse client conditions.

Our proposed FL model aggregation algorithm aims to establish an efficient and dynamic system for global and local model publishing. This system relies on the similarity between consecutive global and local models and publishes updates only when significant changes are detected, thus avoiding redundant updates and improving overall efficiency. Unlike the uniform model updates in FedAvg [93], 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 [94], adding a layer of context sensitivity to our approach.

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

$$\overline{\boldsymbol{W}}_k(t) = \frac{1}{I_k^{max}}[\boldsymbol{W}_k(t)] \tag{5.17}$$

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,

$$\overline{\boldsymbol{W}}_G(t) = \frac{\sum_{k=1}^{N} \omega_k . \overline{\boldsymbol{W}}_k(t)}{\sum_{k=1}^{N} \omega_k} \tag{5.18}$$

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

$$\omega_k = \frac{1}{\sqrt{mn}} \|\overline{\boldsymbol{W}}_k(t)\|_F \exp\left(-\frac{t - T_k}{\tau_{cs}}\right) \quad (t \geq T_k) \tag{5.19}$$

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

Both agents and the central server employ an event-triggered mechanism for transmit-

ting 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,

$$\mathcal{D}_a(\overline{\boldsymbol{W}}_k(t), \overline{\boldsymbol{W}}_G(T_{cs}) = \frac{1}{2\sqrt{mn}}\sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}(a_{ij} - b_{ij})^2} \tag{5.20}$$

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

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

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

$$\mathcal{D}_G(\overline{\boldsymbol{W}}_G(t), \overline{\boldsymbol{W}}_G(T_{cs})) = \frac{1}{2\sqrt{mn}}\sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}(a_{ij} - b_{ij})^2} \quad (t \geq T_{cs}) \tag{5.21}$$

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

$$\dot{\boldsymbol{W}}_k(t) = (1-\delta(t-T_{cs}))\left[\boldsymbol{\alpha} \odot \mathbf{STDP}(\tau) \odot \mathcal{R}(t) - \boldsymbol{\Theta} \odot \operatorname{sgn}(\boldsymbol{W}_k(t))\right] + \delta(t-T_{cs}) I_k^{max}\left(\overline{\boldsymbol{W}}_G(t) - \overline{\boldsymbol{W}}_k(t)\right) \tag{5.22}$$

where δ is the Dirac delta function. Algorithm 3 shows the step-by-step implementation process of the proposed method.

Algorithm 3 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 (5.17)
    Aggregate models from all agents at the Central Server using (5.18)
    Calculate the $\mathcal{D}_G$ between the current and previous global models on the Central Server
    if $\mathcal{D}_G >$ the Central Server's threshold **then**
        Publish the global model
        set $T_{cs} = t$
    end if
    for each Agent in the network **do**
        if Central Server publishes a new global model **then**
            Update the local model of the Agent with the global model
        else
            Agents evaluate their local models against the latest global model ($\mathcal{D}_a$) using (5.20)
            if $\mathcal{D}_a >$ the Agent's threshold **then**
                Send the model to the Central Server
                set $T_k = t$
            end if
        end if
    end for
end if

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

5.3 Results and Discussion

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

5.3.1 Simulation without FL

In this simulation, we modeled five agents, each equipped with its own SNN model, capable of reaching a maximum speed of 1 m/s. The architecture of each agent's neural network included 72 input neurons. Since each agent was designed to detect three distinct objects within its environment, the input layer was organized into sub-layers, with 24 neurons dedicated to each object. The network's output layer comprised 4 neurons, divided equally to represent Δx and Δy movements. The SNN model in the simulation is a fully connected network, and the parameters of the Leaky Integrate-and-Fire (LIF) neuron are also represented in Table 5.1.

Table 5.2: Simulation Parameters

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

Table 5.1: Parameter values for LIF neuron model [95]

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

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

Table 5.2 shows the simulation parameters. The simulation was done in a 10 m by 10 m area, and the leader followed a circular path centered in (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 5.4 shows the definition of the distances.

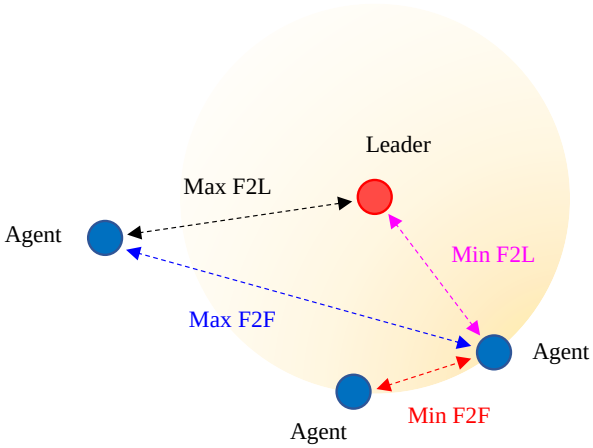

Figure 5.4: The measured distances during the test for evaluating the performance and detecting the collisions.

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

To ensure a fair comparison between different learning methods, we have compared our proposed approach with the FACL algorithm [96]. We opted for a fuzzy controller as it provides an explanation for how the SNN works. The FACL algorithm consists of two components, an actor and a critic. The actor is a fuzzy controller whose output serves as the control signal. The critic is a fuzzy inference system that stores the state value and represents the value function. The actor's output, which is the control signal, is computed as follows:

$$u_t = \sum_{l=1}^{L} \Phi^l \omega_t^l, \tag{5.23}$$

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

$$\Phi^l = \frac{\partial u}{\partial \omega_l} = \frac{\prod_{i=1}^{n} \mu^{F_i^l(x_i)}}{\sum_{l=1}^{L}(\prod_{i=1}^{n} \mu^{F_i^l(x_i)})}, \tag{5.24}$$

