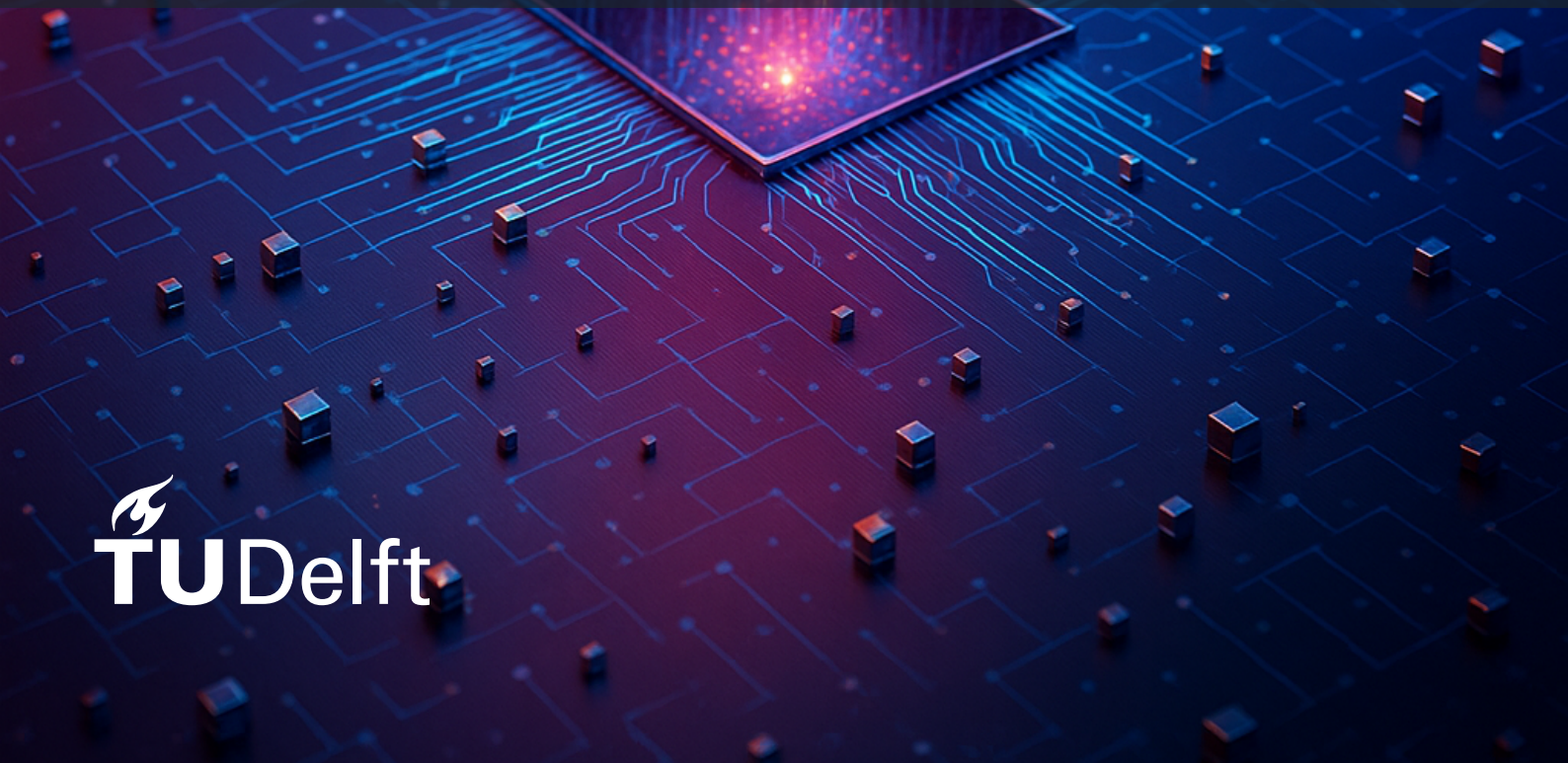
Spiking Neural Networks for High-Speed Continuous Quadcopter Control Using Proximal Policy Optimization

Toward Energy-Efficient Neuromorphic Control of Agile Drones

Michael Fernando van Breukelen Castillo



TU Delft

Spiking Neural Networks for High-Speed Continuous Quadcopter Control Using Proximal Policy Optimization

Toward Energy-Efficient Neuromorphic Control of Agile Drones

by

Michael Fernando van Breukelen Castillo

to obtain the degree of Master of Science
at the Delft University of Technology
to be defended publicly on September 19th, 2025 at 9:30.

Thesis committee:	
Chair:	Dr. G.C.H.E. de Croon
Supervisors:	Dr.ir. C. de Wagter
	Ir. R. Ferede
	Ir. R.W. Vos
External examiner:	Dr.ir. E.J.O. Schrama
Place:	Hall B, Faculty of Aerospace Engineering, Delft
Project Duration:	December 2024 - September 2025
Student number:	5097460

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

Cover image: Image generated by ChatGPT 5, OpenAI [1].

Faculty of Aerospace Engineering · Delft University of Technology

Preface

This thesis report is the culmination of approximately nine months of work on the favourite project I have undertaken. First, I am very grateful for the experience I have gained through this project. It has made me realise how meaningful it is to work hard on a topic I am passionate about and to conclude with a result of which I am very proud. Working on this project also taught me that hard work pays off. Even when I felt stuck during this journey, taking the next step mattered most. What matters is continuing to move forward, even if that sometimes means slowing down.

I would like to especially thank my family and my girlfriend, who, despite the distance, never fail to show their unconditional support and love throughout this process. They are the backbone of my life and have witnessed how much effort and energy this project required of me. It is thanks to them that I am here, happy and alive. I would also like to thank the friends that I have made these last few years during my studies. It has been incredibly helpful to have great friends by my side who all share the same struggle, and who also reminded me to take a step back sometimes to relax.

I am sincerely grateful to my supervisors, Christophe and Guido, for kind and generous guidance, constructive feedback, and steady support throughout this thesis, and for welcoming me as a Master's student in their group. I also thank Robin and Reinier for their thoughtful feedback and their willingness to help whenever I was stuck.

Contents

Nomenclature

List of Abbreviations

AdEx	Adaptive Exponential Integrate-and-Fire
AER	Address-Event Representation
AI	Artificial Intelligence
ANN	Artificial Neural Network
CMOS	Complementary Metal–Oxide–Semiconductor
CNN	Convolutional Neural Network
DDPG	Deep Deterministic Policy Gradient
DNN	Deep Neural Network
DOF	Degree of Freedom
DPG	Deterministic Policy Gradient
DRL	Deep Reinforcement Learning
EOM	Equations of Motion
FWHM	Full Width at Half Maximum
GPU	Graphics Processing Unit
IF	Integrate-and-Fire
INDI	Incremental Nonlinear Dynamic Inversion
ISI	Interspike Interval
LIF	Leaky Integrate-and-Fire
Loihi	Intel Neuromorphic Research Chip
LSTM	Long Short-Term Memory
LTD	Long-Term Depression
LTP	Long-Term Potentiation
ML	Machine Learning
MLP	Multilayer Perceptron
MSE	Mean Squared Error
nMAE	Normalized Mean Absolute Error
PID	Proportional–Integral–Derivative
PILCO	Probabilistic Inference for Learning Control
PPO	Proximal Policy Optimization
QIF	Quadratic Integrate-and-Fire
R-STDP	Reward-modulated Spike-Timing-Dependent Plasticity
RL	Reinforcement Learning
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
RQ	Research Question
SAC	Soft Actor–Critic
SGD	Stochastic Gradient Descent
SNN	Spiking Neural Network
SpiNNaker	Spiking Neural Network Architecture
SRM	Spike Response Model
STDP	Spike-Timing-Dependent Plasticity
TD3	Twin Delayed Deep Deterministic Policy Gradient
TRPO	Trust Region Policy Optimization
TTFS	Time to First Spike
VLSI	Very Large-Scale Integration

List of Symbols

β	Decay factor $e^{-\Delta t/\tau_m}$
$\hat{\lambda}$	Estimated firing rate
$\hat{x}(t)$	Reconstructed signal estimate
$\hat{x}_{ml}$	Maximum likelihood estimate of stimulus
$\hat{x}_{pv}$	Population vector estimate of stimulus
$\lambda(t)$	Instantaneous firing rate at time t
μ_i	Preferred stimulus of neuron i
τ_m	Membrane time constant

τ_r Reconstruction time constant

τ_s Synaptic time constant

τ_{ts} Time surface decay constant

ϑ Error threshold in sigma–delta coding

A Kernel area in rate decoding

C_m Membrane capacitance

d^{π_θ} Discounted state visitation distribution

$e(t)$ Residual error signal in sigma–delta coding

E_L Leak reversal potential

$FWHM$ Full Width at Half Maximum of tuning curve

g_L Leak conductance

$H(t)$ Heaviside step function

i_t Effective input current at discrete step t

$I_{\text{ext}}(t)$ External input current at time t

$I_{\text{syn}}(t)$ Synaptic input current at time t

$J(\pi)$ Objective function of a policy π

$k(t)$ Synaptic or reconstruction kernel

n Number of spikes in a window

n_i Spike count of neuron i

$p(x, y, t)$ Event polarity at pixel (x, y) and time t

$Q^\pi(s, a)$ Action-value function under policy π

$r(s, a)$ Reward function

$r_i(x)$ Mean firing rate of neuron i for input x

R_m Membrane resistance

r Maximum firing rate

$S(x, y, t)$ Time surface value in event camera coding

s_t Binary spike indicator at time t

t Time

t_j Spike time of neuron j

t_s Spike time in TTFS coding

$t_{\text{last}}(x, y)$ Timestamp of last event at pixel (x, y)

$V(t)$ Membrane potential at time t

$V^\pi(s)$ State-value function under policy π

v_t Membrane potential relative to rest at time t

V_{reset} Reset potential after spike

V_{th} Spike threshold potential

$y(t)$ Filtered activity at time t

List of Figures

1 Introduction

The rapid proliferation of artificial intelligence (AI) models in recent years has dramatically expanded the capabilities of autonomous systems across domains. This progress, however, comes with escalating computational and energy demands: modern deep learning models require substantial computing power (and thus energy) for both training and inference. As AI is deployed from cloud datacenters to edge devices, there is a growing imperative to develop energy-efficient computing methods that can deliver real-time performance under tight power budgets. This challenge is especially pronounced in embedded platforms and robotics, where on-board resources are limited and efficiency is paramount [2].

One promising approach to address these challenges is neuromorphic computing, a biologically inspired approach that emulates the brain's energy-efficient information processing. Neuromorphic systems often rely on *spiking neural networks* (SNNs) as their computational backbone. In an SNN, neurons communicate via discrete spikes (analogous to the action potentials in biological neurons) and operate asynchronously, updating only when events (spikes) occur. This event-driven processing enables high concurrency and can drastically reduce power consumption, since inactive neurons draw virtually no compute energy. These characteristics make SNNs highly attractive for energy-efficient, low-latency control tasks, especially when deployed on specialised neuromorphic hardware designed for spike-based computation (e.g. Intel's Loihi chip [3]). By harnessing sparse and timed neural activity, neuromorphic platforms aim to provide the intelligence of deep neural networks at a fraction of the energy cost.

Despite their potential, spiking neural networks have seen limited use in closed-loop control of complex robots. Prior studies have explored SNNs in areas like vision processing and simple motor tasks, but applications to high-speed continuous control remain scarce [4]. In particular, achieving agile flight control with SNNs is an open challenge: the fast dynamics of a quadcopter require rapid, precise control outputs, and training an SNN to meet these demands involves handling spatio-temporal gradient propagation in learning. Recent advances in gradient-based learning for SNNs (e.g. surrogate gradient methods) are beginning to overcome these hurdles, allowing SNNs to be trained with deep reinforcement learning techniques [4, 5]. These developments set the stage for investigating SNN controllers in demanding real-time control scenarios.

High-speed quadcopter flight is an excellent test-bed for energy-efficient, real-time control due to its strict constraints on latency and power. Micro aerial vehicles (MAVs) like racing quadcopters must run their control loops at high frequency (hundreds of updates per second) on weight- and power-constrained hardware. The onboard computer and battery supply are limited, so any viable controller must utilize computational resources efficiently. At the same time, the controller must react within a few milliseconds to rapidly changing dynamics to maintain stability and performance. Traditionally, quadcopter control is handled by cascaded PID controllers, but there is growing interest in end-to-end learned controllers that can directly map sensor inputs to motor commands. In fact, recent work has demonstrated that deep neural network policies can achieve expert-level agility in quadcopter flight. For example, Ferede *et al.* trained an end-to-end artificial neural network (ANN) controller that learns to race a quadcopter through gates in minimum time, outperforming a classical control pipeline in both simulation and hardware tests [6]. Similarly, other researchers have shown that deep reinforcement learning can produce high-performance drone racing controllers [7]. These successes with ANN-based controllers inspire the central objective of this thesis:

To develop and evaluate a spiking neural network controller for high-speed quadcopter flight, with a focus on spiking network design, training methodology, and benchmarking against ANN baselines in terms of control performance and stability

The primary objective of this research is to develop and evaluate a spiking neural network controller for high-speed quadcopter flight using reinforcement learning techniques. In pursuit of this goal, we design a fully spiking actor–critic neural network (policy network) and train it end-to-end with Proximal Policy Optimization (PPO) [8]. The SNN controller is composed of leaky integrate-and-fire (LIF) neurons and is trained with surrogate gradient descent to handle the non-differentiable spike activations. To generate continuous control outputs from binary spikes, we employ a spike-rate decoding approach: the network's spike activity is averaged over short integration windows to produce smooth motor command signals. Importantly, the SNN architecture is carefully aligned with a comparable ANN baseline controller (in network depth and layer sizes) to enable a fair performance comparison. We focus on a challenging task of agile quadcopter navigation (a timed gate-to-gate flight task) and train the SNN policy in simulation. The trained controller is then deployed on a physical quadcopter to assess real-world performance, closing the loop from simulation to hardware.

Extensive simulation and flight experiments demonstrate that the spiking controller can match and surpass the performance of its ANN counterpart. In simulation trials, the SNN policy achieves higher reward scores and exhibits greater robustness (fewer crashes or failures) compared to the baseline ANN, despite both being trained to the same performance criteria. More notably, in real-world high-speed flight tests, the SNN controller outperforms the ANN baseline, particularly in terms of robustness and control performance. The SNN-driven quadcopter achieves higher average velocities and successfully clears more gates within a fixed time horizon, all while leveraging the inherent efficiency of spike-based computation. These results indicate that neuromorphic control policies can not only reach the level of conventional neural controllers but can even provide advantages in resilience and potentially energy usage, a promising step toward energy-efficient, AI-driven flight.

Thesis outline. The remainder of this thesis is structured as follows: Chapter 2 provides background on quadcopter dynamics, reinforcement learning for control, and the fundamentals of neuromorphic computing and spiking neural networks. Chapter 3 details the design of the spiking neural network controller and the experimental setup, including the task formulation, network architecture, PPO training procedure, and techniques such as spike decoding and domain randomization used to enhance robustness. Chapter 4 presents the results of simulation experiments, comparing the performance of the SNN controller against the ANN baseline under various conditions. Chapter 5 describes the real-world flight tests carried out to validate the SNN controller on an actual quadcopter, and it analyzes the controller's latency and flight performance. Finally, Chapter 6 concludes the thesis by summarizing the main findings, addressing the research questions, and discussing avenues for future work in neuromorphic agile flight control.

Part I

Scientific Article

Spiking Neural Networks for High-Speed Continuous Quadcopter Control Using Proximal Policy Optimization

M.F. van Breukelen Castillo *
Delft University of Technology, Kluyverweg 1, 2629 HS Delft, The Netherlands

ABSTRACT

We present the first demonstration of a fully spiking actor-critic neural network policy, trained via Proximal Policy Optimization (PPO), for continuous control of an agile high-speed quadcopter in a gate-based navigation task. The spiking neural network (SNN) controller employs Leaky Integrate-and-Fire neurons with surrogate gradient training and spike-rate decoding over multiple integration cycles, and it is benchmarked against a comparable artificial neural network (ANN) controller in both simulation and real-world flight tests. Results show that despite being trained to the same reward level, the SNN achieves superior performance in simulation, achieving higher episode rewards, greater robustness and reduced crash rate. Additionally, in 12-second real-world trials, the SNN outperforms the ANN, attaining a higher average reward (70.63 vs 59.77), greater mean velocity (7.94 vs 6.99 m/s), and more gates cleared (46.33 vs 40.67). An analysis of the spike integration cycle count reveals a clear trade-off: lower cycle counts (fewer integration steps per control update) reduce control output resolution and hinder learning, whereas higher cycle counts improve smoothness but increase inference latency. Moderate cycle counts (5 or 8) provide the best balance, yielding high rewards, smoother outputs, and low execution time overhead. These findings represent a key step forward for neuromorphic control in embedded autonomous systems, demonstrating that SNN-based policies can outperform conventional ANN controllers in high-speed, agile robotic tasks.

1 INTRODUCTION

Over the past two decades, the rise of machine learning and artificial intelligence has dramatically expanded the capabilities of autonomous systems. From smartphones and self-driving vehicles to large language models, intelligent computation is increasingly integrated into everyday life. However, this progress comes at a cost: modern deep learning models are energy-intensive, requiring substantial compute resources for both training and inference. As neural networks become more prevalent, there is a growing need for energy-efficient models that can deliver competitive real-time performance, especially in embedded systems. This challenge is particularly present in micro aerial vehicles (MAVs), which demand fast, low-latency control under strict power and hardware constraints. High-performance tasks such as drone racing or agile navigation push the limits of onboard computation, as controllers must operate at high update frequencies on lightweight, battery-powered platforms. While conventional artificial neural networks (ANNs) have demonstrated strong control capabilities [1, 2], their poor energy efficiency remains a significant limitation.

Spiking Neural Networks (SNNs) have emerged as a promising alternative. Inspired by the sparse, event-driven signalling of biological neurons, SNNs operate asynchronously through binary spikes, enabling low-power computation with minimal latency. These properties make SNNs particularly attractive for real-time control on embedded platforms, especially when deployed on neuromorphic hardware such as Intel's Loihi [3].

Despite the potential of drastically reduced energy consumption, the use of SNNs in closed-loop robotic control has remained very limited. Prior work has explored their use in visual processing [4], neuromorphic sensing [5], and motor tasks [6], but applications in high-speed continuous control for quadrotors have not been demonstrated. Reinforcement learning (RL) provides a natural framework for this setting, and recent methods have enabled SNNs to be trained with surrogate gradients [7, 8]. Yet these advances have largely remained confined to low-dimensional benchmarks, and robust real-time deployment in fast physical systems is still lacking.