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

$$\hat{V}_t = \sum_{l=1}^{L} \Phi^l \zeta_t^l, \tag{5.25}$$

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

To train the followers, five initial sets of ω^ls and ζ^l are created and set to zero. Each follower generates an action with (5.23). The value of u_t is perturbed via a Gaussian noise with a mean of 0 and variance of ς ($u_t' = u_t + n(0, \varsigma)$). The agents take action u_t' and move to a new state. The reward function returns a signal corresponding to the quality of the actions. In this section, we implemented the reward functions in (5.11) and (5.12), where the weighting coefficients are set to 1. The temporal difference Δ is calculated for each agent as follows,

$$\Delta = \mathcal{R}_t + \Gamma \hat{V}_{t+1} - \hat{V}_t, \tag{5.26}$$

where $\mathcal{R}_t$ is the reward function and is the summation of (5.11) and (5.12). The actor is updated as follows,

$$\omega_{t+1}^{l} \longleftarrow \omega_{t}^{l} + \rho\Delta(u_{t}' - u_{t})\Phi^{l}. \tag{5.27}$$

The critic is updated as follows,

$$\zeta_{t+1}^{l} \longleftarrow \zeta_{t}^{l} + \kappa\Delta\Phi^{l}. \tag{5.28}$$

In (5.27) and (5.28), κ and ρ are the critic's and actor's learning rates. Table 5.3 shows the parameters used for the FACL algorithm. It should be mentioned that seven membership functions are used for each input, and thus, the total number of rules is $7 \times 7 \times 7 \times 7 \times 7 \times 7 = 117,649$.

Table 5.3: Simulation Parameters for the FACL Algorithm

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

Figure 5.5 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.

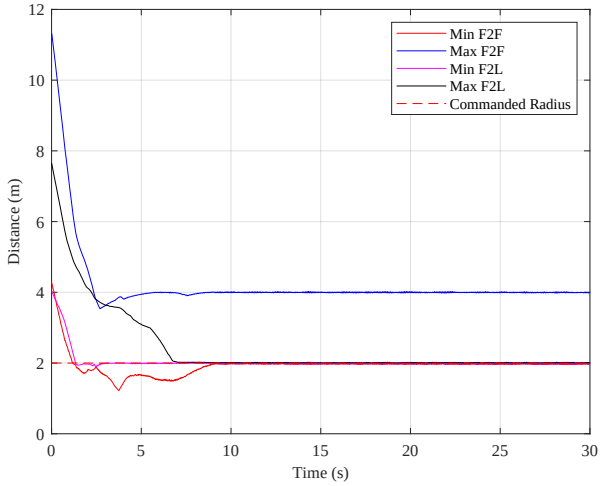

Figure 5.5: Distances during the test phase (RCSE method)

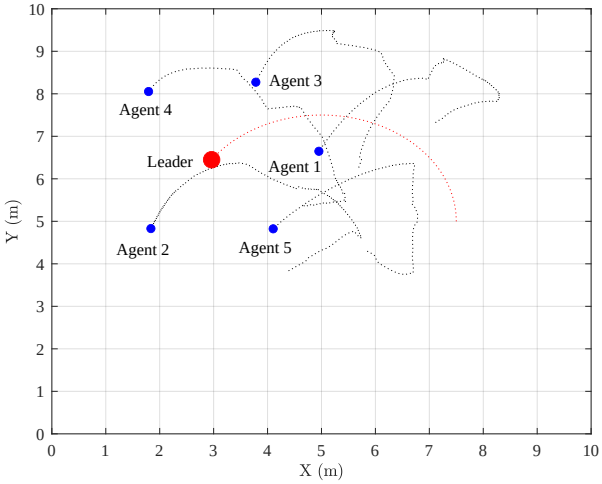

Figure 5.6: Agents' trajectory during the test phase (RCSE method)

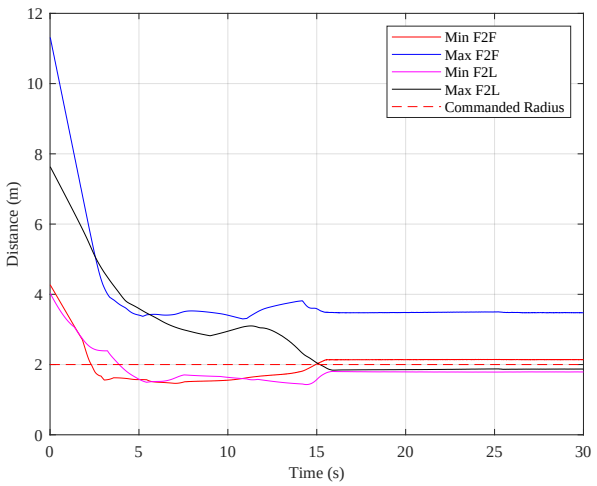

Figure 5.7: Distances during the test phase (FACL method)

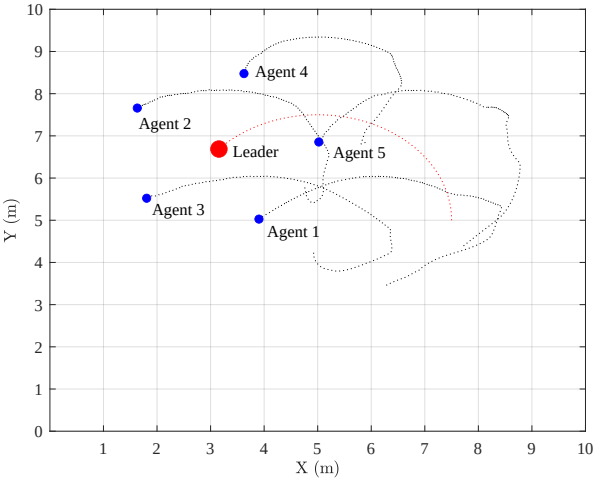

Figure 5.8: Agents' trajectory during the test phase (FACL method)

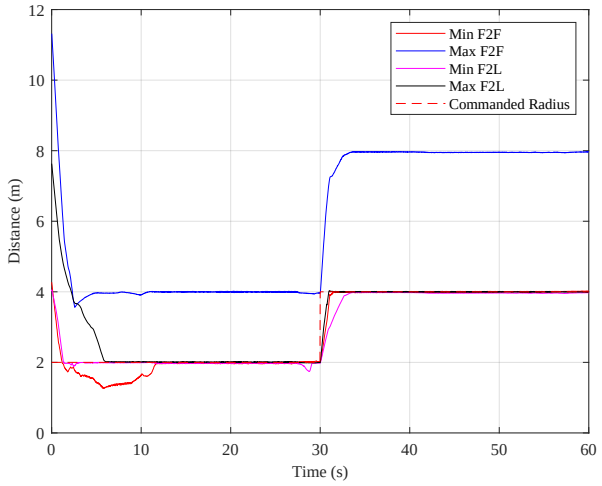

Figure 5.9: Change in commanded distance during the test phase (RCSE method)

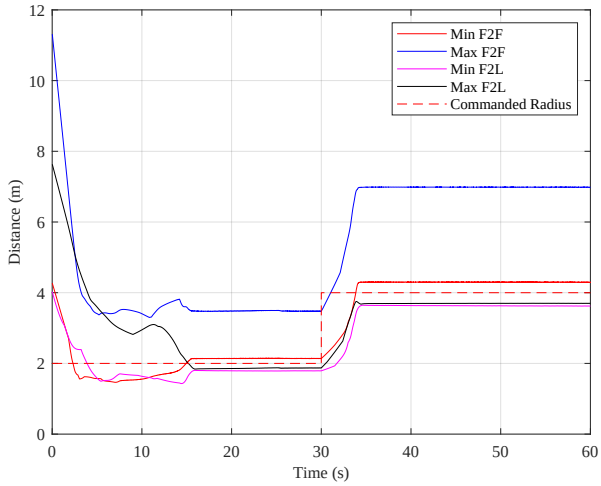

Figure 5.10: Change in commanded distance during the test phase (FACL method)

As mentioned in section 3, the input encoding uses the error between current and commanded distance. Therefore, one of the advantages of the encoding and learning method in this chapter is that the learned policies are independent of the commanded distance. Since the SNN uses the distance error, the commanded distance can be changed after training. Figure 5.9 shows the response of the agents to change in commanded distance after training. According to this figure, when the commanded distance is changed at 30 seconds, the swarm immediately responds to this change in 2.98 seconds without disrupting the formation or any collision.

Figure 5.7 shows the simulation results for the FACL algorithm. The training time was 1000 seconds, and the algorithm converged to the final solution in 667.26 seconds. According to the results, the agents achieved successful formation in approximately 15.85 seconds. Similarly, Figure 5.10 shows the simulation results for the FACL algorithm, where the commanded distance is initially set to 2 m and at $t = 30$, the commanded distance changes to 4 m. The results show that it takes 4.35 seconds for the swarm to adapt to the new command. It is worth mentioning that in Figures 5.7 and 5.10, the distances converge to the commanded distance but stay within a bias. Increasing the number of membership

functions can reduce the bias, but it increases the training time of the FACL algorithm.

After 600 seconds, the leader is changed into an obstacle, and its reward coefficient $\mathcal{C}_F^L$ is changed to 0.0175. The reward sign function, tanh in (5.12), is also changed to -1, so the reward function for the leader is changed as follows,

$$\mathcal{R}_{Fi}^L(t) = -\mathcal{C}_{Fi}^L \left[r_{Fi}^L(t-1) - r_{Fi}^L(t) \right] \tag{5.29}$$

When the leader is transformed into an obstacle, the encoding equation for the input layer must be changed. This is because the obstacle has no commanded distance, and the agents need to maintain a commanded distance only from each other. Therefore, the commanded distance from the obstacle encoder in the input layer must be removed. Therefore, (5.1) can then be rewritten as follows:

$$\mu_I(\phi_i) = \exp\left(-\frac{(\phi_i - \zeta)^2}{2\sigma^2} \right) \tag{5.30}$$

The simulation proceeded for an additional 1200 seconds, during which the synaptic weights were adjusted in accordance with the updated reward function. The results of the reward change are shown in Figure 5.11, which indicates that the agents quickly reduced their initial distance to the commanded distance. Simultaneously, the minimum distance from the obstacle, which was the leader, increased over time, indicating that the agents adapted their behavior to maintain a greater distance from the obstacle. Figure 5.12 shows the trajectory of each agent after the reward change.

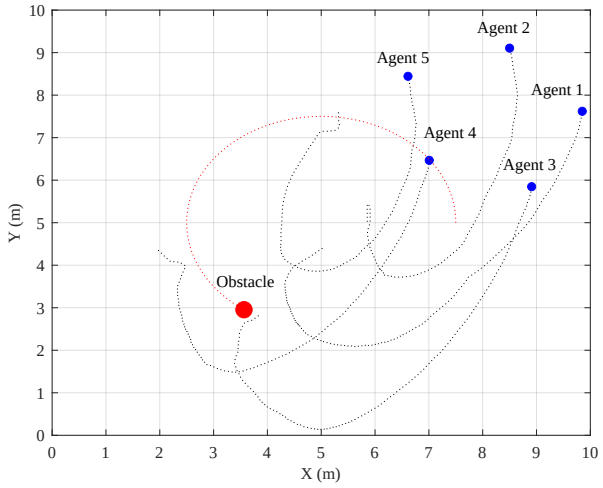

Figure 5.11: Agents' path after reward change (test phase)

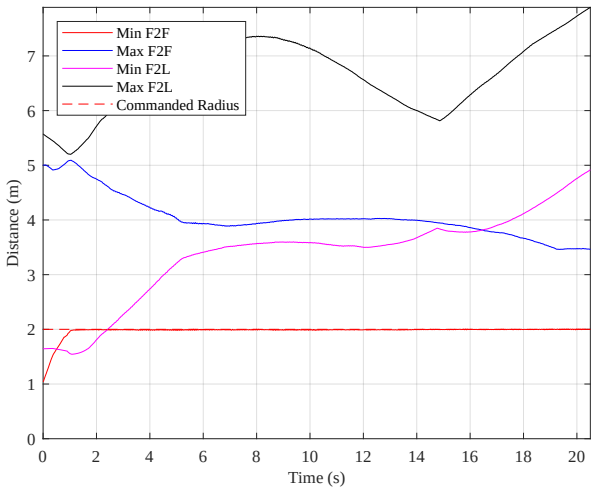

Figure 5.12: Distances after reward change (test phase)