Although limited overall, several recent works illustrate that SNNs are beginning to move from theory to practice. Deep neural networks have already shown strong control capabilities for quadrotors [9], and first demonstrations of SNN-based flight have appeared. Paredes-Vallés et al. implemented a spiking vision-to-control pipeline on Intel Loihi, enabling autonomous drone flight at 200 Hz with microjoule-level inference cost [10]. Stroobants et al. realized end-to-end spiking attitude control on a Crazyflie, achieving near-conventional performance at 500 Hz [11]. At the algorithmic level, Xu et al. proposed a proxy-target framework that stabilizes training for continuous control benchmarks [12]. These studies demonstrate the promise of neuromorphic approaches, but none address the challenge of learning high-speed, continuous quadrotor control directly with fully spiking networks.

In this work, we present the first implementation of a fully spiking actor-critic network trained using Proximal Policy Optimization (PPO) for continuous quadcopter control in a high-speed gate navigation task. The network is composed of Leaky Integrate-and-Fire (LIF) neurons and trained end-to-end using surrogate gradient descent. To convert sparse spike activity into smooth control signals, we employ spike-rate decoding over multiple integration cycles. The SNN architecture is carefully matched in structure and size to an ANN baseline [13], to ensure a fair and meaningful comparison.

Our results show that the SNN not only achieves superior

*Code=https://github.com/michael2992/msc_spiking_quadcopter_control

performance in simulation but also outperforms the ANN in real-world flight tests, demonstrating higher robustness, increased average velocity, and traversing more gates within a 12 second interval. We further evaluate the impact of spike-integration cycles on control performance and system latency. This work bridges the gap between neuromorphic computing and high-performance reinforcement learning, offering a foundation for deploying SNN-based control policies on energy-constrained aerial robots. The code for the training, simulation, and flight data analysis for the project is available at: `https://github.com/michael2992/msc_spiking_quadcopter_control`, and the videos of the real flight tests are available at: `https://drive.google.com/drive/folders/1uGINGe71wu0Hrh0_ZBDAHMIO-s6B5qd3?usp=drive_link`.

2 Methodology

This section outlines the quadcopter model, simulation environment, reinforcement learning framework and spiking network adaptations used for developing and evaluating the SNN controller. Our approach builds directly on the environment and task setup from [14] and [13], but is distinct in that it focuses exclusively on the 5-inch quadcopter platform with a focus on training the spiking actor-critic policy.

2.1 Quadcopter Model

We adopt the parametric quadcopter dynamics model from [13] defined in continuous time. The state vector $\mathbf{x}$ and input vector $\mathbf{u}$ are defined as:

$$\mathbf{x} = [\mathbf{p}, \mathbf{v}, \boldsymbol{\lambda}, \boldsymbol{\Omega}, \boldsymbol{\omega}]^T, \quad \mathbf{u} = [u_1, u_2, u_3, u_4]^T \in [0,1]^4 \tag{1}$$

Here, $p \in \mathbb{R}^3$ is position, $v \in \mathbb{R}^3$ is velocity, $\lambda \in \mathbb{R}^3$ represents Euler angles, $\Omega \in \mathbb{R}^3$ the body rates, and ω_i the propeller speeds in rad/s. The control input $\mathbf{u} \in [0,1]^4$ represents the normalized motor commands. The equations of motion are given by:

$$\dot{\mathbf{p}} = \mathbf{v}, \qquad \dot{\mathbf{v}} = g\mathbf{e}_3 + R(\lambda)\mathbf{F} \tag{2}$$

$$\dot{\lambda} = Q(\lambda)\boldsymbol{\Omega}, \qquad \dot{\boldsymbol{\Omega}} = \mathbf{M} \tag{3}$$

$$\dot{\omega}_i = \frac{\omega_{ci} - \omega_i}{\tau} \tag{4}$$

Where $R(\lambda)$ is the rotation matrix and $Q(\lambda)$ the transformation matrices from body rates to Euler angle derivatives. The motor steady-state response is modelled as:

$$\omega_{ci} = (\omega_{\max} - \omega_{\min})\sqrt{k_l u_i^2 + (1-k_l)u_i} + \omega_{\min} \tag{5}$$

The specific thrust force F and moment M acting on the body are given by:

$$\mathbf{F} = \left[-\sum_{i=1}^{4} k_x v_x^B \omega_i, \; -\sum_{i=1}^{4} k_y v_y^B \omega_i, \; -\sum_{i=1}^{4} k_\omega \omega_i^2 \right]^T \tag{6}$$

$$\mathbf{M} = \begin{bmatrix} -k_{p1}\omega_1^2 - k_{p2}\omega_2^2 + k_{p3}\omega_3^2 + k_{p4}\omega_4^2 \\ -k_{q1}\omega_1^2 + k_{q2}\omega_2^2 - k_{q3}\omega_3^2 + k_{q4}\omega_4^2 \\ -k_{r1}\omega_1 + k_{r2}\omega_2 + k_{r3}\omega_3 - k_{r4}\omega_4 + \\ \ldots - k_{r5}\omega_1 + k_{r6}\omega_2 + k_{r7}\omega_3 - k_{r8}\omega_4 \end{bmatrix} \tag{7}$$

All parameters k_*, $\omega_{\min}$, and $\omega_{\max}$ are identified for the 5-inch drone following [13].

2.2 Reinforcement Learning Task

The control objective is to autonomously navigate a 5-inch racing quadcopter through a sequence of 7 square gates arranged in a figure-eight pattern. The environment state is defined as in [14]:

$$\mathbf{x}_{\text{obs}} = [p^{g_i}, v^{g_i}, \lambda^{g_i}, \Omega, \omega, p_{g_{i+1}}^{g_i}, \psi_{g_{i+1}}^{g_i}]^T \tag{8}$$

where the superscript g^i denotes the reference frame of the i-th gate. The position and orientation of the next gate are given by $p_{g_{i+1}}$ and $\psi_{g_{i+1}}$ respectively.

The reward function is adapted from [14][13] and encourages forward progress while penalizing high angular velocity and collisions:

$$r_k = \begin{cases} -10, & \text{if collision} \\ |p_{k-1} - p_{g_k}| - |p_k - p_{g_k}| - c|\Omega|, & \text{otherwise} \end{cases}$$

Here, the subscript k represents the current timestep. The scalar $c = 0.001$ controls the penalty on angular velocity magnitude. A collision is triggered either by ground contact or when the quadcopter exits the predefined bounding box (10 m x 10 m x 7 m). Additionally, if the drone crosses a gate plane without passing through the designated 1.5 m x 1.5 m gate (i.e. missed the gate), the episode is also terminated. This reward encourages stable, accurate, and efficient gate traversal.

2.3 Leaky Integrate-and-Fire Neuron

The key element of the spiking network is the Leaky Integrate-and-Fire (LIF) neuron, a simplified yet biologically inspired model compatible with gradient-based learning frameworks, implemented using the snnTorch Python library [8]. LIF neurons accumulate input over each time step and emit discrete spikes when their membrane potential exceeds a threshold. After spiking, the LIF neuron is soft-reset, subtracting the threshold potential from the current membrane potential. The evolution of the membrane potential $U[t]$ is governed by the first-order differential equation:

$$U[t+1] = \beta U[t] + I[t] - U_{\text{th}} S[t] \tag{9}$$

$$S[t] = H(U[t] - U_{\text{th}}) \tag{10}$$

Here, $U[t]$ is the membrane potential, $\beta \in [0,1)$ is the decay constant, $I[t]$ is the input current, U_{th} is the firing threshold, and $H(\cdot)$ denotes the Heaviside step function. The output spike $S[t] \in \{0,1\}$, soft-resets the membrane potential by subtracting the threshold potential, denoted by the last term in Equation 9. Our implementation uses LIF neurons with a default threshold of $U_{th} = 1$ and $\beta = 0.999$. The high β minimizes the leak of membrane potential, and encourages the LIF neurons to fire earlier and more frequently. This results in a stronger gradient flow during training which promotes learning via error backpropagation [8]. This is also exploited by rate decoding the output, as explained further in subsection 2.4.

2.4 Implementation of Spiking Neural Networks

SNNs introduce the following two key challenges when applied to gradient-based learning.

1. Spiking neurons, such as the LIF model in Equation 10, show inherent time-dependency, maintaining an internal membrane potential that spans multiple time steps. This stateful property enables temporal integration but complicates the design of standard feed-forward architectures and training methods.

2. The binary, non-differentiable nature of spike outputs, prevents the use of conventional gradient-based optimization without modification.

To address the first challenge of handling the temporal dynamics, we process only the current environment state at each forward pass, without explicitly modelling dependencies across timesteps. This design choice follows the insight of the approach by Ferede [14], where a purely feed-forward ANN was shown to achieve effective control despite the absence of a recurrent structure. This approach allows us to leverage the temporal characteristic of LIF neurons for the decoding of the output. We address the problem by using a rate encoding approach common in ANN-to-SNN conversion methods [15, 16], which we shall refer to as *cycling*. With this strategy each observation state $\mathbf{x} = \mathbf{x}_{\text{obs}}[t] \in \mathbb{R}^{20}$ at timestep t is held constant and repeatedly propagated for a fixed number of steps C (i.e. *cycles*) through the network, consisting of three fully connected layers of LIF neurons. During this process, each LIF neuron can accumulate membrane potential and spike across several cycles. Each observation state $\mathbf{x}$ is input using standard current injection, where the float values are directly passed to the LIF neurons. A direct drawback of this cycling strategy is that we do not explicitly use the temporal modelling capabilities of SNNs, therefore operating without recurrence. Superscripts $(l) \in \{1, 2, 3\}$ indicate the corresponding network layer and at each cycle c, the spiking activations are computed as:

$$\mathbf{s}^{(1)}[c] = \text{LIF}^{(1)}(W^{(1)}\mathbf{x}, U^{(1)}[c-1]) \tag{11}$$

$$\mathbf{s}^{(2)}[c] = \text{LIF}^{(2)}(W^{(2)}\mathbf{s}^{(1)}[c], U^{(2)}[c-1]) \tag{12}$$

$$\mathbf{s}^{(3)}[c] = \text{LIF}^{(3)}(W^{(3)}\mathbf{s}^{(2)}[c], U^{(3)}[c-1]) \tag{13}$$

The output spikes from the final layer are averaged using rate decoding, as motivated in subsection 2.5, whereby the average firing rate of each neuron over the amount of cycles C is computed as shown below [17]:

$$\bar{\mathbf{S}} = \frac{1}{C}\sum_{c=1}^{C} \mathbf{s}^{(3)}[c] \tag{14}$$

The decoded spike-rate vector $\bar{\mathbf{S}} \in [0, 1]^N$, where N denotes the number of neurons in the layer, is then passed to a fully connected linear output layer to produce the continuous motor commands $\hat{\mathbf{u}} \in [-1, 1]^4$ which are then mapped to actual normalized motor commands $\mathbf{u} \in [0, 1]^4$:

$$\hat{\mathbf{u}} = W_{\text{out}}\bar{\mathbf{S}} + \mathbf{b}_{\text{out}} \tag{15}$$

An important consequence of rate decoding is that, since output spikes are binary in each cycle, decoding over C cycles produces a quantized latent output.

$$s^{(c)} \in \{0, 1\}^N, \qquad \bar{S} \in \left\{0, \frac{1}{C}, \frac{2}{C}, \ldots, 1\right\}^N \tag{16}$$

The resolution is $\frac{1}{C}$ with exactly $C + 1$ possible values for the average firing rate of each LIF neuron (e.g., C=1 : $\{0, 1\}$; C=2 : $\{0, 0.5, 1\}$; C=3 : $\{0, \frac{1}{3}, \frac{2}{3}, 1\}$). This discretization precedes the continuous action mapping and limits the latent output's representational resolution to $\frac{1}{C}$.

The second challenge, the non-differentiability of spikes, is caused by the discontinuous nature of the Heaviside step function $H(\cdot)$. As its derivative $\delta(\cdot) \in \{0, \infty\}$ evaluates to zero almost everywhere and diverges to infinity at the threshold, it prevents the use of exact gradient-based optimization. To solve this issue, we adopt a surrogate gradient method, wherein the true derivative of $H(\cdot)$ is replaced by a smooth, differentiable approximation during the backward pass as in [8]. Specifically, we use the derivative of a shifted arctangent function as the surrogate, adapted from [7]:

$$\frac{\delta H}{\delta U} \approx \frac{1}{\pi\left(1 + (\pi U)^2\right)} \tag{17}$$

This approach preserves the ability to train the network through standard back propagation techniques while leaving the discrete spiking behaviour untouched in the forward pass.

2.5 Spiking Neural Network Architecture

The choice to specifically use spike-rate decoding is grounded in well-established findings from ANN-to-SNN conversion literature. Numerous studies have shown that under rate encoding, the firing rate of LIF neurons closely approximates the behaviour of the ReLU activation function [15, 17, 18]. This functional similarity has made spike-rate decoding a natural and effective choice in many prior works as it facilitates the direct transfer of architectures and training methods from ANNs to SNNs with minimal modification [19, 20, 21]. This similarity allows us to retain the structure of the baseline ANN used in [13], and replace the ReLU activation function in the original network with LIF neurons. Whereas the SNN cycles each input state and uses rate decoding, while the ANN does not, the architecture of the SNN, including the number of layers and hidden units, is otherwise unchanged from the ANN.

The resulting architecture comprises three fully connected layers of LIF neurons, each containing 64 units as shown in Figure 2. The identical architecture also ensures that subsequent benchmarking between the ANN and SNN is both fair and meaningful.

2.6 Training procedure and randomization

We train the SNN policy using the PPO algorithm [22], implemented via the Stable-Baselines3 reinforcement learning library [23]. The training setup integrates the quadcopter dynamics model described in subsection 2.1, the reward function and environment defined in subsection 2.2, and the SNN-specific adaptations introduced in subsection 2.4.

In PPO, the policy is represented by two *separate* neural networks: the actor and the value network. In our implementation, both networks share the same architecture, differing only in the dimensionality of their outputs (actor: $\in \mathbb{R}^4$, value: $\in \mathbb{R}^1$). The actor network maps a given state $\mathbf{x}$ to the parameters of a diagonal Gaussian distribution, producing a mean vector $\mu_\theta(x) \in \mathbb{R}^d$ and a log standard deviation vector $\log \sigma_\theta \in \mathbb{R}^d$, where d denotes the dimensionality of the action space (i.e. 4). Actions can

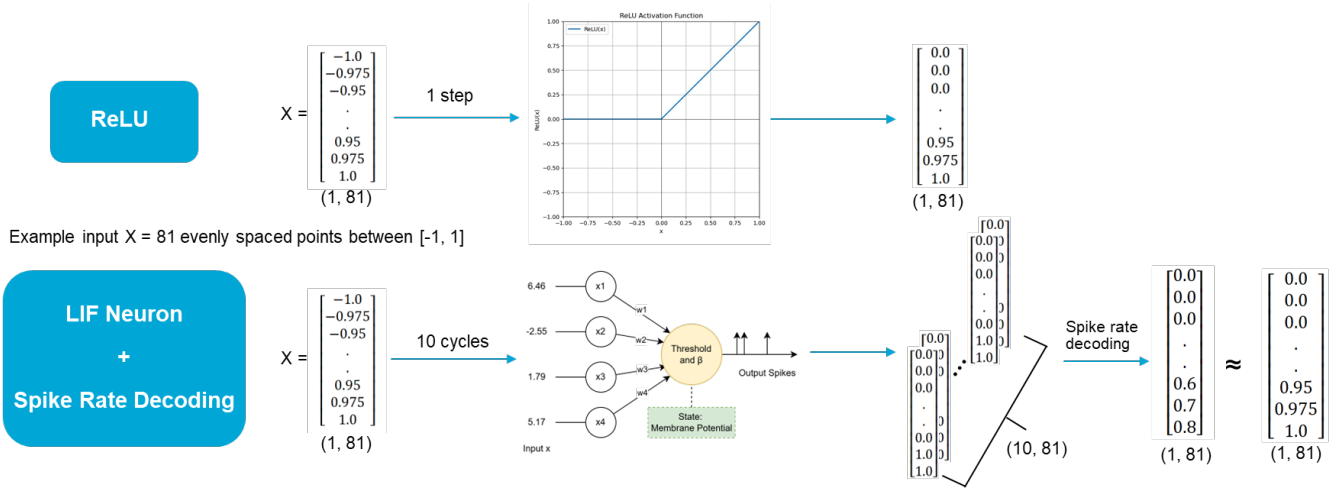


Figure 1: Comparative schematic showing how a LIF neuron with rate decoding over several cycles is similar to the ReLU activation function for an example input X of 81 evenly spaced points between [-1, 1].

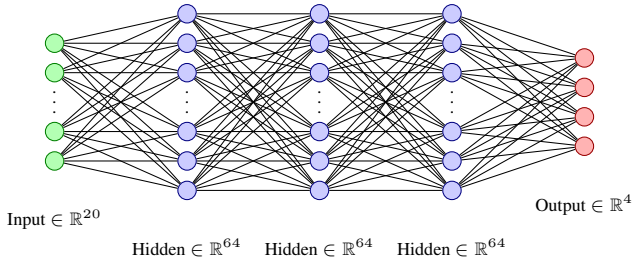


Figure 2: Schematic of the spiking neural **actor** network architecture. Each hidden layer contains 64 neurons. The value network is identical with the exception of the size of the output dimension which in contrast is 1

then be obtained either deterministically, by taking the mean, or stochastically, by sampling from the distribution. It is important to note that for training, samples are taken stochastically:

$$a \sim \mathcal{N}\big(\mu_\theta(x), \mathrm{diag}(\sigma_\theta^2)\big), \tag{18}$$

where $a \in \mathbb{R}^4$ is the action vector and $\mathrm{diag}(\sigma_\theta^2)$ denotes a diagonal covariance matrix with entries σ_θ^2. The value network, in contrast, outputs a single scalar $V_\phi(s) \in \mathbb{R}$, which estimates the expected return from state $\mathbf{x}$. Here, θ and ϕ represent the trainable parameters of the actor and value networks, respectively.

The training procedure with PPO is accelerated by running 100 parallel environments, each simulating a single drone. This enables the agent to collect experience from multiple simulations simultaneously, significantly reducing the time required for each policy update for PPO. Episodes have a maximum duration of 12 seconds to allow the drone to fly multiple laps of the track. Simulating at a control frequency of $100\,Hz$, the 12 seconds correspond to 1200 simulation steps. We use a discount factor of $\gamma = 0.999$ to prioritize long-term rewards over immediate rewards at a given step. Lastly, a default learning rate of $\eta = 3 \times 10^{-4}$ is used. The training parameters are identical to those used to train the baseline ANN in [13], ensuring a fair and consistent comparison. To investigate the effect of the number of cycles on SNN performance, we train a separate SNN model for each cycle count in the set $\{2,\ 3,\ 4,\ 5,\ 8,\ 10\}$.