In order to better understand the effect of reward change on SNN, the synaptic weights matrix before and after reward change has been illustrated in Figures 5.13 and 5.14, 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 5.2 along with the reward coefficients $\mathcal{C}_{Fi}^{Fj} = 0.02$ and $\mathcal{C}_{L}^{Fj} = 0.07$, and $r_{Fi}^{Fj} = 1\ m/s$ and $r_{Fi}^{L} = 0.1\ m/s$, the maximum rewards at each weight update interval (ΔT) for $\mathcal{R}_{Fi}^{Fj}$ and $\mathcal{R}_{Fi}^{L}$ are calculated as 4×10^{-4}

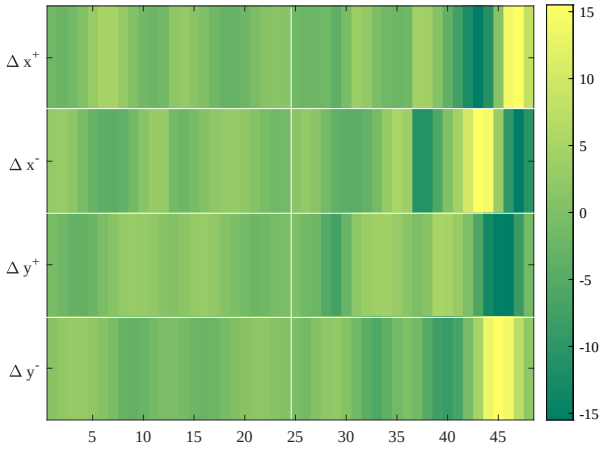

Figure 5.13: Synaptic Weights before Reward change

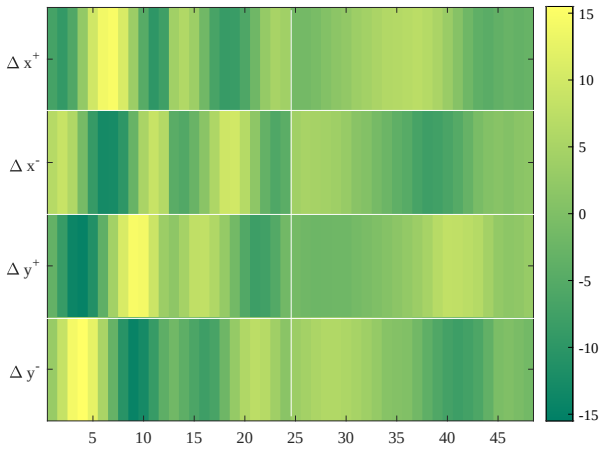

Figure 5.14: Synaptic Weights after Reward change

and 7.7×10^{-4}, respectively. Consequently, $\mathcal{R}^{G}_{max} = \max(\mathcal{R}^{Fj}_{Fi}, \mathcal{R}^{L}_{Fi}) = 7.7 \times 10^{-4}$. The Ψ^{S} for the follower section in the network is $\left[\frac{4\times10^{-4}}{7\times10^{-4}}\right] 15.5 = 8.0519$, and for the leader section, it is $\left[\frac{7\times10^{-4}}{7\times10^{-4}}\right] 15.5 = 15.5$. The η and β for the follower section within the network are $\frac{\mathcal{A}\mathcal{R}^{G}_{max} \ln(2^{\lambda^2}-1)}{\lambda\Psi^{\mathcal{S}} \log(2)} = 0.0011$ and $\beta = \frac{\ln(2^{\lambda^2}-1)}{\Psi^{\mathcal{S}}} = 2.152$, respectively. For the leader section, these values are $\frac{\mathcal{A}\mathcal{R}^{G}_{max} \ln(2^{\lambda^2}-1)}{\lambda\Psi^{\mathcal{S}} \log(2)} = 5.719 \times 10^{-4}$ and $\beta = \frac{\ln(2^{\lambda^2}-1)}{\Psi^{\mathcal{S}}} = 1.118$, respectively.

According to Figure 5.13, we can see a gradient formed because of the fuzzy nature of the input. Since the reward coefficients for followers and leaders are different, there is a difference in the maximum allowed synaptic weight for them. The proposed method for controlling the unbounded growth of synaptic weights has successfully stabilized the network while maintaining the fuzzy patterns of the synaptic connections.

Figure 5.14 shows the synaptic weights after the reward change. In this case, since the reward coefficients are changed, the η and β values are changed for the represented sub-layers, and the proposed method has helped the R-STDP algorithm to adjust the weights based on the new situation in the environment.

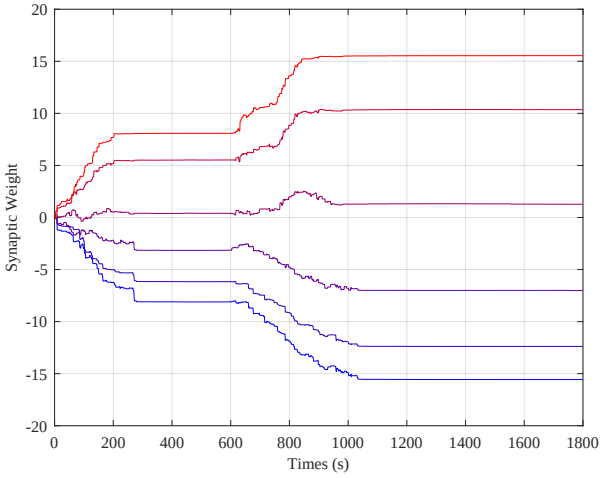

Figure 5.15: Synaptic weights increase after reward change

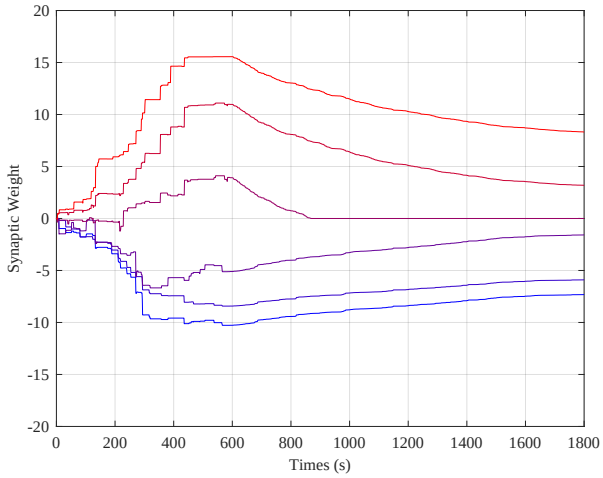

Figure 5.16: Synaptic weights decrease after reward change

5.3.2 Simulation with FL and R-CSE

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

Figure 5.17 shows the distances between agents and the leader before the reward change. According to the figure, the agents converge to the solution faster than the non-federated learning scenario without any error. Figure 5.18 shows the simulation results for the reward change scenario. According to the figure, the proposed aggregation method has improved the learning performance so that the swarm converges to the solution in 6 seconds.

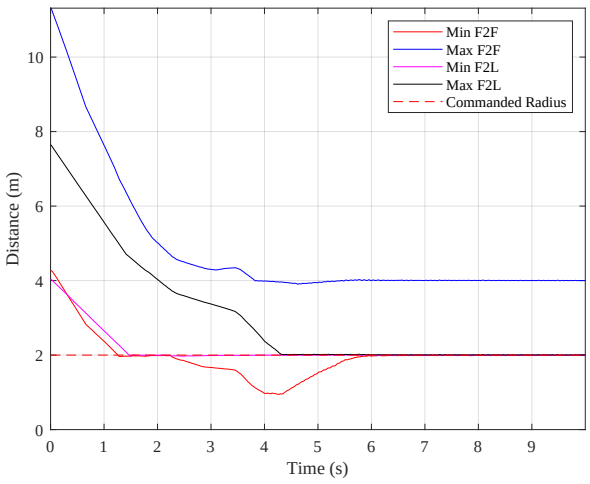

Figure 5.17: Distances during the test phase before reward change in the proposed aggregation method.

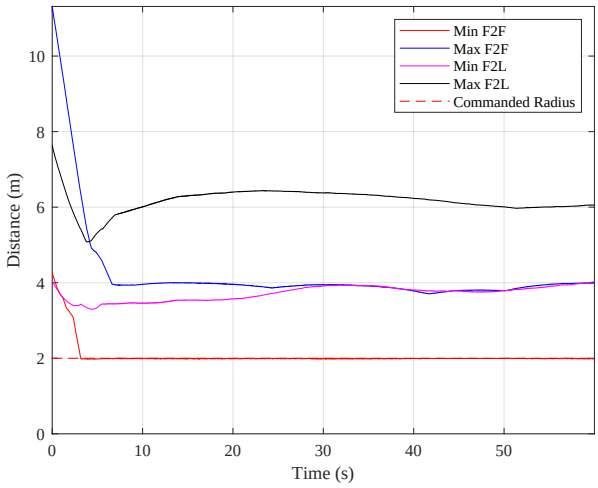

Figure 5.18: Distances during the test phase after reward change in the proposed aggregation method.

The norm of the synaptic weights can represent the changes in the synaptic weights due to the change in the environment during the training process, which can be used as an index to adjust the learning process in SNNs. The proposed aggregation method helps SNN models converge into a single model. Agents communicate with the Central Server (leader) during the training process, and Figure 5.20 shows the communication times. The aggregation sample 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 Frobenius norm determines the principle of the aggregation process, which results in small intervals of aggregation when the change rate is high, and larger intervals when it reduces.

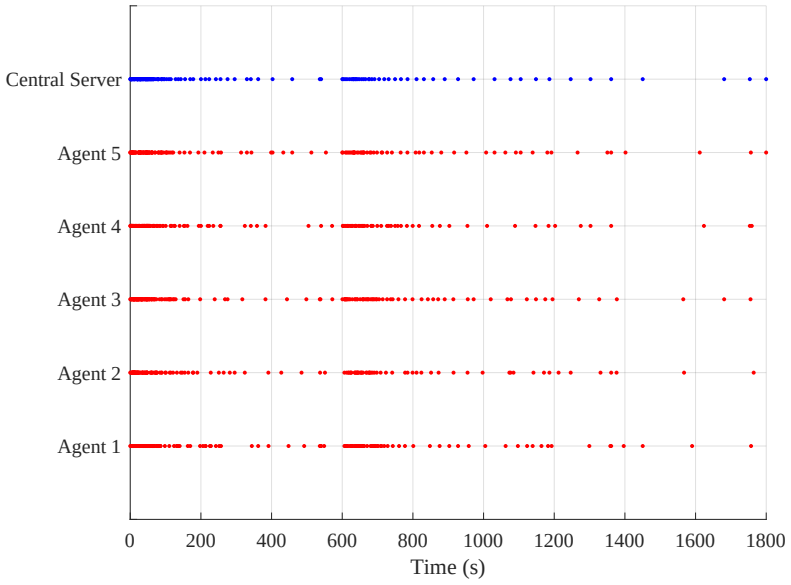

Figure 5.20: Communication times for agents and the Central Server (Leader). Red and blue dots show the times that agents and the Central Server have sent their model, respectively.