A common struggle for quadcopters is the transfer of behaviour and performance to real-life flight tests, which makes it crucial to design robust policies in simulation. To overcome this sim-to-real gap and improve robustness both in simulation and real-life, we apply domain randomization with a 30% uniform scaling on all physical parameters during training. Specifically, each parameter θ is drawn from $\theta \sim \mathcal{U}(0.7\theta_0, 1.3\theta_0)$, where θ_0 is the nominal value for the 5-inch drone identified in [13].

In addition to this, randomized initial conditions are applied at the start of each episode, enhancing the drone's ability to navigate toward the next target gate from a broader range of states. The drone's position is uniformly sampled as $x_0,\ y_0 \sim \mathcal{U}(-5,\ 5)\,\mathrm{m}$ and $z_0 \sim \mathcal{U}(-3,\ 0)\,\mathrm{m}$. Linear velocities are initialized from $v_x,\ v_y,\ v_z \sim \mathcal{U}(-0.5,\ 0.5)\,\mathrm{m/s}$. Orientation is randomized over roll and pitch angles $\phi,\ \theta \sim \mathcal{U}(-\frac{\pi}{9},\ \frac{\pi}{9})$ and yaw $\psi \sim \mathcal{U}(-\pi,\ \pi)$ radians. Body angular rates are sampled as $p,\ q,\ r \sim \mathcal{U}(-0.1,\ 0.1)\,\mathrm{rad/s}$, and each motor is initialized with an angular velocity $w_i \sim \mathcal{U}(-1,\ 1)\,\mathrm{rad/s}$ for $i \in \{1,\ 2,\ 3,\ 4\}$.

3 Experimental Setup

We adopt the same experimental platform as in [13], using a 5-inch quadcopter configured for indoor autonomous flight. The control firmware is based on INDIflight[1], a fork of Betaflight, which runs on a STM32H743 microcontroller. State estimation is handled onboard using an Extended Kalman Filter (EKF) that fuses inertial data from a TDK InvenSense ICM-42688-P IMU with external position and attitude measurements from an OptiTrack motion capture system. The state is then used as input to the SNN which outputs the corresponding motor commands. All experiments are conducted in the CyberZoo flight arena at TU Delft, a $10\,\mathrm{m} \times 10\,\mathrm{m} \times 7\,\mathrm{m}$ indoor space equipped for autonomous drone testing.

[1] https://github.com/tudelft/indiflight

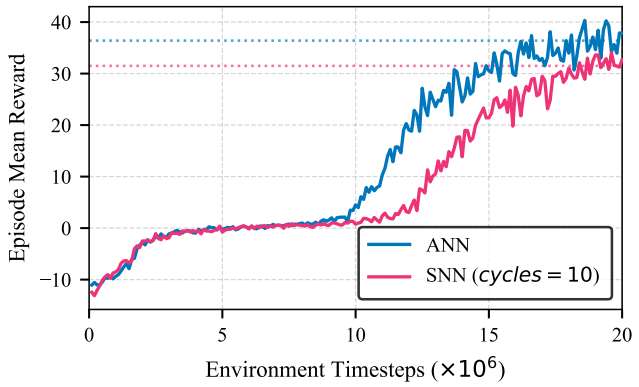


Figure 3: Training of an ANN compared to a SNN policy with 10 cycles over 20 million timesteps

Metric	Unsuccessful (Crash)	Successful
ANN		
Mean Reward	2.00 ± 14.52	58.23 ± 8.36
Max Reward	66.87	77.13
Crash Rate (%)	67.30	0.0
Mean Episode Length (steps)	206.0	1200.0
$\bar{v}$ (m/s)	6.39 ± 3.17	7.43 ± 2.00
v_{max} (m/s)	**28.67**	15.16
SNN (cycles=10)		
Mean Reward	-2.75 ± 10.32	$\mathbf{64.93 \pm 8.59}$
Max Reward	58.87	**87.68**
Crash Rate (%)	**44.80**	0.0
Mean Episode Length (steps)	99.2	1200.0
$\bar{v}$ (m/s)	5.56 ± 3.45	$\mathbf{7.95 \pm 1.95}$
v_{max} (m/s)	20.49	**15.43**

Table 1: **Simulation** performance comparison of ANN and SNN policies over 1000 simulated episodes. Metrics are shown for unsuccessful (crash) and successful (non-crash) episodes separately.

4 Results

4.1 Baseline ANN vs. SNN Performance

Using the procedure and parameters described in subsection 2.6, we train the ANN and SNN policies for a maximum of 20 million timesteps. Both models share an identical architecture, and are trained under the same conditions. Figure 3 shows the mean episode reward over the 20 million timesteps and the final converged value, taken as the average reward of the last 10% of the timesteps. The SNN model used in this comparison uses a cycle count of 10, selected based on a preliminary analysis across multiple cycle values. That analysis revealed that a cycle count of 10 offered a favourable trade-off between training time and performance, compared to even higher cycle models, achieving quite comparable reward albeit at a much higher training time compared to the ANN. It should be noted that the presented curves represent a single training run for each model. In reinforcement learning it is standard practice to evaluate average performance across multiple training runs due to the strong influence of randomness in the training process[24]. However, this was not feasible in the present study because of the limited computational resources available and the very long training times of some models. As a result, the analysis is based on single representative runs for both the ANN and SNN in this and subsequent comparisons.

Although the SNN demonstrates slower learning and converges to a slightly lower average reward (31.5) compared to the ANN baseline (36.4), the overall training reward convergence remains comparable. This indicates that the spiking policy is capable of effectively learning the task in simulation, with minimal degradation. However, a significant drawback of the cycled SNN becomes immediately apparent: while the ANN completed training in 16.17 minutes, the SNN required 5.98 hours to train on a standard consumer-grade laptop, with training time being proportional to the cycle count of the model.

To further analyze flight performance of the spiking policy in simulation, the SNN and ANN controllers were both trained with PPO to achieve a comparable task reward of $r = 50$ before evaluation. The ANN was trained until convergence to a mean episode reward of $r = 50.7$, while the SNN until a similar reward of $r = 50.82$.

Although both models reached the target reward after a comparable number of timesteps (ANN: 40.3×10^6; SNN: 41.1×10^6), the SNN required 10.4 hours of training versus 33 min for the ANN. Following training, each policy was evaluated over 1,000 simulation episodes with the reward and velocity performance metrics being shown in Table 1. Metrics are calculated for successful and unsuccessful runs separately to highlight the performance differences between the models.

Despite comparable training reward, the SNN delivers a higher mean reward of 64.93 on successful episodes than the ANN with a reward of 58.23. The SNN exhibits not only better performance but also a significantly lower crash rate of 44.80% than the ANN with a crash rate of 67.87%, indicating a more robust policy, while achieving a higher average velocity. The high crash rate for both are largely a consequence of the randomized initial conditions used for each episode as detailed in subsection 2.6. In many occasions this leads to starting conditions that are too extreme for the drone to recover from before the first gates. This is supported by the statistics for unsuccessful episodes: both controllers accumulate little reward (SNN: -2.75, ANN: 2.00) and terminate well before the 1,200-step horizon (SNN: 99.2 steps, ANN: 206.0 steps). Intriguingly, the ANN shows both a higher crash rate and longer failed episodes, suggesting greater sensitivity to poor initialisations, often surviving long enough to accumulate some reward yet ultimately failing later in the lap. By contrast, the SNN tends to either recover and complete the episode or fail quickly when initial conditions are unrecoverable, aligning with its lower overall crash rate.

The higher SNN performance is confirmed with real flight tests. Both policies were flown for three identical runs of 12 seconds each, equivalent to one fully charged battery. As opposed to simulation where start position, velocity and orientation are randomized, the real tests are initialized from a defined hovering start point, 1 meter from the ground and 1 meter from the top left gate in the track.

Furthermore for the evaluation of the policies in real-flight tests, the actions of the policy were sampled deterministically

	ANN				SNN (cycles=10)			
Metric	T1	T2	T3	$\bar{\mathbf{T}}$	T1	T2	T3	$\bar{\mathbf{T}}$
Reward	59.38	59.93	60.00	**59.77**	70.95	70.99	69.95	**70.63**
$\bar{v}$ (m/s)	6.91	7.03	7.03	**6.99**	7.87	7.99	7.95	**7.94**
Gates	40	41	41	**40.67**	46	47	46	**46.33**

Table 2: **Real flight** performance metrics over 3 trials of ANN and SNN (10-cycle) policies in real test flights.

from the action distribution to ensure more consistent behaviour and to draw more accurate conclusions about the performance of the flight data. Both networks were configured with a target control update frequency of $f_{des}^{(c)} = 1000Hz$. The following Table 2 presents the performance metrics for both models. An important note on the results for the real flight tests is that the observed control update frequency during flight was lower than the desired frequency due to onboard hardware limitations. The ANN reached $f_{obs}^{c} = 895.07Hz$ for its best trial while the SNN, limited by its higher cycle-dependent execution time, achieved only $f_{obs}^{c} = 356.86Hz$. This is caused by the increased execution time of the cycled SNN, which prevents new output states from being computed at the desired rate $f_{des}^{(c)}$ and represents the primary disadvantage of such models when deployed on real hardware.

The trajectory comparison of the best performing trial of the ANN and SNN policies in Figure 4 highlights distinct behavioural differences. While the ANN follows a relatively smooth flight path with a roughly constant velocity of 7.03 m/s, The SNN policy demonstrates a more dynamic control strategy, reaching a higher average velocity of 7.99 m/s. Notably, the SNN shows a greater variation in speed, accelerating sharply through the central gate and hence decelerating more aggressively to navigate the outermost turns. This results in sharper cornering and frequent changes in acceleration. The SNN policy appears to prioritize aggressively accelerating during the straighter sections of the track, trading off smoothness for speed, resulting in the SNN outperforming the ANN in both episode reward (70.99 vs. 60.00) and gates passed (47 vs. 41).

4.2 Analysis of SNN Performance

To better understand why the SNN achieves superior performance compared to the ANN, we analyze both the motor outputs recorded during real flight tests and two key training metrics from PPO: the entropy loss and the clip fraction.

The differences in control strategy are evident in the distribution of output motor RPMs from real flight tests. As shown in Figure 5, the SNN consistently generates higher output RPMs than the ANN, with approximately 17% of outputs saturated at the maximum normalized RPM of 1.0, and a higher number of zero-RPM outputs as well. It indicates a more extreme actuation pattern in the SNN's control policy, with the motors being used at a higher capacity than the ANN.

To investigate the source of this behaviour, we further compare the training metrics of the ANN and SNN models previously trained in subsection 4.1. Another interesting distinction is observed during training when comparing the entropy loss terms of both networks. In PPO, entropy is explicitly used as a term in the loss function to encourage exploration: a higher entropy

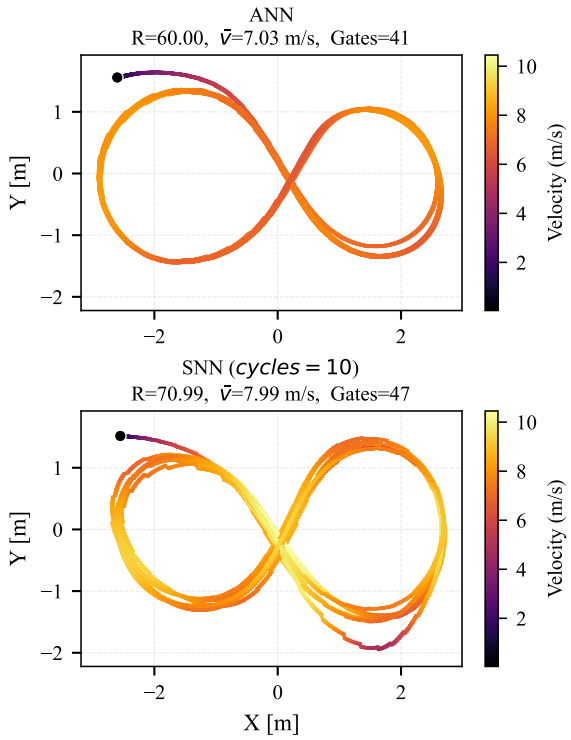


Figure 4: **Real flight** trajectory comparison between the best trials of the ANN vs the SNN with 10 cycles. Videos for the real flights are available at: `https://drive.google.com/drive/folders/1uGINGe71wu0Hrh0_ZBDAHMIO-s6B5qd3?usp=drive_link`

indicates that the policy maintains greater uncertainty over its action distribution, with lower entropy corresponding to more deterministic policies [22]. While the original PPO formulation maximizes entropy to promote exploration, the implementation in Stable-Baselines3 [23] adopts the convention of minimizing the negative entropy loss so that all terms contribute to a minimization objective.

As shown in Figure 6, although similar in the beginning, the SNN maintains a significantly lower negative entropy loss onwards from timestep 10×10^6 compared to the ANN, showing the SNN sustains higher entropy and thus stronger exploration. By contrast, the ANN continuously increases its negative entropy loss, reflecting a continuous decline in exploration and earlier convergence to a specific policy.

The clip fraction metric provides further insight into the differing training behaviour of the two networks. In PPO, the clipping mechanism limits the size of policy updates by constraining the probability ratio (measure of similarity) between new and old policies, with the clip fraction quantifying the proportion of updates that are affected by this constraint [22]. High clip fractions therefore indicate that the policy is frequently attempting to change more aggressively than the clipping threshold allows. As shown in Figure 6, the SNN exhibits a consistently higher clip fraction, reaching fractions above 0.3, while the ANN remains around a fraction of 0.1. This suggests that the SNN undergoes more larger policy shifts more frequently as a larger proportion of updates are clipped to the maximum allowed threshold.

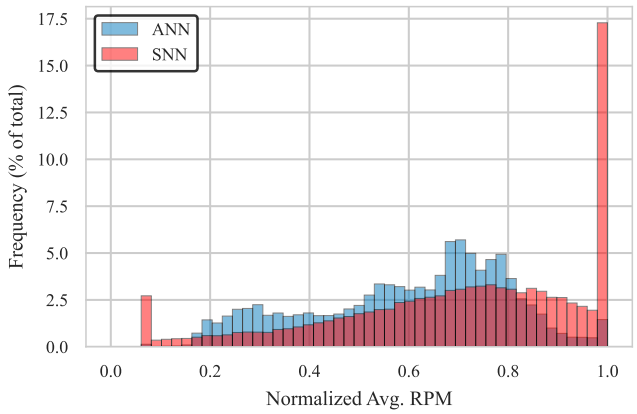


Figure 5: **Real flight** histogram of normalized output RPMs for the ANN and SNN over all three trials

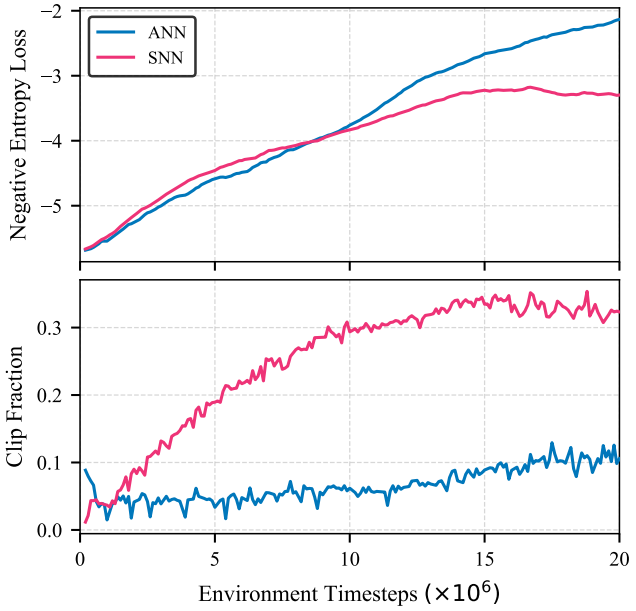


Figure 6: Negative entropy loss and clip fraction for baseline ANN and SNN models during training over 20 million timesteps

A hypothesis which can account for these observations and the reason for the higher performance of the SNN, is rooted in the combined effect of using a surrogate gradient with rate decoding. The derivative of the arctangent gradient, as given in subsection 2.4 is specifically highest in a narrow band around the membrane threshold. This concentrates the gradient flow around points where the membrane potential approaches or crosses the threshold [25]. In practice, this means that neurons that fire more frequently therefore produce larger gradient signals, biasing learning toward higher firing rates [26]. For high-speed quadcopter control, persistently higher motor commands can yield greater thrust and acceleration, enabling higher speeds and thus faster gate transitions, consistent with the increased velocities and rewards observed in Figure 4. Moreover, with rate decoding over C cycles, the average firing rates are quantized to values $\{0, \frac{1}{C}, \ldots, 1\}$ with a resolution of $\frac{1}{C}$ as noted in subsection 2.4. An upward firing-rate bias during training can shift this empirical distribution of the firing rates $\bar{\mathbf{S}}$ toward the upper quantization levels, which, after the action layer maps these latent firing rates to motor commands as in Figure 5, can yield higher and more frequently saturated output RPMs.

Importantly, the same mechanism also explains the SNN's higher exploration in training: elevated firing rates keep neurons near threshold, yielding stronger surrogate gradients and thus larger policy updates. In addition, the quantization of the rate decoding causes a many-to-one mapping of the output with respect to the input, so small changes in $\mathbf{x}$ or the weights W often leave $\bar{\mathbf{S}}$ unchanged because of the coarse resolution. As a consequence, the final action output mean $\mu_\theta(\mathbf{x})$ and the realized RPMs are also unchanged. To remain effective under this mapping, the policy learns to increase its action variance σ_θ^2 via the trainable parameters θ, so samples around $\mu_\theta(\mathbf{x})$ still cross quantization thresholds and produce distinct motor output RPMs, unlike before. In the training parameters this manifests as higher policy entropy (in SB3, lower negative-entropy loss), which is proportional to the variance σ_θ^2, and a larger clip fraction, consistent with increased exploration.

While this hypothesis is consistent with the observed behaviour, establishing causality requires further research through systematic experiments and empirical activity analysis of the neuron activations during training and evaluation. Specifically, (i) Analyze the layer-wise LIF activations, membrane-potentials, resulting latent firing rates during training and evaluation; (ii) quantify how the surrogate-gradient design and the number of cycles C affect the distribution of the action-layer outputs $\boldsymbol{\mu}_\theta(\mathbf{x})$ conditional on the quantized latent firing rates $\bar{\mathbf{S}}$ and inputs $\mathbf{x}$; and (iii) assess these relationships using appropriate statistical tests while controlling for confounders such as initialization, input distributions, learning rate, and other hyper-parameters.

4.3 Effect of Cycles on SNN Flight Performance

To assess the effect of the cycle count on SNN training and flight performance, six models were trained using cycle counts: $\{2, 3, 4, 5, 8, 10\}$. Each model was trained for a total of 150 million timesteps, a value selected with the goal of reaching convergence in mean episode reward for all models, while maintaining feasible training time. To highlight the reward convergence, an exponential moving average (EMA) with a smoothing factor of ($\alpha = 0.1$) was applied to the mean episode reward. The converged reward is calculated as the average reward of the last 10% of the training timesteps. Figure 7 shows the evolution of the mean episode reward for the six SNN models and the converged reward

As the number of cycles increases, the converged episode reward improves consistently, indicating a clear performance gain from averaging the spiking output over a higher cycle count. The lowest-performing model, using only 2 cycles, converged to a mean reward of 44.87, while the best-performing model, with 10 cycles, reached a reward of 62.57. Interestingly, 3 and 4 cycles yield comparable rewards (51.05 and 50.56), while 5, 8, and 10 cycles result in increasingly higher performance (55.35, 58.64, and 62.57, respectively). This demonstrates that a higher cycle count provides finer resolution in spike-rate decoding, resulting in smoother motor commands, which in turn improves

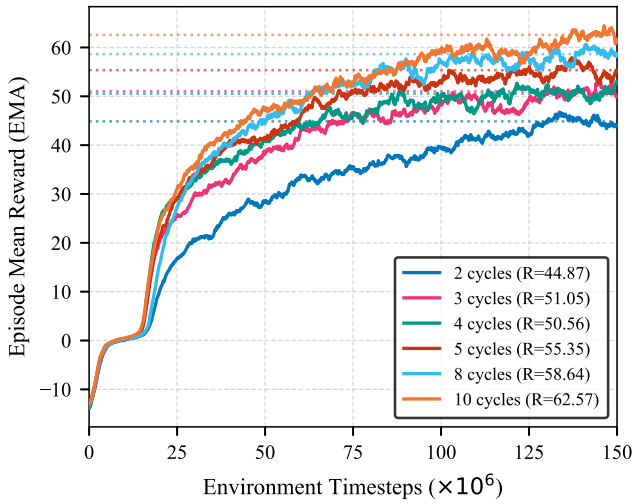


Figure 7: Training runs for all six SNN models, showing the EMA of the mean episode reward over 150 million timesteps.

Cycles	$f_{des}^{(c)}$ (Hz)	$f_{obs}^{(c)}$ (Hz)	τ (iter/s)	Rew.	$\bar{v}$ (m/s)	Gates
2	500	403.08	806.17	65.04	7.14	42.67
3	333	268.11	804.32	65.45	7.80	44.33
4	250	204.31	817.25	68.43	8.01	44.67
5	200	167.80	838.98	70.27	8.25	50.00
8	125	105.95	847.63	70.76	8.09	45.67
10	100	85.16	851.60	69.81	8.03	47.00

Table 3: **Real flight** average performance metrics for **inference-constrained** SNN flight tests across different cycle counts for three trials. τ denotes the amount of forward passes per second (inference rate)

learning and performance. However, in addition to the significantly longer training time compared to the baseline ANN, a second drawback of cycled SNNs becomes evident during deployment: increased execution time. This latency is especially a challenge in real-time deployment for models with higher cycle counts.

To investigate the real-time deployment, each model was flown in three real-world trials using specific combinations of cycle count and desired control update frequency $f_{des}^{(c)}$. The goal was to compare the models under two conditions: constant computational load and constant observed control frequency. For the constant computational load condition, models and parameters were matched based on a similar inference rate τ, defined as the product of the observed control frequency and the cycle count. The average performance metrics across trials are summarized in Table 3.

Despite the hardware limitations on the observed frequency, the achieved inference rate for the combinations remains relatively stable ranging from 806 to 852 inferences per second. Table 3 shows that flight performance generally improves with increasing cycle count, peaking at cycles=8 with the highest average reward of 70.76. The best gate completion performance, however, is achieved by the 5-cycle model, which passes 50.00

Cycles	$f_{des}^{(c)}$ (Hz)	τ (iter/s)	Rew.	Progress Rew.	Rate Penalty	Gates
2	500	806.17	65.04	73.30	-8.26	42.67
3	333	804.32	65.45	74.41	-8.96	44.33
4	250	817.25	68.43	77.45	-9.02	44.67
5	200	838.98	70.27	80.66	-10.39	50.0
8	125	847.63	70.76	79.74	-8.98	45.67
10	100	851.60	69.81	79.46	-9.65	47.0

Table 4: **Real flight** average reward decomposition metrics for **inference-constrained** SNN flight tests from Table 3. Reward is decomposed as *Progress* + *Rate Penalty*

gates with a higher average velocity of 8.25 m/s and a reward of 70.27. Despite better objective performance in terms of gates passed, the lower reward of the 5 cycle model is largely attributed to the reward function's rate penalty term, as shown in Table 4, indicating the reward function can still be optimized for time-optimal trajectories.

Lower cycle counts (e.g., 2 and 3) result in reduced rewards (65.04 and 65.45) and fewer gates passed (42.67 and 44.33), due to lower output resolution from the spike-rate decoding. Interestingly, while the 10-cycle model maintains a high reward (69.81), it shows no further improvement in velocity or gate count, indicating diminishing returns. Overall, the results suggest that moderate cycle counts (5–8) offer the best trade-off between model performance and control responsiveness and network execution time, with 8 cycles yielding the highest reward and 5 cycles providing the most consistent all-around performance.

Similarly, Table 5 presents the results under similar observation frequency. Again, the 8-cycle model achieves the highest reward (70.01) with a high average velocity (8.30 m/s) and gates passed (47.67). The 5-cycle model again performs best in terms of velocity (8.36 m/s) and gate count (50.00), with a competitive reward (69.81). However, its observed frequency (337.20 Hz) was unexpectedly higher than in previous experiments, likely due to inconsistencies in the flight controller's task scheduling, potentially giving it an advantage in responsiveness.

Lower-frequency models (e.g., 3 and 4 cycles at 333 Hz) again show degraded performance, with lower rewards (65.45 and 64.37) and fewer gates completed. The 10-cycle model performs similarly to the unconstrained setting, reinforcing the observation that excessive execution time from higher cycles can reduce responsiveness despite improved spike-rate resolution. The trajectories for some of the most significant trials are shown in Figure 8.

The trajectories **(a)** and **(d)** in Figure 8 show the two highest-performing models, both achieving a reward of 71.52. Like the 10-cycle SNN in subsection 4.1, they show aggressive acceleration, sharp turns, and peak speeds on straight segments, reinforcing our earlier observation. In contrast, the 4-cycle **(b)** and 2-cycle **(e)** models achieved the lowest rewards. Interestingly, the 2-cycle model improved significantly its reward from 44.87 in simulation to 63.13 in real-world tests, displaying slower, smoother flight resembling the ANN baseline. Despite higher reward and velocity, the 4-cycle model passed fewer gates than the 2-cycle model. A similar mismatch appears for the 5-cycle

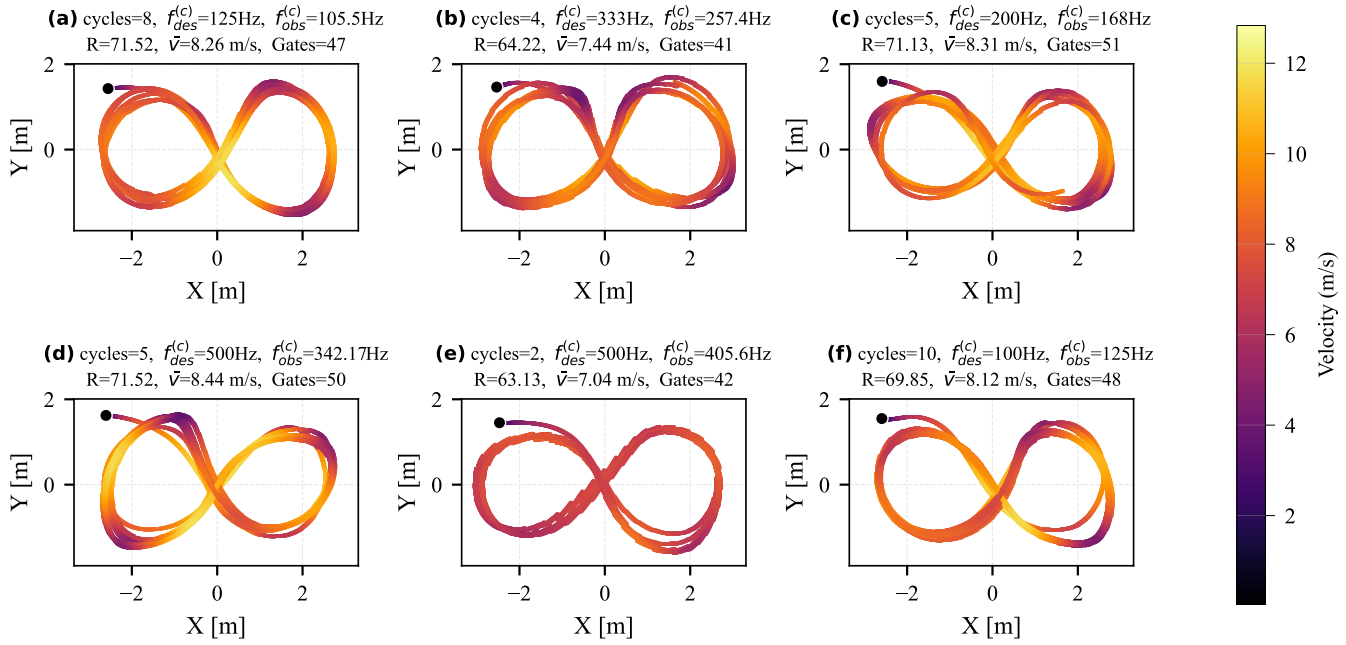


Figure 8: **Real flight** trajectory plots for significant trials, (a) and (d): highest reward, (b) and (e): second lowest and lowest reward, (c): highest gates passed, (f): highest cycle model. Videos for the flight test are available at: `https://drive.google.com/drive/folders/1uGINGe71wu0Hrh0_ZBDAHMIO-s6B5qd3?usp=drive_link`

Cycles	$f_{des}^{(c)}$ (Hz)	$f_{obs}^{(c)}$ (Hz)	τ (iter/s)	Rew.	$\bar{v}$ (m/s)	Gates
3	333	268.11	804.32	65.45	7.80	44.33
4	333	257.08	1028.32	64.37	7.65	41.67
5	500	337.20	1686.00	69.81	8.36	50.00
8	500	289.33	2314.67	70.01	8.30	47.67
10	500	261.07	2610.67	69.07	7.83	45.33

Table 5: **Real flight** averaged performance metrics for **frequency-constrained** SNN flight tests across different cycle counts for three trials.

model in trajectory **(c)**, which passed the most gates (51) but had a lower reward than the 8-cycle model, highlighting the need to refine the reward function to better reflect objective time-optimal performance.

To further illustrate the effect of cycle count on control fidelity, Figure 9 shows the distribution of normalized output RPMs for the 2-cycle and 10-cycle models used in subsection 4.3 during real-flight tests. The 2-cycle model produces a coarse output distribution, with many activations concentrated at the extremes near the minimum and maximum values. This indicates a lower effective resolution, as intermediate control signals are under-represented. In contrast, the 10-cycle model yields a smoother distribution with a broader spread across the mid-to-high ranges and a more gradual drop-off toward lower values. While both models show comparable activation density in the highest output bin, the 2-cycle model generates a substantially larger fraction of low values, which reduces expressiveness in motor control. This confirms that reducing the cycle count compromises output resolution and leads to a coarser control signal.

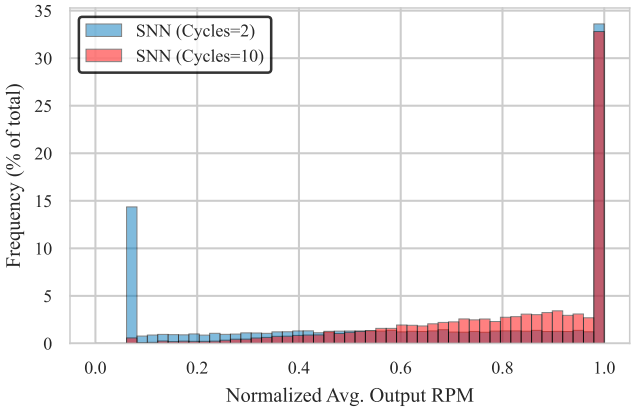


Figure 9: Distribution of normalized output RPMs for the 2-cycle and 10-cycle models. The 2-cycle model concentrates activations at the extreme low and high ends, indicating lower output resolution. The 10-cycle model produces a smoother distribution with more intermediate values, resulting in finer control signals.

5 Conclusion

This work presented the first successful application of a fully spiking actor-critic network trained with PPO for continuous quadcopter control. The SNN, implemented with LIF neurons and trained using surrogate gradients and spike-rate decoding, achieved performance superior to state-of-the-art ANNs in both simulated and real-world high-speed navigation tasks. Evaluation on the 5-inch racing quadcopter revealed that the SNN not only matched the ANN in control fidelity but also outperformed it in reward, robustness, and average velocity, despite its slower training and higher inference latency due to cycle-based spike integration. Analysis of the SNN further indicates that the superior performance of the SNN can be attributed to a bias towards higher firing rates introduced through the surrogate gradient and rate decoding mechanism, which in turn yields stronger motor actuation and sustained exploration during training. These effects likely enable the SNN to adopt more aggressive control strategies than the ANN, consistent with the higher velocities and rewards observed.

An extensive analysis of different cycle counts demonstrated a clear trade-off between temporal resolution and execution delay. While higher cycle counts yielded smoother motor outputs and improved control performance, they also imposed greater computational costs and reduced update frequencies on embedded hardware. The 8-cycle model attained the highest average reward, whereas the 5-cycle model had the best balance between reward, velocity, and gates passed, confirming that moderate cycle counts provide the most favourable trade-off for real-time applications.

Future work will focus on extending this approach to neuromorphic hardware platforms such as Intel Loihi or other event-driven processors to fully leverage the energy efficiency of spiking neurons. To enable efficient deployment, further investigation is needed into quantizing the network weights and activations and improving runtime efficiency. Additionally, we aim to explore how varying the network parameters and architecture impacts both control performance and computational cost. These directions are critical for designing scalable, lightweight SNN controllers for embedded applications in autonomous aerial vehicles.

References

[1] Robin Ferede, Guido de Croon, Christophe De Wagter, and Dario Izzo. End-to-end neural network based optimal quadcopter control. *Robotics and Autonomous Systems*, 172:104588, February 2024.

[2] Yunlong Song, Mats Steinweg, Elia Kaufmann, and Davide Scaramuzza. Autonomous Drone Racing with Deep Reinforcement Learning, August 2021. arXiv:2103.08624 [cs].

[3] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Girish Chinya, Yongqiang Cao, and et al. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018.

[4] Amirhossein Tavanaei, Mohammad Ghodrati, Saeed Reza Kheradpisheh, Timothée Masquelier, and Anthony Maida. Deep learning in spiking neural networks. *Neural Networks*, 111:47–63, 2019.

[5] Carlos Zamarreño-Ramos, Luis A Camuñas-Mesa, Jorge A Pérez-Carrasco, Timothée Masquelier, Teresa Serrano-Gotarredona, and Bernabé Linares-Barranco. On spike-timing-dependent-plasticity, memristive devices, and building a self-learning visual cortex. *Frontiers in Neuroscience*, 5:26, 2011.

[6] Luca Zanatta, Francesco Barchi, Simone Manoni, Silvia Tolu, Andrea Bartolini, and Andrea Acquaviva. Exploring spiking neural networks for deep reinforcement learning in robotic tasks. *Scientific Reports*, 14(1):30648, December 2024. Publisher: Nature Publishing Group.

[7] Wei Fang, Zhaofei Yu, Yanqi Chen, Timothee Masquelier, Tiejun Huang, and Yonghong Tian. Incorporating Learnable Membrane Time Constant to Enhance Learning of Spiking Neural Networks, August 2021. arXiv:2007.05785 [cs].

[8] Jason K Eshraghian, Max Ward, Emre Neftci, and et al. Training spiking neural networks using lessons from deep learning. In *Proceedings of the IEEE*, volume 111, pages 1016–1054, 2023.

[9] Cezary Kownacki, Slawomir Romaniuk, and Marcin Derlatka. Applying neural networks as direct controllers in position and trajectory tracking algorithms for holonomic UAVs. *Scientific Reports*, 15(1):12605, April 2025. Publisher: Nature Publishing Group.

[10] Federico Paredes-Vallés, Jesse Hagenaars, Julien Dupeyroux, Stein Stroobants, Yingfu Xu, and Guido de Croon. Fully neuromorphic vision and control for autonomous drone flight, March 2023. arXiv:2303.08778 [cs].

[11] Stein Stroobants, Christophe de Wagter, and Guido C. H. E. De Croon. Neuromorphic Attitude Estimation and Control, November 2024. Issue: arXiv:2411.13945 arXiv:2411.13945 [cs].

[12] Zijie Xu, Tong Bu, Zecheng Hao, Jianhao Ding, and Zhaofei Yu. Proxy Target: Bridging the Gap Between Discrete Spiking Neural Networks and Continuous Control, May 2025. arXiv:2505.24161 [cs].

[13] Robin Ferede, Till Blaha, Erin Lucassen, Christophe De Wagter, and Guido C. H. E. de Croon. One Net to Rule Them All: Domain Randomization in Quadcopter Racing Across Different Platforms, April 2025. arXiv:2504.21586 [cs].

[14] Robin Ferede, Christophe De Wagter, Dario Izzo, and Guido C. H. E. de Croon. End-to-end Reinforcement Learning for Time-Optimal Quadcopter Flight. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6172–6177, May 2024. arXiv:2311.16948 [cs].

[15] Yongqiang Cao, Yang Chen, and Deepak Khosla. Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition. *International Journal of Computer Vision*, 113(1):54–66, May 2015.

[16] Yangfan Hu, Qian Zheng, Xudong Jiang, and Gang Pan. Fast-SNN: Fast Spiking Neural Network by Converting Quantized ANN. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(12):14546–14562, December 2023.

[17] Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu. Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification. *Frontiers in Neuroscience*, 11, December 2017. Publisher: Frontiers.

[18] Sijia Lu and Feng Xu. Linear Leaky-Integrate-and-Fire Neuron Model Based Spiking Neural Networks and Its Mapping Relationship to Deep Neural Networks, May 2022. arXiv:2207.04889 [cs].

[19] Hanle Zheng, Yujie Wu, Lei Deng, Yifan Hu, and Guoqi Li. Going Deeper With Directly-Trained Larger Spiking Neural Networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(12):11062–11070, May 2021.

[20] Wei Fang, Zhaofei Yu, Yanqi Chen, Tiejun Huang, Timothée Masquelier, and Yonghong Tian. Deep Residual Learning in Spiking Neural Networks, January 2022. arXiv:2102.04159 [cs].