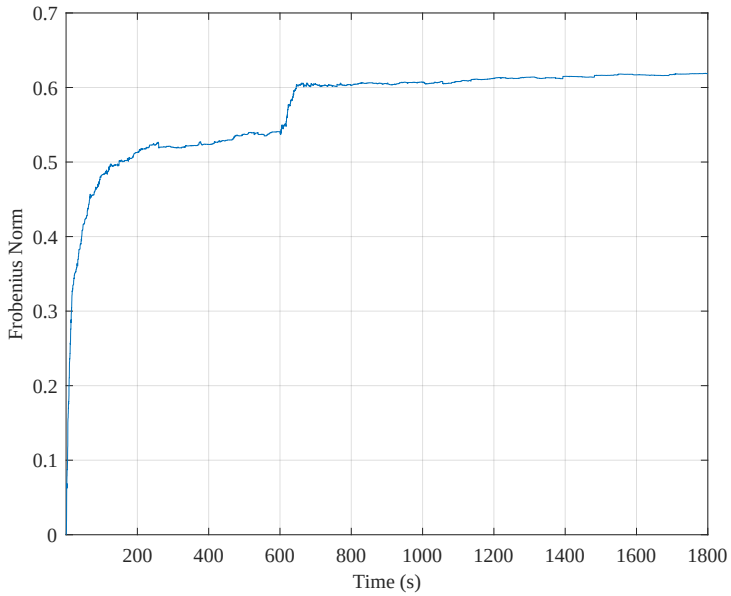

Figure 5.19: Frobenius norm of the agents during the learning phase. The reward changes for the Leader after 600 s.

Therefore, the aggregation frequency is very high initially, and it reduces after the change in synaptic weights goes to zero because of the learning rate in (5.13). Also, after the reward

changes and the leader becomes an obstacle after 600 seconds, the Euclidean distance between the converged and current models increases and reaches the threshold. As soon as the first agent sends its model to the server, the aggregation process starts again, and the agents adjust the associated synaptic weight.

Table 5.4: Comparative Performance Analysis of the Proposed Aggregation Algorithm+RCSE, RCSE, and FACL

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

Table 5.4 compares the proposed Federated Learning (FL) algorithm with the non-FL approach and FACL algorithm. The comparison focuses on three critical metrics: learning time, maximum error after convergence, and convergence time. The proposed FL algorithm demonstrates significant improvements in terms of efficiency and accuracy, as evidenced by its considerably shorter learning and convergence times and a notable reduction in error after convergence.

5.4 Conclusion

In this chapter, we presented a comprehensive approach addressing the challenges of uncontrolled growth in synaptic weights and the limited responsiveness of R-STDP to real-time changes within SNNs. Our proposed solution integrates the RCSE method with a dynamic aggregation interval in FL, significantly enhancing learning time and performance. The RCSE method introduces a novel mechanism to manage the unbounded growth of synaptic weights by dynamically adjusting the decay rate through the SoftPlus function. This adjustment is sensitive to the learning stages and rewards changes, ensuring that synaptic weight

adjustments remain responsive over time. By addressing the challenge of synaptic weight saturation, the RCSE method facilitates a balanced approach to weight adjustment, preventing network saturation and promoting continuous learning adaptability. We introduce a novel approach that uses FL in SNN and employs the Frobenius norm to adjust weighted aggregation in FL. Additionally, we include weight decay proportional to the time elapsed since an agent's last model publication. This improves the efficiency and responsiveness of the learning process. Our model's dynamic nature of the sample time adjusts based on the Euclidean norm. This metric measures the distance between the weight matrices of the agents and the server, determining reduced intervals for model publication. Our results show that the proposed aggregation method significantly accelerates agents' learning and performs better than non-federated learning and FACL cases. Moreover, the dynamic aggregation interval effectively reduces communication overhead between the agents and the central server, particularly after model convergence. This reduction is critical in scenarios where communication bandwidth is limited or costly. Our findings suggest that integrating R-STDP with federated learning, supported by our dynamic aggregation approach, provides a robust framework for advancing multi-agent reinforcement learning systems.

Chapter 6

Proposed Method

Bibliography

[1] 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.

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

[3] K. Kasaura, S. Miura, T. Kozuno, R. Yonetani, K. Hoshino, and Y. Hosoe, "Benchmarking actor-critic deep reinforcement learning algorithms for robotics control with action constraints," *IEEE Robotics and Automation Letters*, 2023.

[4] J. EEßerer, N. Bach, C. Jestel, O. Urbann, and S. Kerner, "Guided reinforcement learning: A review and evaluation for efficient and effective real-world robotics," *IEEE Robotics & Automation Magazine*, 2022.

[5] B. Li, Z. Fei, and Y. Zhang, "UAV communications for 5G and beyond: Recent advances and future trends," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2241–2263, 2018.

[6] R. Amin, L. Aijun, and S. Shamshirband, "A review of quadrotor UAV: control methodologies and performance evaluation," *International Journal of Automation and Control*, vol. 10, no. 2, pp. 87–103, 2016.

[7] S. Bouabdallah, A. Noth, and R. Siegwart, "PID vs LQ control techniques applied to an indoor micro quadrotor," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3, pp. 2451–2456, 2004.

[8] D. Valério and J. S. Da Costa, "Tuning of fractional PID controllers with ziegler–nichols-type rules," *Signal processing*, vol. 86, no. 10, pp. 2771–2784, 2006.

[9] M. Pena, E. Vivas, and C. Rodriguez, "Simulation of the quadrotor controlled with LQR with integral effect," in *ABCM Symposium Series in Mechatronics*, vol. 5, pp. 390–399, 2012.

[10] I. Sadeghzadeh, A. Mehta, Y. Zhang, and C.-A. Rabbath, "Fault-tolerant trajectory tracking control of a quadrotor helicopter using gain-scheduled PID and model reference adaptive control," in *Annual Conference of the PHM Society*, vol. 3, 2011.

[11] B. Han, Y. Zhou, K. K. Deveerasetty, and C. Hu, "A review of control algorithms for quadrotor," in *2018 IEEE International Conference on Information and Automation (ICIA)*, pp. 951–956, 2018.