[21] Ziming Wang, Shuang Lian, Yuhao Zhang, Xiaoxin Cui, Rui Yan, and Huajin Tang. Towards Lossless ANN-SNN Conversion under Ultra-Low Latency with Dual-Phase Optimization. *IEEE Transactions on Neural Networks and Learning Systems*, 36(2):3189–3203, February 2025. arXiv:2205.07473 [cs].

[22] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, August 2017. arXiv:1707.06347 [cs].

[23] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[24] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G. Bellemare. Deep Reinforcement Learning at the Edge of the Statistical Precipice, January 2022. arXiv:2108.13264 [cs].

[25] Friedemann Zenke and Tim P. Vogels. The Remarkable Robustness of Surrogate Gradient Learning for Instilling Complex Function in Spiking Neural Networks. *Neural Computation*, 33(4):899–925, March 2021.

[26] Emre O. Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate Gradient Learning in Spiking Neural Networks, May 2019. arXiv:1901.09948 [cs].

Part II

Preliminary Analysis

3

Literature Review

This literature review establishes the foundation for investigating spiking neural networks as controllers for high-speed quadcopter flight. It begins in Section 3.1 with the historical development of neuromorphic computing and artificial neural networks, highlighting how biological inspiration shaped both fields and how their trajectories converge in modern spiking models. Within this context, spiking neurons introduced in Section 3.1.2 emerge as the unifying concept that links brain-inspired computation with machine learning practice.

The review then shifts from history to theory in Section 3.2, which covers the key components that define spiking networks. These include neuron models in Section 3.2.2 that capture integration and spiking dynamics, coding schemes in Section 3.2.3 that translate between continuous information and discrete events, and learning methods in Section 3.2.4 that enable SNNs to adapt to tasks. Supervised, unsupervised, and reinforcement paradigms are examined, with particular attention to mechanisms that align with neuromorphic constraints such as locality and energy efficiency. Reinforcement learning methods are further expanded in Section 3.2.5, where their relevance for continuous control tasks is detailed.

Finally, the review surveys applications of learned control and perception for quadcopters in Section 3.3. Work with artificial neural networks in Section 3.3.1 provides the current benchmark for high-speed autonomous flight, while emerging studies with spiking networks in Section 3.3.2 demonstrate how event-driven sensing and neuromorphic hardware can support energy-efficient autonomy in drones. This progression from historical background to theoretical principles and practical applications provides the basis for the research questions and methodology that follow in Chapter 4.

3.1. Historical Overview of Neuromorphic Computing and Artificial Neural Networks

This chapter details how biological inspiration shaped neuromorphic computing and how these developments connect to modern spiking neural networks. We begin with biomimicry as the conceptual root and then provide a brief overview of the evolution of neuromorphic hardware and computing, highlighting recurring principles such as event-driven computation, co-location of memory and compute, and sparse parallelism. Finally, we also provide an overview in the development of artificial neural networks and how advances regarding architectures, training methods, and tooling, have transferred into the domain of spiking neural networks, motivating the detailed spiking neural network theory in the next Section 3.2.

3.1.1. Biomimicry and the Origins of Neuromorphic Computing

The first direct evidence of life on Earth dates back approximately 3.8 billion years, with Stromatolite fossils of microorganisms discovered in metasedimentary rocks in Greenland [9]. Over these billions of years, life has evolved to overcome numerous challenges, including energy consumption, adaptation, resource gathering, and information processing. As a result, nature has optimised systems and processes that are incredibly efficient and finely tuned to survive on Earth. Given the vast amount of time nature has had to perfect these solutions, it is no surprise that throughout history, humans have increasingly turned to nature for inspiration in solving complex problems. The concept of imitating nature, known as *Biomimicry*, has thus existed for centuries.

One of the earliest and most prominent examples is Leonardo da Vinci, whose fascination with nature in the late

15th century led to groundbreaking designs, particularly for flying machines, inspired by the flight of birds [10]. More recent examples even include the Shinkansen, the high-speed train first developed in the 1990s, where Japanese engineers modelled the front of the train after the beak of a kingfisher to reduce the noise of the loud tunnel pressure wave that occurred when exiting tunnels[11]. These successes highlight a key insight: borrowing strategies from biology can lead to incredible improvements in man-made systems.

From that same philosophy emerged the field of Neuromorphic Computing in the late 20th century, introducing a new perspective for designing computational systems by emulating the structure and processing principles of the biological brain, unlike conventional von Neumann architectures. This field was pioneered in the 1980s by Carver Mead and Lynn Mahowald, who, in their paper *A Silicon Model of Early Visual Processing*, described the development of the first analog silicon retina, an implementation of the initial stages of retinal processing on a single silicon chip [12]. This groundbreaking work demonstrated the potential of silicon-based systems to replicate complex neural functions and it foreshadowed an entirely new type of brain-like computation, given the minimal area and power the device consumed compared to conventional digital processors. It marked a foundational step in the development of neuromorphic computing, and motivated many researchers to further investigate how to incorporate principles from neural processing into the design of computing systems.

3.1.2. Spiking Neurons: Connecting Neuromorphic Computing to Artificial Neural Networks

Before reviewing the historical developments in Section 3.1.3 and Section 3.1.4, we briefly introduce spiking neurons and spiking neural networks (SNNs) to clarify terminology and the provide the reader with a better understanding of SNNs before expanding on the historical background. A more detailed description of SNNs and the theory is given in Section 3.2

Artificial neural networks (ANNs) and neuromorphic computing grew in parallel from the broader idea of learning from biology. The common link is the spiking neuron. A spiking neuron integrates incoming synaptic inputs into a membrane potential. When this potential crosses a threshold, the neuron emits a spike, then resets and briefly becomes refractory [13]. Different mathematical models implement this same basic mechanism with different levels of biological detail or computational practicality. Examples include the leaky integrate-and-fire family [14]. Section 3.2 goes into further depth regarding existing models and their implementation.

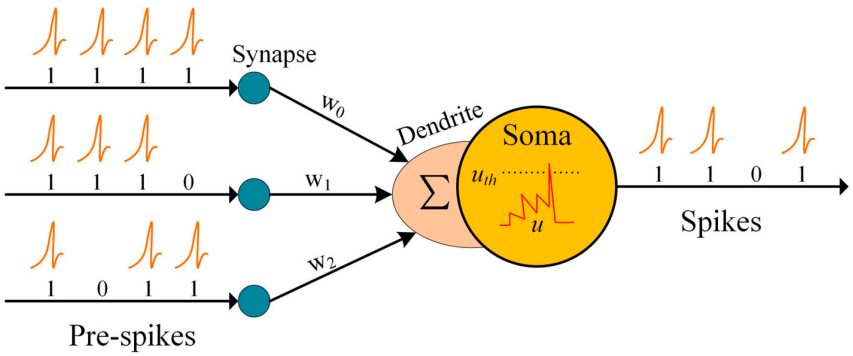


Figure 3.1: The basic model of a single spiking neuron which integrates a weighted sum of synaptic inputs, and spikes when its membrane potential exceeds the threshold [15].

SNNs are built by interconnecting spiking neurons with weighted synapses. Many models also include axonal or synaptic delays that shape when inputs arrive at a neuron. In contrast to conventional ANNs that pass continuous activations on a fixed clock, SNNs communicate via discrete events in time. Computation is event-driven and often asynchronous, so neurons and synapses update only when spikes occur. This produces sparse activity and can reduce energy use on neuromorphic hardware, where power tends to scale with spike activity [16, 17, 3].

With this conceptual foundation in place, the next two sections review the historical developments that shaped today's SNNs. Section 3.1.3 summarizes key milestones in neuromorphic computing, including hardware and sensors that favour event-driven operation. Section 3.1.4 then reviews developments in ANNs that supplied many of the architectural and training ideas later adapted to spiking models.

3.1.3. A Brief Historical Overview of Neuromorphic Computing

Inspired by the work of Mead [12], many other developments ensued in neuromorphic computing, developing hardware, sensors, and algorithms that aim for efficient, event-driven computation. This section gives a concise overview of the main developments within the field of neuromorphic computing and introduces three recurring design principles.

- **Event-driven computation.** Circuits and programs update only when relevant events arrive, rather than on a fixed clock. This asynchronous operation reduces unnecessary computations, lowers latency and is more energy efficient [16].
- **Co-location of memory and compute.** Synaptic weights and state are stored near the neuron and synapse circuits that use them. Shorter data paths cut data movement, significantly reducing energy use and delay [16].
- **Sparse activity with massive parallelism.** At any moment only a small subset of neurons and synapses is active, and many units operate in parallel without waiting for a global clock. Systems and algorithms are designed to exploit this sparsity and parallelism [16].

We describe how these ideas appeared in practice, from early analog and mixed-signal chips to large-scale digital platforms and modern learning methods for neuromorphic systems. The first wave of neuromorphic hardware development explored analog and mixed-signal approaches, which directly implemented neural dynamics in silicon.

Analog and mixed-signal VLSI foundations.

In the late 1980s, neuromorphic hardware emerged from analog very-large-scale integration (VLSI), the practice of packing very large numbers of transistors onto a single chip, so that brain-inspired circuits could be implemented compactly and at very low power [18, 19]. Analog VLSI uses continuous voltages that naturally represent physical quantities such as membrane potential. Mixed-signal designs combine analog cores for efficient continuous-time dynamics with digital logic for configuration, routing, and monitoring. Early chips demonstrated basic neural mechanisms: a leaky integrator that accumulates input while slowly decaying, a threshold element that emits a spike when a level is crossed, and adaptation that adjusts gain based on recent activity to avoid saturation. Designers also kept synaptic weights and state physically close to the neuron circuits that use them, which shortens data paths and reduces energy and latency, choices which established two recurring principles in neuromorphic systems: continuous-time dynamics and memory-compute co-location [16].

Asynchronous spike communication and event sensing.

During the 1990s and 2000s, neuromorphic systems needed scalable ways to move spikes within and across chips without wasting energy. In light of this, the Address-Event Representation (AER) communication scheme was developed which encodes each spike as a short digital packet that carries the neuron address. Events travel on an asynchronous bus that transmits only when data are present rather than at fixed clock ticks. Simple arbitration handles simultaneous requests, and routers can copy events to multiple destinations. When the network is inactive nothing is sent, so bandwidth and power track activity. This makes AER efficient for sparse spiking workloads [20].

In the same period, dynamic vision sensors, also known as event cameras, appeared. Instead of sending full image frames at a fixed rate like conventional cameras, in dynamic vision sensors each pixel reports an event only when the change in log-intensity at that pixel exceeds a threshold. Each event includes the pixel location, the time of the change, and the polarity, which indicates whether brightness increased or decreased. This design provides microsecond-level latency and very high dynamic range, and it aligns well with spike-based processing [21].

Large-scale neuromorphic systems.

By the 2010s, several platforms demonstrated neuromorphic computation at large scale. IBM's TrueNorth chip [17, 22], for example, integrates 4,096 neurosynaptic cores, emulating 1 million spiking neurons and 256 million synapses, all on a single CMOS chip. Consuming only 65 mW, it demonstrates orders-of-magnitude lower power usage for tasks such as visual object recognition compared to von Neumann processors. The design focused on very low energy per synaptic event and reliable large-scale operation [17].

Intel's Loihi neuromorphic research chip [3] contains 128 neuromorphic cores supporting approximately 130,000 neurons and 130 million synapses consuming under 1.5 W. Intel Loihi introduced on-chip learning. In practice this means the chip can update synaptic weights during execution using programmable plasticity rules, rather than only offline on a CPU or GPU. Loihi also supports configurable synaptic delays, hierarchical connectivity across cores, and multi-compartment neuron models where subunits integrate inputs separately before they influence the soma [3].

SpiNNaker targeted real-time emulation of spiking networks using a very large number of low-power processor cores. Each spike is packed into a small network packet and routed over a custom interconnect, which allows flexible models to run with biological wall-clock timing [23].

In parallel, the BrainScaleS-2 platform pursued accelerated analog computation. Its physical neuron and synapse dynamics run faster than biological real time, while digital logic handles configuration and learning support. This acceleration enables rapid experiments with continuous-time dynamics, and the system exposes an event-routing network for external sensors and actors [24].

Across all these platforms, the shared philosophy is to maximize parallelism, minimize idle computation, and align the hardware to the sparse, event-driven computation, making them ideal for SNNs.

Learning methods and integration.
Nearing the late 2010s-2020s training spiking networks with gradients became practical through surrogate-gradient learning. These developments made deeper SNNs trainable on GPUs and CPUs and supported deployment on neuromorphic hardware for efficient inference [5]. Learning methods for SNNs are further detailed in Section 3.2.4.

With the main principles, hardware, sensors, and communication mechanisms for event-driven computation in view, we now turn to Section 3.1.4 for the historic developments in ANNs that supplied the architectures and training methods later adapted to SNNs, linking the neuromorphic and ANN fields.

3.1.4. From Artificial Neural Networks to Spiking Neural Networks

This section reviews three generations of artificial neural networks and the training advances that later transferred into spiking models.

First Generation: Threshold Logic Networks
The modern story of artificial intelligence can be traced back to 1943, when neurophysiologist Warren McCulloch and mathematician Walter Pitts introduced the first ANN, a mathematical neuron model capable of performing logical operations through a network of interconnected neurons [25]. It laid the crucial theoretical foundation for the development of the first generation of ANNs.

Building upon this foundation, the first practical implementation of an ANN was introduced by Frank Rosenblatt in 1958. He developed the Perceptron, a single-layer neural network designed to address binary classification tasks [26]. It included some of most primary components that are crucial for any modern ANN: input nodes were connected to the single layer by a weight. The Perceptron then computed a weighted sum of the inputs and added a bias. This result was then passed to a step activation function which triggered at a given threshold, resulting in an binary output which classified the input to either 1 or 0. In order to learn, the Perceptron was able to update the weights incrementally by adding or subtracting a fraction of the error, allowing it improve its classification performance over time. Rosenblatt also proposed the multilayered perceptron (MLP) model by including a hidden layer of perceptrons. These models were however not yet capable of learning, as there was no general method for training throughout multiple perceptron layers.

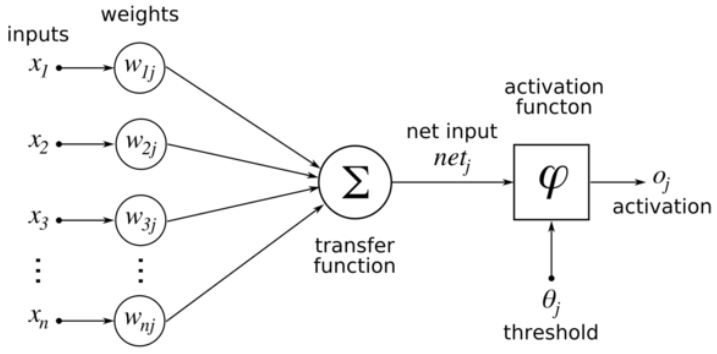


Figure 3.2: The structure of the Perceptron and its main mechanisms such as the weights, summation and threshold step activation function, elements which have now become the crucial components of modern ANNs.[27]

Despite its simplicity, the Perceptron demonstrated the potential of neural networks in pattern recognition. It was in 1969, however, that a key limitation of the Perceptron was brought to light in the book *Perceptrons* by Marvin

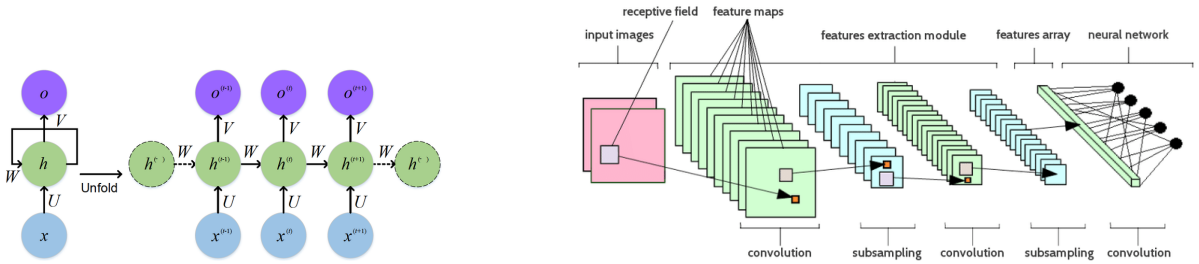
Minsky and Seymour Papert [28]. They showed that the Perceptron was incapable of solving linearly inseparable problems, such as XOR logic function, which cannot be accurately classified by a single-layer neural network. This revelation even led to what is known as the first major AI winter, where AI research experienced a long period of reduced interest [29].

Second Generation: Continuous Activation Networks

Interestingly, it was during this period of reduced interest that two of the most popular architectures in AI appeared [30]. Firstly the Convolutional Neural Network (CNN) in 1980 proposed by Kunihiko Fukushima [31], named the Neocognitron, was designed for recognizing visual patterns based on geometric similarity, regardless of their position or small discrepancies in shape. It marked a significant advancement in the field of visual processing for AI. Another important contribution in this paper was the introduction of the rectified linear unit (ReLU) activation function, being one of the most widely used activation functions to this day for ANNs. The second architecture introduced during this time was the idea of a recurrent neural network (RNN) by John Hopfield in 1982 [32]. The RNN provided a model for solving memory-dependent problems, which required the network to retain memory over time, such as learning associations and optimization problems. These networks had a feedback loop that enabled the network to store and recall patterns. This contribution is a critical foundation for many modern day networks which involve time-series data, speech recognition and language modelling [33].

It was thanks to David Rumelhart and Geoffrey Hinton in 1986 that interest in AI was reignited with their paper *Learning Representations by Back-Propagating* [34]. In this work, they demonstrated how the backpropagation algorithm can be used to efficiently train MLPs by propagating error gradients backward through the network, enabling the adjustment of weights across all layers. It addressed the major limitations of single-layered perceptrons and paved the way for deeper network architectures capable of learning more complex tasks. Backpropagation has since become one of the foundational components of neural networks, serving as the primary method for training deep learning models through gradient-descent.

As a part of this second wave also came the first implementation of a CNN trained with backpropagation by Yann Le Cun in 1989, achieving great success with handwritten digit recognition [35]. Theoretical advancements also contributed to the growing credibility of neural networks. In the same year, George Cybenko presented the Universal Approximation Theorem, proving that a single-layer feed-forward neural network with a finite number of neurons can approximate any continuous function [36]. As neural networks expanded in depth and complexity, training these models became increasingly impractical due to a significant issue identified by Sepp Hochreiter in 1991: the vanishing gradient problem [37]. This problem arises during backpropagation, where gradients used to update weights diminish exponentially as they are propagated backward through each layer. Consequently, weights in earlier layers receive minimal updates, impeding the network's ability to learn effectively. It was this finding that also led to a second period of reduced interest in AI.



(a) The standard RNN unit and unfolded RNN [38].

(b) Example of a CNN architecture [39].