[12] R. Zhang, Q. Quan, and K.-Y. Cai, "Attitude control of a quadrotor aircraft subject to a class of time-varying disturbances," *IET control theory & applications*, vol. 5, no. 9, pp. 1140–1146, 2011.

[13] L. Besnard, Y. B. Shtessel, and B. Landrum, "Quadrotor vehicle control via sliding mode controller driven by sliding mode disturbance observer," *Journal of the Franklin Institute*, vol. 349, no. 2, pp. 658–684, 2012.

[14] Y. Li and S. Song, "A survey of control algorithms for quadrotor unmanned helicopter," in *2012 IEEE fifth international conference on advanced computational intelligence (ICACI)*, pp. 365–369, 2012.

[15] F. Chen, W. Lei, K. Zhang, G. Tao, and B. Jiang, "A novel nonlinear resilient control for a quadrotor UAV via backstepping control and nonlinear disturbance observer," *Nonlinear Dynamics*, vol. 85, no. 2, pp. 1281–1295, 2016.

[16] S. H. Derrouaoui, Y. Bouzid, and M. Guiatni, "PSO based optimal gain scheduling back-stepping flight controller design for a transformable quadrotor," *Journal of Intelligent & Robotic Systems*, vol. 102, no. 3, pp. 1–25, 2021.

[17] J. Güldenring, P. Gorczak, F. Eckermann, M. Patchou, J. Tiemann, F. Kurtz, and C. Wietfeld, "Reliable long-range multi-link communication for unmanned search and rescue aircraft systems in beyond visual line of sight operation," *Drones*, vol. 4, no. 2, p. 16, 2020.

[18] N. Hosseini, H. Jamal, J. Haque, T. Magesacher, and D. W. Matolak, "UAV command and control, navigation and surveillance: A review of potential 5G and satellite systems," in *2019 IEEE Aerospace Conference*, pp. 1–10, 2019.

[19] S. K. Armah, S. Yi, and W. Choi, "Design of feedback control for quadrotors considering signal transmission delays," *International Journal of Control, Automation and Systems*, vol. 14, no. 6, pp. 1395–1403, 2016.

[20] H. Liu, T. Ma, F. L. Lewis, and Y. Wan, "Robust formation trajectory tracking control for multiple quadrotors with communication delays," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 6, pp. 2633–2640, 2019.

[21] R. Panuntun, O. Wahyunggoro, S. Herdjunanto, A. Rafsanzani, and N. Setiawan, "Networked control system in quadrotor altitude control with time delay compensation," in *Journal of Physics: Conference Series*, vol. 1577, pp. 12–31, 2020.

[22] R. Sanz, P. Garcia, Q.-C. Zhong, and P. Albertos, "Predictor-based control of a class of time-delay systems and its application to quadrotors," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 1, pp. 459–469, 2016.

[23] E. Slawiñski, D. Santiago, and V. Mut, "Control for delayed bilateral teleoperation of a quadrotor," *ISA transactions*, vol. 71, pp. 415–425, 2017.

[24] M. Sharma and I. Kar, "Control of a quadrotor with network induced time delay," *ISA transactions*, vol. 111, pp. 132–143, 2021.

[25] S. K. Armah and S. Yi, "Adaptive control for quadrotor UAVs considering time delay: Study with flight payload," *Robotics and Automation Engineering Journal*, vol. 2, no. 05, 2018.

[26] P. Farajiparvar, H. Ying, and A. Pandya, "A brief survey of telerobotic time delay mitigation," *Frontiers in Robotics and AI*, vol. 7, 2020.

[27] J. Yoo and K. H. Johansson, "Learning communication delay patterns for remotely controlled UAV networks," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 13216–13221, 2017.

[28] M. S. Asghar, S. Arslan, and H. Kim, "A low-power spiking neural network chip based on a compact lif neuron and binary exponential charge injector synapse circuits," *Sensors*, vol. 21, 2021.

[29] M. T. Ramezanlou, V. Azimirad, S. V. Sotubadi, and F. Janabi-Sharifi, "Spiking neural controller for autonomous robot navigation in dynamic environments," in *2020 10th International Conference on Computer and Knowledge Engineering (ICCKE)*, pp. 544–548, 2020.

[30] P. Blouw, X. Choo, E. Hunsberger, and C. Eliasmith, "Benchmarking keyword spotting efficiency on neuromorphic hardware," in *Proceedings of the 7th annual neuro-inspired computational elements workshop*, pp. 1–8, 2019.

[31] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, pp. 668–673, 2014.

[32] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *Ieee Micro*, vol. 38, pp. 82–99, 2018.

[33] 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.

[34] A. Alemi, C. Machens, S. Deneve, and J.-J. Slotine, "Learning nonlinear dynamics in efficient, balanced spiking networks using local plasticity rules," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[35] T. DeWolf, T. C. Stewart, J.-J. Slotine, and C. Eliasmith, "A spiking neural model of adaptive arm control," *Proceedings of the Royal Society B: Biological Sciences*, vol. 283, 2016.

[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, p. 104362, 2021.

[37] H. Raslan, H. Schwartz, and S. Givigi, "A learning invader for the "guarding a territory" game: A reinforcement learning problem," *Journal of Intelligent & Robotic Systems*, vol. 83, pp. 55–70, 2016.

[38] D. Li and J. B. Cruz, "Defending an asset: a linear quadratic game approach," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 47, pp. 1026–1044, 2011.

[39] J. F. Fisac and S. S. Sastry, "The pursuit-evasion-defense differential game in dynamic constrained environments," in *2015 54th IEEE Conference on Decision and Control (CDC)*, pp. 4549–4556, 2015.

[40] M. Chen, Z. Zhou, and C. J. Tomlin, "Multiplayer reach-avoid games via pairwise outcomes," *IEEE Transactions on Automatic Control*, vol. 62, pp. 1451–1457, 2016.