Figure 3.3: Comparison of neural network architectures. (a) shows the unfolded RNN structure and how the feedback loop carries information across multiple timesteps with weight W, while (b) shows a general CNN architecture and how input images can be transformed over multiple layers to continuous output.

While these limitations of ANNs stalled deep networks for a time, they also inspired new ideas and eventually the third generation of neural models. This resurgence during the 2000s was driven by several key advancements. First was the development of more efficient training architectures and algorithms, which addressed the vanishing gradient problem, such as the introduction of the Long Short-Term Memory (LSTM) network in 1997 [40]. Secondly were the rapid advancements in hardware. This led many researchers to advocate using Graphics Processing Units (GPUs) for faster, more efficient training of larger models on bigger datasets. [41]. And lastly was the increasing availability

of large-scale high quality labelled datasets, providing the necessary resources for the increasingly complex models. An example of this is the introduction of ImageNet in 2009, a database of 14 million labelled images, which largely promoted the training of models for object classification, image recognition and automatic object clustering [42].

Third Generation: Spiking Neural Networks

It was also during this time that researchers began pursuing neuron models more faithful to biological dynamics. In 1997, Wolfgang Maass formally characterized SNNs as the third generation of neural networks [13]. Unlike ANNs, which use continuous-valued activations, SNNs communicate via discrete spikes in time, integrating inputs until a threshold is reached, at which point a spike is emitted. This allows them to exploit both spike timing and firing rates for information processing, aligning naturally with neuromorphic hardware and event-based sensing. It is thanks to the many developments regarding ANNs, SNNs could be enhanced using architectures and optimization techniques from traditional ANNs.

Implications for Spiking Neural Networks

The importance of the historical background of neuromorphic computing and ANNs now becomes evident. Neuromorphic computing contributed the core operating principles and platforms for event-driven processing. These include asynchronous communication that transmits information only when activity occurs, memory located close to the compute units that use it, and sensors and hardware that exploit sparse activity for low latency and low energy use. Over time this produced practical platforms for spike-based computation, from address-event networks and event cameras to large-scale digital and accelerated analog systems.

Advances in ANNs have directly enabled modern spiking models. Convolutional and residual patterns for spatial feature extraction have been transferred to the spiking domain through network conversion and deep SNN design [43, 44]. Gradient-based learning has been adapted to spiking dynamics with surrogate gradients, which replace the non-differentiable spike function in the backward pass while keeping hard spikes in the forward pass. This makes deeper spiking networks trainable and supports backpropagation through time for temporal tasks [5, 45, 46]. Tooling, datasets, and accelerators from deep learning have accelerated spiking research. Examples include PyTorch-based SNN libraries with autograd and GPU support (snnTorch [47], SpikingJelly [48], SINABS [49]), simulation platforms for SNNs (Brian2 [50], NengoDL [51]) and code-generation and GPU back ends for large simulations (PyGeNN [52], Brian2CUDA [50]).

SNNs are at the convergence of these two fields. They compute with discrete spikes and threshold dynamics, which align with event-driven hardware, and they benefit from deep-learning practice through techniques such as surrogate gradients [5]. This combination enables real-time and energy-efficient systems while keeping a path to gradient-based learning and established network design patterns. With this background in place, the following Section 3.2 goes into the theory of SNNs, including neuron models and their formulation in Section 3.2.2, encoding methods for information transfer in Section 3.2.3 as well as the specifics of training SNNs in Section 3.2.4.

3.2. Spiking Neural Networks

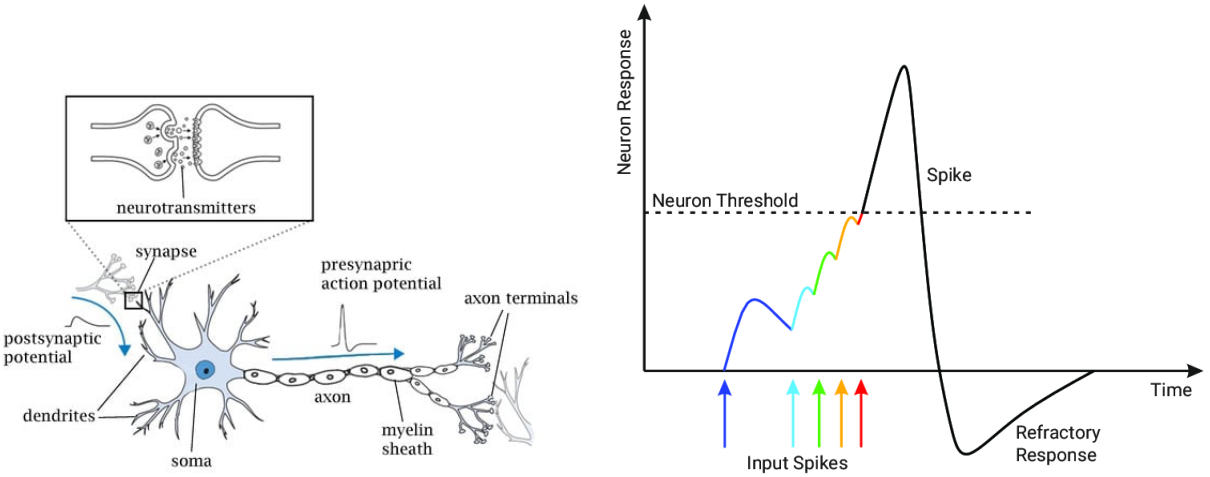
This chapter provides the theoretical foundation required for later sections. It begins with the functioning of biological neurons to motivate the abstractions used in computational models. It then reviews widely used spiking neuron models, ranging from biophysically detailed conductance-based equations to simplified point-neuron formulations such as the leaky integrate-and-fire model. Next, it examines coding methods that define how continuous information is represented and decoded in spike trains, with an emphasis on their implications for accuracy, latency, and energy efficiency. Finally, it surveys learning methods for SNNs, spanning supervised, unsupervised, and reinforcement paradigms, and highlights the mechanisms that make them compatible with neuromorphic constraints. Together, these elements form the conceptual and mathematical basis for the experiments in this thesis, where spiking networks are applied to high-speed quadcopter control.

3.2.1. Basic Functioning of Biological Neurons

This introduction explains how real neurons operate so that the abstractions used in SNNs are clear. Detailed equations and models follow later in this chapter.

Neurons can be described by four main parts: dendrites, the soma, the axon initial segment, and the axon, which ends in synaptic terminals as shown in Figure 3.4a. Neuron inputs arrive at chemical synapses on the dendrites and soma. When a presynaptic spike reaches a synapse, neurotransmitters are released; this opens postsynaptic ion

channels and produces a postsynaptic current that changes the membrane potential of the neuron. These voltage changes spread passively toward the soma and sum across space and time. The axon initial segment is a short segment at the start of the axon with a high density of voltage-gated sodium (Na^+) channels [53]. When the net depolarization at the axon initial segment exceeds the neuron threshold, the Na^+ channels open rapidly and initiate and increase the action potential, known as a spike. After a spike there is an absolute refractory period when a new spike cannot occur. There is also a relative refractory period when the threshold is temporarily higher. These properties limit the firing rate and influence spike timing [54, 55]. This generated spike then propagates along the axon and its branches to the synaptic terminals, where it can trigger transmitter release onto the following neurons [55].



(a) A biological neuron showing the main components that characterize its behaviour and inspire SNN models [56].

(b) Simplified spiking neuron behaviour with integrate–threshold–reset dynamics.

Figure 3.4: From biology to abstraction: a biological neuron and a simplified spiking neuron model used in neuromorphic computing.[57]

The membrane acts like a capacitor also has ion channels which allow the membrane potential to leak over time. Small inputs change this membrane potential with low-pass dynamics set by the membrane time constant, which is the product of membrane resistance and membrane capacitance. The membrane potential also attenuates along dendrites according to cable properties that depend on axial resistance inside the dendrite and membrane conductances. These passive properties shape how inputs integrate over space and time [55, 58].

Another important concept is synaptic strength, which determines how strongly a presynaptic spike changes the postsynaptic membrane potential. Synaptic strength is activity dependent and can change through plasticity. In spike-timing-dependent-plasticity (STDP), the sign and magnitude of long-term change depend on the relative timing of presynaptic and postsynaptic spikes: pre-before-post spikes tend to produce long-term potentiation (LTP), whereas post-before-pre spikes tend to produce long-term depression (LTD) as shown in Figure 3.5 [59, 55]. When many neurons are connected into a network, populations can represent information in different ways. In rate coding, information is carried by average firing rates over a time window. In temporal coding, information is carried by precise spike times, spike order, or inter-spike intervals. Both coding views are used in theory and experiments and motivate different modelling choices, these will be further detailed in Section 3.2.3 [60, 14, 55].

The spiking neural models represent these mechanisms by combining leaky integration, a fixed threshold for spike initiation, a reset after each spike, and optional refractoriness. These models preserve temporal integration, thresholding, and spike-based communication while remaining useful for analysis and simulation [14]. The next Section 3.2.2 details existing neuron models in literature and their implementation.

3.2.2. Spiking Neuron Models

Modern spiking neuron modelling was mainly inspired by the Hodgkin and Huxley equations introduced in 1952. These non-linear differential equations describe ionic membrane currents and the generation of spikes in the giant squid axon [54, 55]. The model couples membrane voltage with activation and inactivation variables for sodium Na^+ and potassium K^+ conductances and reproduces spike initiation and conduction with high quantitative fidelity [54, 60] in a four-dimensional set of equations. The Hodgkin and Huxley framework established a standard in neuron

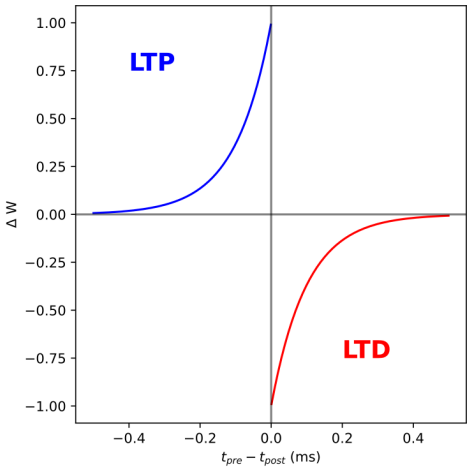


Figure 3.5: Spike-timing-dependent-plasticity describes the observation that synaptic strength W between two neurons depends on the timing of the pre and post-synaptic spike signal. Pre before post leads to long term potentiation (LTP), while post before pre leads to long term depression (LTD) [61].

modelling that inspired many reduced models that trade biological plausibility for simplicity while preserving the main neuronal characteristics [55, 60].

A first major reduction is the FitzHugh–Nagumo system that isolates excitation and recovery in a two-dimensional dynamical system that captures threshold, refractory period, and sustained periodic spiking with abstract variables [62, 63]. The Morris–Lecar model is a more grounded model based on barnacle muscle fibres and provides a two-variable conductance-based description that reproduces oscillations and different excitability behaviours with calcium Ca^{2+} and potassium K^{+} currents [64, 60].

Although these models preserved biological accuracy and were able to replicate a wide range of neuronal behaviours, they remained relatively complex and computationally demanding. To enable more efficient simulations and simpler mathematical analyses, neurons were further abstracted into point-neuron models, in which the entire cell is represented as a single compartment that integrates inputs until a threshold is reached, triggering a spike followed by reset [14, 65]. Historically, this line of thought traces back to Lapicque's 1907 RC threshold model with reset, which anticipated later point-neuron formulations [66, 67]. Given their widespread use in SNNs and practical ease of implementation, we emphasize these models as the main models of interest for spike-based computation and deployment:

- **Leaky integrate-and-fire (LIF)**: Models the membrane as a leaky RC circuit and emits a spike when the voltage crosses a fixed threshold. The model maps input current and leak to the membrane potential with a single state which enables fast simulation and simple analysis and it is one of the most widely used neuron models in SNNs due to its practicality [60, 65, 55]. Its implementation is further detailed in Section 3.2.2.
- **Integrate-and-fire (IF)**: Further simplifies the LIF neuron by leaving out the leak term, which leads to its membrane potential only diminishing when a spike is generated[14, 55].
- **Quadratic integrate-and-fire (QIF)**: Uses a single voltage variable with a quadratic term that makes the voltage accelerate rapidly as it approaches spike generation. A spike is represented by this rapid divergence followed by a reset, which yields rich firing behaviour with a minimal state description [68].
- **Izhikevich model**: This model consists of two states, a membrane voltage and a recovery variable. Its quadratic voltage equation with a simple reset reproduces many firing patterns at very low computational cost [69, 70].
- **Adaptive exponential integrate-and-fire (AdEx)**: This model includes an exponential term that captures sharp spike initiation and a slow adaptation current that produces spike-frequency adaptation. With two coupled equations it matches a wide range of experimental recordings of neuronal voltage responses.[71, 72, 55]
- **Spike Response Model (SRM)**: offers a kernel-based view in which causal filters describe sub-threshold dynamics and the post-spike refractory phase, combined with a threshold condition for spike emission [73, 14].

Leaky integrate-and-fire (LIF) neuron

The practicality of the LIF neuron has made it a central component of SNN research. For example, it is the most common model used when converting ANNs to SNNs, as detailed later in Section 3.2.4, by mapping activations to firing rates, and these converted LIF networks typically preserve accuracy within a small margin on benchmarks such as MNIST and CIFAR [43, 44]. Beyond software, the LIF model has also shaped hardware design: large-scale neuromorphic systems such as IBM TrueNorth and Intel Loihi implement LIF-like compartments to achieve real-time operation at very low energy per spike [22, 3]. Given its central role across software and hardware, in this subsection we detail the discrete formulation of the current-based LIF neuron as used in Python libraries such as snnTorch [47].

Original LIF RC-circuit equations in continuous time

The classical LIF model represents the membrane as a leaky RC circuit driven by input currents:

$$C_m \frac{dV}{dt} = -g_L \left[V(t) - E_L\right] + I_{\text{syn}}(t) + I_{\text{ext}}(t), \tag{3.1}$$

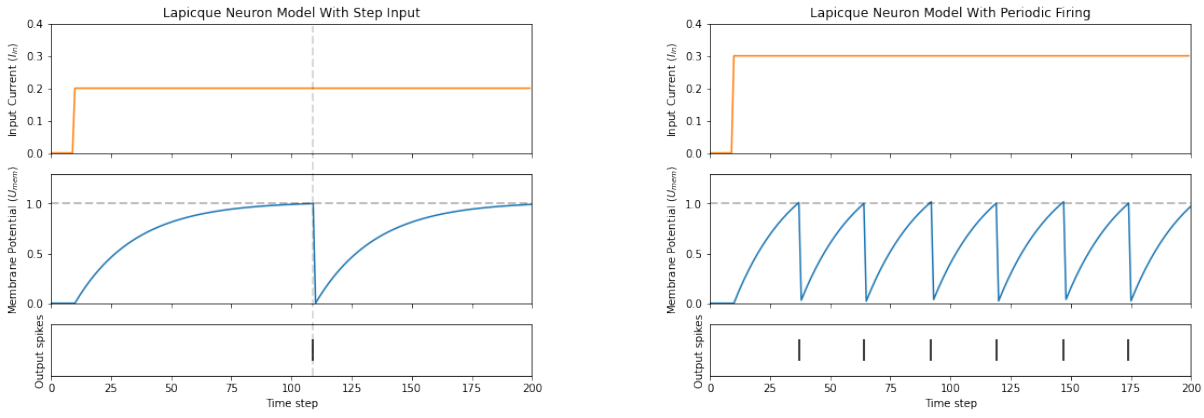
where C_m is membrane capacitance, g_L leak conductance, E_L the leak reversal potential (the voltage at which the leak current is zero), $V(t)$ the membrane potential, $I_{\text{syn}}(t)$ synaptic current, and $I_{\text{ext}}(t)$ an external current. Introducing $\tau_m = C_m/g_L$ and $R_m = 1/g_L$ gives the equivalent form

$$\tau_m \frac{dV}{dt} = -\big[V(t) - E_L\big] + R_m \Big(I_{\text{syn}}(t) + I_{\text{ext}}(t)\Big). \tag{3.2}$$

Spiking is defined by a threshold–reset rule: when $V(t)$ reaches V_{th}, a spike is emitted, V is set to V_{reset}, and the neuron is inactive for a refractory period t_{ref}.

There are two common ways to model synaptic input. In a current-based model, each synapse contributes an added current (simple, fast, and common in analysis and many libraries) [65, 47, 49]. In a conductance-based model, each synapse changes the membrane conductance toward a synapse-specific reversal potential. Because conductance changes alter how easily current flows, they capture shunting effects and make the effective membrane time constant state dependent. Conductance-based models are more biophysically faithful, while current-based models are lighter and easier to tune [55].

Discrete formulation of the current-based LIF model



(a) Membrane potential and spiking activity for LIF neuron with constant input current $I_{in} = 0.2$.

(b) [Membrane potential and spiking activity for LIF neuron with a higher constant input current of $I_{in} = 0.3$.

Figure 3.6: Side-by-side comparison of membrane potential for the Lapicque LIF neuron in snnTorch for two different input currents. A higher constant input current I_{in} leads to a higher firing rate of the neuron [47].

To simulate the LIF neuron, the continuous-time dynamics are stepped in discrete time. Two standard one-step updates are widely used [50, 47, 52]. The first method is the exact exponential update, being a closed-form solution which assumes the input is approximately constant over one step Δt:

$$V_{t+\Delta t} = E_L + \left(V_t - E_L\right) e^{-\Delta t/\tau_m} + R_m I_t \left(1 - e^{-\Delta t/\tau_m}\right). \tag{3.3}$$

Another simpler method to obtain the solution is using a Forward-Euler update, being a simple first-order approximation:

$$V_{t+\Delta t} = V_t + \frac{\Delta t}{\tau_m}\Big(E_L - V_t + R_m I_t\Big). \tag{3.4}$$

After each step, a threshold is checked and a reset is applied. An optional refractory counter can hold the voltage fixed for a few steps after a spike. In practice, Python libraries recentre voltage at rest and absorb the membrane resistance into the input. Defining $v = V - E_L$, $i_t = R_m I_t$, and $\beta = e^{-\Delta t/\tau_m}$ (or $\beta \approx 1 - \Delta t/\tau_m$ for Euler), the recurrent form used in snnTorch and similar libraries is:

$$v_{t+\Delta t} = \beta\, v_t + (1-\beta)\, i_t \;-\; s_t\, V_{\text{th}}, \tag{3.5}$$

where s_t is the binary spike at step t (1 if a spike occurred, else 0). The subtraction implements a soft reset by one threshold. The reset mechanism can also reset the membrane potential $v_{t+\Delta t}$ to zero, know as a hard reset. In this parametrization, R_m does not appear explicitly because it is folded into the effective input i_t, and E_L vanishes because v is stored relative to rest [47].