[41] E. Garcia, D. W. Casbeer, and M. Pachter, "Optimal target capture strategies in the target-attacker-defender differential game," in *2018 Annual American Control Conference (ACC)*, pp. 68–73, 2018.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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, 2022.

[48] R. Gupta and T. Alam, "Survey on federated-learning approaches in distributed environment," *Wireless personal communications*, vol. 125, no. 2, pp. 1631–1652, 2022.

[49] M.-A. Lahmeri, M. A. Kishk, and M.-S. Alouini, "Artificial intelligence for uav-enabled wireless networks: A survey," *IEEE Open Journal of the Communications Society*, vol. 2, pp. 1015–1040, 2021.

[50] 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.

[51] 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.

[52] S. K. Lee, "Distributed deformable configuration control for multi-robot systems with low-cost platforms," *Swarm Intelligence*, vol. 16, no. 3, pp. 169–209, 2022.

[53] M. Zhao, J. Xi, L. Wang, K. Xia, and Y. Zheng, "Edge-based adaptive secure consensus for nonlinear multiagent systems with communication link attacks," *Neurocomputing*, 2023.

[54] M. Yu, J. Xia, J.-e. Feng, S. Fu, and H. Shen, "Leader–follower output consensus of multiagent systems over finite fields," *Neurocomputing*, vol. 550, 2023.

[55] Z. Peng, Y. Jiang, L. Liu, and Y. Shi, "Path-guided model-free flocking control of unmanned surface vehicles based on concurrent learning extended state observers," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2023.

[56] C. Zheng and K. Lee, "Consensus decision-making in artificial swarms via entropy-based local negotiation and preference updating," *Swarm Intelligence*, pp. 1–21, 2023.

[57] Y. Yang, W. He, and S. Li, "Refined dynamic event-triggering cluster consensus of multiagent systems with fixed/switching topology," *IEEE Transactions on Cybernetics*, 2022.

[58] 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.

[59] 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, pp. 104–362, 2021.

[60] 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.

[61] D. Antonov, K. Sviatov, and S. Sukhov, "Continuous learning of spiking networks trained with local rules," *Neural Networks*, vol. 155, pp. 512–522, 2022.

[62] D. Xing, J. Li, T. Zhang, and B. Xu, "A brain-inspired approach for collision-free movement planning in the small operational space," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 5, pp. 2094–2105, 2022.

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

[64] G. Liu, W. Deng, X. Xie, L. Huang, and H. Tang, "Human-level control through directly trained deep spiking q-networks," *IEEE Transactions on Cybernetics*, 2022.

[65] S. A. Lobov, A. N. Mikhaylov, M. Shamshin, V. A. Makarov, and V. B. Kazantsev, "Spatial properties of stdp in a self-learning spiking neural network enable controlling a mobile robot," *Frontiers in neuroscience*, vol. 14, 2020.

[66] C. Teeter, R. Iyer, V. Menon, N. Gouwens, D. Feng, J. Berg, A. Szafer, N. Cain, H. Zeng, M. Hawrylycz, *et al.*, "Generalized leaky integrate-and-fire models classify multiple neuron types," *Nature communications*, vol. 9, 2018.

[67] J. Shen, Y. Zhao, J. K. Liu, and Y. Wang, "Hybridsnn: Combining bio-machine strengths by boosting adaptive spiking neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[68] 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.

[69] X. Cheng, T. Zhang, S. Jia, and B. Xu, "Meta neurons improve spiking neural networks for efficient spatio-temporal learning," *Neurocomputing*, vol. 531, pp. 217–225, 2023.

[70] P. Bertens and S.-W. Lee, "Network of evolvable neural units can learn synaptic learning rules and spiking dynamics," *Nature Machine Intelligence*, vol. 2, no. 12, pp. 791–799, 2020.

[71] M. Białas and J. Mańdziuk, "Spike-timing-dependent plasticity with activation-dependent scaling for receptive fields development," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 10, pp. 5215–5228, 2021.

[72] 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.

[73] D. Haşegan, M. Deible, C. Earl, D. D'Onofrio, H. Hazan, H. Anwar, and S. A. Neymotin, "Training spiking neuronal networks to perform motor control using reinforcement and evolutionary learning," *Frontiers in Computational Neuroscience*, vol. 16, 2022.

[74] 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.

[75] Y. Wang, S. Duan, and F. Chen, "Efficient asynchronous federated neuromorphic learning of spiking neural networks," *Neurocomputing*, vol. 557, 2023.

[76] 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.

[77] Y. Li, J. Yang, and K. Zhang, "Distributed finite-time cooperative control for quadrotor formation," *IEEE Access*, vol. 7, pp. 66753–66763, 2019.

[78] Y. Oda and M. Kumon, "Autonomous flight control of quadrotor helicopter by simple adaptive control with inner loop PD controller," *Journal of Robotics and Mechatronics*, vol. 30, no. 3, pp. 380–389, 2018.

[79] A. Al-Mahbashi, H. Schwartz, and I. Lambadaris, "Machine learning approach for multiple coordinated aerial drones pursuit-evasion games," in *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 642–647, 2020.

[80] S. Bouabdallah and R. Siegwart, "Backstepping and sliding-mode techniques applied to an indoor micro quadrotor," in *Proceedings of the 2005 IEEE international conference on robotics and automation*, pp. 2247–2252, 2005.

[81] M. Huang, B. Xian, C. Diao, K. Yang, and Y. Feng, "Adaptive tracking control of underactuated quadrotor unmanned aerial vehicles via backstepping," in *Proceedings of the 2010 American Control Conference*, pp. 2076–2081, 2010.

[82] Y. Ge, Q. Chen, M. Jiang, and Y. Huang, "Modeling of random delays in networked control systems," *Journal of Control Science and Engineering*, vol. 2013, 2013.

[83] 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.

[84] 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.

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

[86] 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.

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

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

[89] 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.

[90] 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.

[91] 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.

[92] 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.

[93] 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.

[94] 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.

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

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