3.2.3. Neural Coding Methods

To connect many neurons into a functional SNN, the interface to continuous signals must be defined at both ends. Inputs are encoded into spike trains or injected currents and outputs are read back (decoded) into continuous estimates or actions [60, 14, 55]. This section presents the main coding rules with their usual decoding and highlights applications that report effects on latency, spike count, energy, accuracy, and training stability [56, 47].

Rate Coding

A continuous signal is mapped to a firing rate that is proportional to the input magnitude [60, 14, 55].

$$\lambda(t) = \kappa\, x(t) \tag{3.6}$$

Here, $\lambda(t)$ is the instantaneous firing rate, $x(t)$ is the input signal, κ is a gain, and t is time. Spikes are drawn from an inhomogeneous Poisson generator. In small time bins each bin behaves like a single Bernoulli trial with probability equal to the local rate times the bin width, and the probability of two spikes in the same bin is negligible for sufficiently small bins [60, 14, 55]. A common readout estimates the average firing rate by counting spikes in a fixed window [60, 14].

$$\hat{\lambda} = \frac{n}{T} \tag{3.7}$$

Here, $\hat{\lambda}$ is the estimated rate, n is the number of spikes observed, and T is the window length. An alternative decoding method is exponential filtering, which assigns exponentially decaying weight to past spikes [60, 14, 55].

$$k(t) = H(t)\, \exp\Big[-\frac{t}{\tau_s}\Big] \tag{3.8}$$

$$y(t) = \sum_j k\big(t - t_j\big) \tag{3.9}$$

Here, $k(t)$ is a causal synaptic kernel, $H(t)$ is the Heaviside step, τ_s is the synaptic time constant, $y(t)$ is the filtered activity, and t_j are spike times. This remains rate decoding because the expected filtered activity tracks the firing rate smoothed by the kernel. For slowly varying or constant rates a local rate estimate follows by dividing by the kernel area A[60, 14].

$$\hat{\lambda}(t) \approx \frac{y(t)}{A\tau_s} \tag{3.10}$$

An important application of rate coding is within ANN-to-SNN conversion methods, where trained ANNs are converted to SNNs through adaptations which align the behaviour of the SNN with the ANN. These procedures rescale weights and biases, choose neuron thresholds, and select a simulation window so that spike counts or filtered postsynaptic currents reproduce ANN activations with small error [43]. This approach converts common layers such as ReLU, pooling, batch normalization, and softmax, and it preserves accuracy on large scale vision benchmarks with VGG and Inception style models while enabling event-driven execution [43, 44, 74]. The same decoded outputs map cleanly onto neuromorphic chips where synaptic filters accumulate spikes and computation triggers only on events,

which reduces switching activity and supports low power inference [17, 22, 3]. In a different setting, unsupervised digit recognition used Poisson rate input to drive stable feature learning and reported robustness to input noise, with longer windows and higher spike counts as the trade-off [75]. Broader surveys describe the same pattern across tasks. Rate coding is simple and reliable, and the main cost is added latency and spike budget compared to temporal codes [56, 47].

Time to First Spike (TTFS)

A normalized input maps to a single spike time so that larger values fire earlier and smaller values fire later [14, 55].

$$t_s = t_0 + \tau_{max}\,[1 - x] \tag{3.11}$$

Here, t_s is the spike time, t_0 is the earliest allowed time, τ_{max} is the maximal latency, and $x \in [0, 1]$ is the normalized input. To decode this scheme either takes the earliest spike across a population for decisions or inverts the mapping to recover an estimate of x [14, 55]. First-to-spike decisions, when trained with standard SNN learning methods, achieved fast and sparse inference on vision tasks with fewer spikes and shorter decision times than rate codes, which reduced energy and latency in practice [76, 77]. Hardware-oriented networks using first–spike rules further cut reaction time at the edge while maintaining accuracy, provided training controls timing noise [78, 47].

Rank Order

Each neuron is allowed at most one spike and the information is carried by the order in which neurons fire [79, 80].

$$t_{s,\pi(1)} < t_{s,\pi(2)} < \cdots < t_{s,\pi(N)} \tag{3.12}$$

Here, $t_{s,i}$ is the first spike time of neuron i, N is the population size, and π is the permutation that orders neurons by first spike. Decoding uses either the first spike or the explicit order of first spikes across the population to produce a decision. This is rank–based readout [81, 14]. Latency pipelines that encode intensity as earlier spikes and then apply first spike (order) or rank pooling extracted oriented features and reached competitive accuracy with only a few spikes per image, which enabled rapid categorization [82]. Supervised methods that match precise spike times solved standard benchmarks under low spike budgets (spike count is constrained). These readouts rely on exact latencies rather than being strictly rank–invariant, yet they still show that the relative order of first spikes can support fast decisions with little activity [83, 81]. The approach is sparse and fast, and it is more sensitive to timing jitter and conveys weaker amplitude information than rate codes [80, 81, 56].

Burst Coding

Burst coding conveys information by groups of spikes fired in rapid succession, called bursts, rather than by isolated single spikes [84]. A burst is typically defined as a short sequence of spikes separated by interspike intervals (ISI) much shorter than those between bursts. Encoding schemes use the number of spikes in a burst, the timing of burst onset, or both to represent signal features [85]. This provides a richer vocabulary than single spikes because bursts can simultaneously transmit discrete events and graded values.

Decoding methods count spikes within bursts or detect burst onset times. Burst count can be mapped to input amplitude while burst onset time carries temporal information, which allows the combination of rate and temporal codes, this can enhance robustness by reducing the probability that important events are lost to noise or transmission failures [85]. Computational models demonstrate that burst codes can increase efficiency in SNNs by allowing low baseline activity with rapid, high–fidelity responses when relevant stimuli appear [72].

Phase of Firing

This scheme uses an oscillating carrier signal which provides a reference. Information is then encoded into the spikes by assigning a phase to each spike [14, 86].

$$\phi(t) = \omega t \bmod 2\pi, \qquad \phi_s = \phi(t_s) \tag{3.13}$$

Here, $\phi(t)$ is the instantaneous phase, ω is the carrier angular frequency, t is time, and ϕ_s is the phase at the spike time t_s. For decoding, the scheme uses circular averaging or phase bins and then maps the phase code to values or classes [86]. Research shows that in the auditory cortex of animals, the phase of firing relative to local oscillations adds information about natural sounds beyond rate and improves decoding accuracy on real recordings [86]. Models that align input phases boost timing-based plasticity and converge faster on temporal tasks, which supports efficient learning when oscillatory structure is present. These gains depend on a stable carrier and precise timing [87, 56].

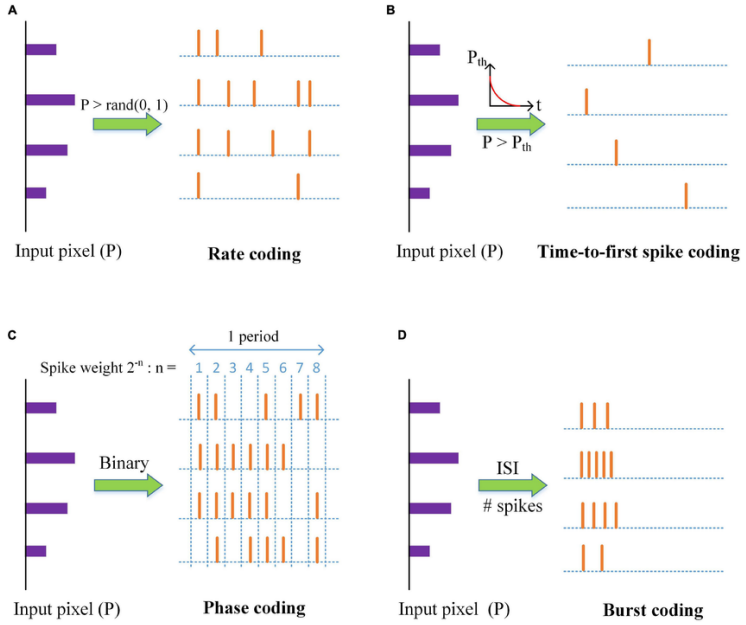


Figure 3.7: A representation of coding schemes A: Rate coding, B: TTFS, C: Phase coding, D: Burst coding, with a example input pixel P and the resulting encoding spikes for each scheme [88].

Population Coding

A single value is represented by many neurons that respond over overlapping ranges. A common choice is Gaussian tuning [60, 55].

$$r_i(x) = r_{max}\ \exp\Big[-\frac{(x-\mu_i)^2}{2\sigma^2}\Big] \tag{3.14}$$

Here, $r_i(x)$ is the mean firing rate of neuron i, x is the encoded input, r_{max} is the peak rate, μ_i is the preferred value of neuron i, and σ is the tuning width. Spikes are typically generated from these rates with a Poisson sampler [60]. A simple readout forms a population vector that averages preferred values using the current activity as weights [60].

$$\hat{x}_{pv} = \frac{\sum_i \bar{r}_i\,\mu_i}{\sum_i \bar{r}_i} \tag{3.15}$$

Here, $\hat{x}_{pv}$ is the estimate and $\bar{r}_i$ is a rate estimate for neuron i obtained from spike counts or exponential filtering under a rate decoding scheme as in Section 3.2.3 [60]. Under a Poisson model, maximum likelihood uses observed counts n_i in a window of length T [60].

$$\hat{x}_{ml} = \arg\max_x \Big[\sum_i n_i \log r_i(x) - T\sum_i r_i(x)\Big] \tag{3.16}$$

Here, $\hat{x}_{ml}$ is the value that best explains the counts given the tuning curves. For Gaussian tuning the maximum response occurs at $x = \mu_i$, and σ sets the selectivity bandwidth. A common width metric is the full width at half maximum,

$$\mathrm{FWHM} = 2\sqrt{2\ln 2}\,\sigma, \tag{3.17}$$

which shows how σ controls the half–maximum width [60]. Probabilistic population codes pair these tuning functions with noise models to support robust inference, and linear decoders can be very accurate in this setting. This is useful for continuous control and for readouts that must stay calibrated under sensor drift or changing noise [89, 60, 55].

When fitting tuning curves to data, μ_i and σ are curve–fit parameters estimated from responses of neuron i. They describe preference and selectivity rather than the mean and spread of the external stimulus ensemble. Spike–count variability around the mean rate is commonly modelled separately, for example with Poisson noise where the count variance equals the mean [60, 55].

Direct Current Injection

The first layer receives a continuous current that is proportional to the input. No Poisson sampling is used and the layer itself acts as a learnable spike generator [90, 91, 92, 93].

$$I_{\text{ext}}(t) = \kappa\, x(t) \tag{3.18}$$

Here, $I_{\text{ext}}(t)$ is the injected current at time t, $x(t)$ is the input signal, and κ is a gain that sets the scale. Through this method continuous input values can also be directly passed on to the neuron with $\kappa = 1$. Subsequent neurons integrate this current and produce spikes when their membrane potentials cross threshold. A linear output layer then reads filtered activity, the membrane potential, or spike counts using rate decoding.

This method reduces latency and energy because it removes input sampling noise and allows accurate inference with few time steps. A representative example is DIET-SNN, which applies analog pixel values directly to the input layer and, during training, optimises thresholds, leaks, and weights. It reports near ANN accuracy on image benchmarks with very few time steps and lower compute energy than rate encoded baselines [90]. Calibration based conversion reaches high accuracy at low time steps by passing input to the first layer directly and correcting internal mismatches with analytic and layer-wise calibration, which stabilizes low latency inference [91].

This method does have the downside that training and conversion at very small time steps are sensitive to internal deviations that accumulate across layers. Direct input reduces sensor noise but does not remove these internal errors. Accurate low latency SNNs therefore rely on careful threshold balancing, normalization, and error control in deeper layers [92, 93].

Event Camera Output

Dynamic vision sensors emit asynchronous address events when the change in log intensity at a pixel crosses a contrast threshold [21, 94].

$$\Delta\log I(x,y,t) \geq \theta \quad \text{or} \quad \Delta\log I(x,y,t) \leq -\theta \tag{3.19}$$

Here, $\Delta\log I(x,y,t)$ is the local change in log intensity at pixel (x,y) and time t, and θ is the contrast threshold. Each event carries a pixel address, a precise timestamp, and a polarity bit that indicates the sign of the change [21, 94].

$$p(x,y,t) = \text{sign}\big[\Delta\log I(x,y,t)\big], \qquad p \in \{+1,-1\} \tag{3.20}$$

Thus the event values are binary in polarity. There is no grayscale intensity attached to an event. Information is encoded by when and where events occur and by their sign [21, 94]. A SNN can consume the stream directly by assigning one input neuron per pixel and emitting an input spike at the event time with the sign of the synapse set by the polarity. Positive events will excite the membrane potential and negative events inhibit it. The postsynaptic current is then updated by the used spiking neuron model [94]. Alternatively, it is also possible to produce a decaying time trace of events so that recent events produce larger values and older events decay away. A common construction stores, for each pixel, a leaky memory of the last event time [94].

$$S(x,y,t) = \exp\Big[-\frac{t - t_{\text{last}}(x,y)}{\tau_{ts}}\Big] \tag{3.21}$$

Here, $S(x,y,t)$ is the time surface value, $t_{\text{last}}(x,y)$ is the timestamp of the most recent event at (x,y), and τ_{ts} is the decay constant. This yields a continuous map that can drive the first layer as a current input or be combined with polarity by maintaining separate traces for $p = +1$ and $p = -1$ [94]. The use of these event streams can provide microsecond latency and very high dynamic range. Systems that use these streams achieve fast tracking and robust perception in robotics and navigation where frame cameras saturate or blur. Like many other methods the asynchronous and sparse nature of events also matches neuromorphic hardware well and restrains computation only to events [94, 21]. A downside, however, is that event cameras provide polarity and time but not absolute intensity, so static regions produce no data and tasks that need brightness require reconstruction or fusion [94]. Event cameras also exhibit pixel dependent thresholds, background activity in low light, missed events, and refractory effects at very high contrast motion, which reduce accuracy if not modelled [94, 21].

Sigma–delta Coding

This code scheme transmits only when the signal changes enough to matter. It emits sparse signed events that keep a running reconstruction close to the true signal within an error band set by a threshold. The aim is to cut bandwidth and operations while preserving accuracy in deep SNNs where many activations vary slowly over time. A predictive

decoder maintains an estimate of the input and triggers an event when the instantaneous error exceeds a threshold [95].

$$e(t) = x(t) - \hat{x}(t), \qquad \text{emit } s \in \{+1, -1\} \text{ when } |e(t)| \geq \vartheta \tag{3.22}$$

Here, $x(t)$ is the input, $\hat{x}(t)$ is the running estimate, $e(t)$ is the residual, ϑ is the threshold, and s is the event sign. The estimate is computed by filtering the signed event train with a causal reconstruction kernel and scaling by the step size [95].

$$\hat{x}(t) = \hat{x}_0 + \vartheta \sum_i s_i \, k\big(t - t_i\big), \qquad \int_0^\infty k(u)\, du = 1 \tag{3.23}$$

Here, $\{t_i, s_i\}$ are event times and signs, $k(t)$ is a unit–area kernel, and $\hat{x}_0$ is an initial offset. A common choice is an exponential kernel that implements a leaky integrator [95].

$$k(t) = H(t) \, \frac{1}{\tau_r} \, \exp\Big[- \frac{t}{\tau_r} \Big] \tag{3.24}$$

Here, $H(t)$ is the Heaviside step and τ_r is the reconstruction time constant. In SNNs the same kernel typically defines the postsynaptic current, so later layers consume the filtered current directly without an explicit reconstruction stage [95]. Applied inside deep networks, sigma-delta neurons reduce operation counts while keeping accuracy close to dense baselines. The trade-off is that it functions as a decoder with a state and weighted signed spikes, and performance depends on proper choices of threshold and kernel [95].

3.2.4. Learning Methods for Spiking Neural Networks

Training SNNs is difficult because spike generation is non-differentiable and because temporal dynamics complicate gradient propagation across many steps [5, 96]. In addition, neuromorphic hardware favours local updates, bounded memory, and low precision which constrains which learning rules are practical [3, 22]. To analyse the existing learning methods for SNN, it is helpful to first make the distinction between learning algorithms and update rules [5, 97]. Learning algorithms specify labels or objectives that guide the learning procedure of the network, while mechanisms specify how parameters within the network are updated, such as global gradients, local three factor rules, eligibility traces, or conversion pipelines [97, 5]. Learning algorithms have been separated by supervised and unsupervised methods, while reinforcement learning methods for both ANNs and SNNs will be detailed further in Section 3.2.5.

Supervised Learning

Supervised learning uses labelled input-output pairs and an explicit loss to align network predictions with known outputs [98, 97]. The objective guides all updates so that the model reduces classification error or regression error on the training distribution [98, 5]. In SNNs, using this method also entails using an appropriate decoding method to obtain continuous values from output spikes as discussed in Section 3.2.3 [5, 99].

Surrogate Gradient Descent replaces the non-linear gradient of a spike with a smooth differentiable alternative during the backward pass which allows backpropagation through time to proceed [5, 99]. The forward dynamics maintain thresholding and reset while the backward pass uses a bounded surrogate derivative to approximate the gradient of the spike function [99, 5]. Common supervised objectives optimise the cross entropy on spike counts for classification and rank order coding for timing tasks [99, 100, 5]. A standard surrogate derivative is the sigmoid function, shown in Figure 3.8. It is centred at the firing threshold with a slope parameter α that controls gradient magnitude and width which provides stable and smooth gradients for training [99, 5]. Another common choice is a piecewise linear triangular window that assigns a constant derivative within a narrow band around the threshold and zero derivative outside which preserves locality of the learning signal in voltage space [100, 5]. Implementations improve stability with gradient clipping and careful handling of membrane resets during the backward pass to avoid exploding or vanishing gradients through time [5, 99]. Surrogate gradient training achieves strong accuracy on static frames and event streams but increases memory and time because temporal states must be stored across many steps during backpropagation through time [5, 100].

Online Approximations with Eligibility Traces compute a local trace at each synapse from pre and post activity and then multiply this trace by a broadcast learning signal to update the weights [96]. E-prop is an example that uses filtered traces that capture how earlier spikes influence current activity and uses target or error signals that do not require exact symmetric weight transport, which is the requirement that the feedback pathway of weights

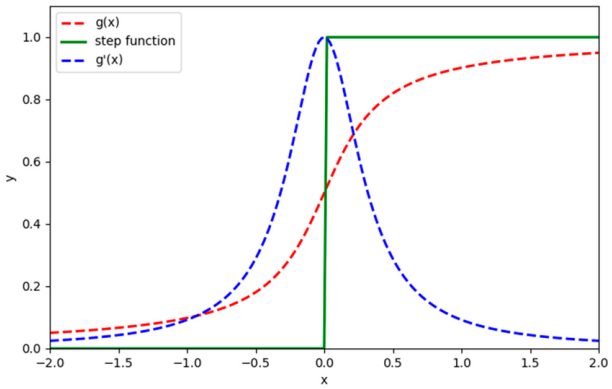


Figure 3.8: The spike heavy side step function, the sigmoid function $g(x)$ and its derivative $g'(x)$ which is a commonly used surrogate gradient in SNNs to enable backpropagation [101].

during backpropagation must mirror the forward pathway for effective backpropagation [96]. E-prop therefore yields online learning with modest memory and competitive performance on tasks that require temporal integration [96]. The separation of local eligibility and global learning signals improves hardware fit because it avoids global synchronization of gradients [96].

Local Errors and Feedback Alignment attach simple auxiliary heads to each layer in the network so that each layer receives a training signal that depends only on nearby activity [102]. Feedback alignment replaces exact transpose feedback paths with fixed random feedback which reduces the requirement for symmetric weights in deep training [103]. Direct feedback alignment sends error signals to every layer through fixed random projections which further relaxes global gradient flow [104]. These approaches reduce coordination across layers and improve compatibility with on chip learning while often trading peak accuracy for locality [103, 102].

Timing Based Perceptrons also known as Tempotrons, learns to classify input spike patterns by shifting synaptic efficacies so that the membrane potential crosses threshold for target classes and stays below threshold for others [105]. The rule uses spike timing encoding of pre spikes and the membrane kernel to compute a margin like quantity which guides weight updates from single trial outcomes [105]. Alternatively, the Chronotron learns to produce target output spike times by minimizing a spike train distance and adapting synapses to align actual and desired latencies [106]. These methods are attractive when information is carried in precise spike times and when low inference latency is important [105, 106].

Spike Train Distance Objectives turn temporal alignment into a supervised loss for spiking outputs [107, 108]. The Victor Purpura metric defines a set of edit operations with costs and the training objective reduces the edit cost between the produced and the target spike trains [107]. The van Rossum distance filters spike trains with an exponential kernel and computes a squared difference which produces a smooth objective that respects temporal structure [108]. These distances encourage networks to match the full temporal pattern rather than only spike counts [107, 108].

ANN-to-SNN Conversion as mentioned in Section 3.2.3 trains a conventional network with ReLU units and then maps activations to integrate and fire neurons by setting thresholds and normalizations so that rate codes reproduce the analog responses [43]. Proper weight normalization and bias handling avoid rate saturation and preserve activation ordering across layers which maintains accuracy on vision benchmarks [43]. Deeper models can be converted by correcting layerwise imbalances and by calibrating thresholds which improves robustness and speed of rate accumulation [44]. Conversion yields high accuracy and easy reuse of mature ANN training pipelines but it introduces a latency-accuracy trade-off because rate estimates need multiple time steps to be accurate [43, 44].

Unsupervised and Self Supervised Learning

Unsupervised learning discovers structure in data without labels or known ground truths, and optimises objectives that capture regularities in inputs [98, 97]. Typical goals include clustering, sparse coding, prediction, and reconstruction

which build representations that transfer to subsequent tasks [98, 5]. In SNNs these goals map naturally to local plasticity rules and to layerwise targets that use only nearby activity [97, 5].

Hebbian Learning strengthens synapses when presynaptic and postsynaptic activity occur near each other and weakens them when they are anti correlated which captures the idea that neurons that fire together wire together [109, 110]. Simple Hebbian rules discover structure in inputs such as oriented edges under competition and lateral inhibition which yields selective receptive fields [111, 110]. Pure Hebbian learning can be unstable because activity can grow without bound which motivates the implementation of synaptic normalization and activity homeostasis to maintain balanced firing [112, 110].

Spike-timing-dependent-plasticity (STDP) makes the synaptic change depend on the relative timing between pre and post spikes which introduces causality into the rule [59, 110]. A pre spike that precedes a post spike within tens of milliseconds drives potentiation while the reverse order drives depression which shapes temporal feature sensitivity [59, 110]. Triplet STDP adds interactions among pairs of post spikes or pairs of pre spikes which captures rate effects that are not explained by pair based rules [113]. Competitive STDP with lateral inhibition leads to emerging filters and orientation tuning in early vision layers which demonstrates unsupervised emergence of structure [87]. These dynamics depend on input statistics and on the width and amplitude of the plasticity windows which link coding choices to learned features [110, 87].

.

Unsupervised Pipelines with Simple Readouts STDP based encoders can feed a simple classifier that operates on spike counts which yields competitive accuracy on digit recognition without global labels during representation learning [75]. The encoder learns features from streams of spikes while a linear readout or a simple vote on spike counts handles the final decision which keeps training local in the feature extractor [75]. Surveys report that such pipelines provide sample efficient representations while trading maximal accuracy for locality and energy benefits [97]. Self supervised objectives extend these ideas by training layers to predict future events or to reconstruct streams which builds temporal features without external labels [97, 5].

3.2.5. Reinforcement Learning Methods

Next to supervised and unsupervised methods, reinforcement learning (RL) is another main approach towards training both ANNs and SNNs, which is particularly suited to robotic control tasks. RL addresses control as sequential decision making under uncertainty. An agent observes a state, selects an action, and receives a reward from the environment as shown in Figure 3.9. The aim is to learn a policy that maximizes long term performance, which aligns well with robotic control tasks [114].

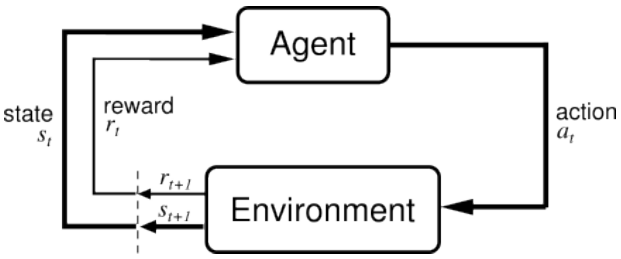


Figure 3.9: The agent-environment interaction which encapsulates the basic elements of reinforcement learning [114].

The basic task behind RL can be modelled as a Markov decision process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma, \rho_0)$ with state space $\mathcal{S}$, action space $\mathcal{A}$, transition law $P(s'|s, a)$, reward function $r(s, a)$, discount factor $\gamma \in (0, 1)$, and initial state distribution ρ_0. A policy $\pi(a|s)$ assigns a distribution over actions to each state [114]. The objective function $J(\pi)$ quantifies performance as the expected discounted return along trajectories generated by ρ_0, P, and π:

$$J(\pi) \;=\; \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t\, r(s_t, a_t)\right]. \tag{3.25}$$

The factor $\gamma \in (0, 1)$ downweights distant rewards and ensures the series is finite [114]. Two value functions summarize future utility under a fixed policy. The state value $V^{\pi}(s)$ measures how good it is to start in state s and then follow π, while the action value $Q^{\pi}(s, a)$ measures how good it is to take action a at s and then follow π:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \, r(s_t, a_t) \;\middle|\; s_0 = s\right], \tag{3.26}$$

$$Q^{\pi}(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \, r(s_t, a_t) \;\middle|\; s_0 = s,\ a_0 = a\right]. \tag{3.27}$$

These functions are linked by the policy weighted average

$$V^{\pi}(s) \;=\; \mathbb{E}_{a \sim \pi(\cdot|s)}\big[Q^{\pi}(s, a)\big], \tag{3.28}$$

which shows that state value aggregates action values under the current policy [114]. Many algorithms either learn Q^{π} or V^{π} to guide action selection, or they optimise $J(\pi)$ directly when the policy is differentiable. The policy gradient theorem provides a practical gradient that does not require differentiating through unknown dynamics:

$$\nabla_{\theta} J(\pi_{\theta}) \;=\; \mathbb{E}_{s \sim d^{\pi_{\theta}},\, a \sim \pi_{\theta}}\big[\nabla_{\theta} \mathsf{log}\, \pi_{\theta}(a|s)\, Q^{\pi_{\theta}}(s, a)\big], \tag{3.29}$$

where $d^{\pi_{\theta}}$ is the discounted state visitation distribution under π_{θ}. Actor critic methods estimate the value terms to reduce variance and improve sample efficiency in robotics [114, 115].

This formulation forms the basis for the main algorithm families reviewed next in Section 3.2.5 for both ANNs and SNNs, including value-based methods that learn V or Q for action selection, policy gradient and actor critic methods that optimise $J(\pi)$ from sampled rollouts (sampled state-action pairs from the environment), and model based methods that learn dynamics to plan or to improve the policy [114].

Reinforcement Learning Algorithms for ANNs

ANNs implement policies and value functions that are trained from reward signals using gradient based updates, which makes deep RL practical at scale [114]. In robotics the choice of algorithm is shaped by the action space, the amount of data that can be collected, and real time compute limits [116]. We therefore organize the methods that follow into four families that are standard in practice for ANN based control in robots: value-based control, on-policy gradient with trust region updates, off-policy actor critic for continuous actions, and model based policy search [114, 116].

Value-Based methods learn an action value function and extract a greedy or near greedy controller. Deep Q learning and its descendants initiated the modern wave of deep RL [117]. For continuous control in robotics, direct Q learning is adapted through continuous action selection or stochastic policies driven by a learned Q function. QT Opt is a scalable vision based grasping system that learns a continuous control grasping policy with batch Q learning and achieves strong success rates on real manipulators [118]. Applications in practice include closed loop robotic grasping under visual feedback with high success on unseen objects [118].

On-policy Policy Gradient and Trust Region Methods optimise a stochastic policy from fresh rollouts using a surrogate objective. Trust Region Policy Optimization (TRPO) constrains updates for stable improvement [119]. Proximal Policy Optimization (PPO) simplifies the constraint with clipped ratios and is widely used in robotics due to robustness and ease of use [8].

Champion level autonomous drone racing demonstrates on-policy learning at scale. The Swift system trains a visuomotor policy in simulation and transfers it to the real platform, running onboard with only onboard sensing. The learned controller trained with proximal policy optimization matches and at times surpasses top human pilots on real tracks [120, 8].Another similar study shows that a single neural controller can generalize across distinct quadcopter platforms when trained using PPO to map state to output motor RPMs. One Net to Rule Them All trains one policy with domain randomization and validates it on 3 inch and 5 inch race drones in real flights. The generalized controller is slightly slower than fine-tuned models yet transfers across platforms reliably and reaches speeds up to ten meters per second, which highlights the trade-off between robustness and peak performance when generalizing a model through policy training [121, 8].

At low level, attitude controllers for quadcopters trained with PPO have been compiled to embedded firmware and evaluated on real airframes. Neuroflight reports higher tracking precision than PID baselines in simulation, real time execution at kilohertz rates on resource constrained hardware, and successful flight tests, which illustrates feasibility for deployable flight control [122, 123]. For legged locomotion, massively parallel PPO produces policies for the ANYmal quadruped in minutes in simulation and transfers them to hardware with strong zero-shot performance. This setup enables rapid iteration and robust behaviours over uneven terrain [124]. Earlier work combined accurate actuation models and robustness training to achieve agile gaits and recovery on the real robot using policies learned in simulation [125]. Dexterous manipulation also leverages on-policy training. A recurrent policy trained entirely in simulation with PPO and automatic domain randomization solved a Rubik's Cube with a multi fingered robot hand and showed resilience to perturbations during real world execution [126, 8].

Off-policy Actor–critic for Continuous Control methods learn a policy by following gradients provided by a value critic while reusing data through a replay buffer, which improves sample efficiency relative to on-policy methods [127, 124]. Deterministic Policy Gradients introduced the deterministic policy gradient theorem and showed that an off-policy actor can improve by ascending the gradient of the critic with respect to the action, which laid the foundation for continuous control [128]. Deep Deterministic Policy Gradients (DDPG) extended this idea with deep function approximation, target networks, and replay, and demonstrated end-to-end learning from pixels and state inputs across many physics tasks [127]. Twin Delayed Deep Deterministic Policy Gradients (TD3) reduced overestimation and training instability by learning two critics, updating the actor less frequently than the critics, and smoothing target actions with clipped noise, which produced strong gains over DDPG on continuous control benchmarks [129]. Soft Actor–Critic (SAC) optimised a maximum entropy objective that encourages high-entropy policies while maximizing return, and combined off-policy updates with a stochastic actor and a stable critic to achieve state-of-the-art performance with good data efficiency [130]. On hardware, SAC learned a stable walking gait for the Minitaur robot directly on the real system in about two hours and produced a policy that was robust to moderate perturbations, which highlighted the practicality of off-policy learning for real robots [131].

Model-based Policy Search learns a predictive dynamics model and uses it to plan actions or to improve a policy, which can reduce interaction cost by trading computation for data. PILCO learned Gaussian process dynamics and evaluated policies analytically, which enabled learning controllers for classic control tasks and small robots in very few trials with high data efficiency [132]. Model-Based Policy Optimization also combined short rollouts from a learned ensemble with real experience and trained a model-free learner on this mixed dataset, which limited model bias and delivered state-of-the-art sample efficiency across continuous control tasks [133]. In practice, such model-based components can speed up learning on hardware-scale systems where interaction is costly, and they can also warm-start or augment model-free training to reach strong final performance with less data [132, 133].

Reinforcement Learning Algorithms for Spiking Neural Networks

SNNs offer event-driven computation and low latency, yet training remains difficult due to discontinuous spikes and long range credit assignment across timesteps [5, 134]. Currently two main approaches exist in practice for RL with SNNs. The first uses surrogate gradients to enable end to end policy optimization. The second uses three factor rules that combine local eligibility traces with a global reward prediction error, including e-propagation style online updates [5, 135, 136, 96, 134]. The following examples mentioned in this subsection, fall under these two categories.

Within continuous control, a consistent pattern has emerged. Population coded spiking actors are trained together with deep critics and reach returns comparable to deep actors while enabling efficient deployment on neuromorphic chips [137]. The PopSAN framework integrates with on-policy and off-policy algorithms and, on the Intel Loihi, reduced energy per inference by about two orders of magnitude at similar task performance to a deep actor on embedded GPUs [137]. The same neuromorphic deployment method extends to mobile navigation. A hybrid spiking actor paired with a deep critic was trained for mapless navigation and executed on Loihi with large energy savings while maintaining success rates, which supports onboard autonomy under tight power budgets [138]. General purpose toolkits now make it practical to train fully spiking policies with standard deep reinforcement learning [47]. A recent comparative study trained SNN policies with PPO in the Isaac Gym simulator, explored network configurations, and reported competitive performance and useful training throughput when compared to ANN baselines [4]. Beyond simulation, actor-critic SNNs with temporal coding have been validated in hardware in the loop experiments.

value-based agents follow a similar trajectory. Deep spiking Q networks were ported to Intel Loihi and evaluated in closed loop on classic control tasks, where quantized spiking variants were adopted to meet hardware constraints while preserving the distributional target and replay design choices of deep Q learning. These results show that event-

driven execution can be combined with established value-based ingredients without sacrificing the core algorithmic structure [139].

Perception and navigation with asynchronous sensors also benefit from spiking policies. An event enhanced multimodal spiking actor fused laser range measurements with event camera input and used deep RL to improve dynamic obstacle avoidance, outperforming prior methods on moving obstacle scenarios [140]. Reward modulated plasticity (R-STDP) has delivered end to end control on real robots as well. Lane keeping and target tracking controllers trained with R-STDP illustrate the three factor family on physical platforms with neuromorphic vision [141]. Broader evaluations conclude that spiking function approximators can learn complex continuous control when paired with modern deep reinforcement learning algorithms, provided careful tuning of neuron models and simulation hyperparameters [142].

3.3. Applications of Learning-Based Control and Perception for Quadcopters

Having reviewed the main theoretical components regarding SNNs for quadcopter control and how to train them, we can finally move on to emphasizing what specific application of learning-control exist in literature. This chapter reviews the current literature for ANNs and SNNs applied to quadcopter control. The focus is on methods, learning setups, and reported results on real platforms. We first cover ANNs for agile flight, perception, and deployment under tight resource budgets. We then turn to SNNs that leverage event-driven sensing and neuromorphic hardware.

3.3.1. Artificial Neural Networks for Quadcopters

Vision-Based Agile Flight and Racing

End-to-end vision to motor policies have raised the ceiling of autonomous racing by training perception and control together and validating the full stack on physical tracks [120, 143]. Swift combines a learned perception encoder with a policy trained in simulation and improves it with real track data. The system wins multi lap races against human champions using only onboard sensing and compute, which sets a modern benchmark for closed loop racing with learned controllers [120]. A complementary approach imitates a privileged expert in simulation and then transfers zero-shot to diverse outdoor and indoor sites. Mapping raw onboard sensing to short horizon trajectories reduces latency and improves robustness compared to map-then-plan pipelines, which supports tight perception to action coupling for high speed flight [143].

Recent work extends this end-to-end paradigm to direct motor control for time-optimal racing. Ferede et al. train a policy with RL that outputs motor commands and close the reality gap using a learned residual model together with adaptive compensation for thrust and moment errors. The controller outperforms a strong baseline that relies on an inner loop controller for body rates and thrust, with faster laps in simulation and measurable gains in real flight, which demonstrates the potential of end-to-end RL for time-optimal quadcopter flight [6].

Learning-based tracking enables aggressive maneuvers that exceed traditional safety margins. A neural tracker that follows optimal reference trajectories flies loops and barrel rolls with only onboard sensing, which demonstrates precise attitude and thrust control under extreme setpoints [144]. Direct comparisons between model-based control and deep RL in racing tasks show that policies trained with RL can outperform strong optimal-control baselines on time-optimal gate sequences. Analysis points to task objective choice and landscape properties that favour RL for fast progress through gates [7].

Platform support has also matured. Agilicious provides an open research quadcopter with high thrust-to-weight ratio, GPU-accelerated onboard perception, and support for both model-based and learned controllers. The platform tracks trajectories at high speed and performs vision-based acrobatics with onboard inference, which enables reproducible studies of learned control in real flight [145]. Methods that optimise both progress and and camera input in tandem, further improve robustness. Finally perception-aware training shapes policies that maintain visibility while moving fast through clutter, which leads to safer and quicker flight when field-of-view limits dominate [146].

Perception and State Estimation for Fast Navigation

Fast flight relies on sensing and estimation that remain accurate under motion blur, lighting changes, and dynamic obstacles. Event cameras enable obstacle avoidance with millisecond perception-to-control latency by exploiting asynchronous events fused with inertial data. Experiments show onboard dodging of fast moving obstacles that exceed what conventional frame-based pipelines can handle [147]. Shallow policies trained to estimate ego-motion and object motion from events can output collision-avoidance actions directly. EVDodgeNet is trained in simulation

and transfers zero-shot to real flight with strong success rates across shapes and illumination, which validates the value of event streams for fast reactions [148]. Learning-based inertial odometry tailored to racing conditions also advances state estimation. A temporal convolutional model fused with a model-based filter using inertial and thrust signals achieves drift and tracking accuracy competitive with visual–inertial odometry while remaining robust to blur and lighting [149].

Resource-Constrained Deployment on Nano Drones

Neural control can run on micro air vehicles with strict power budgets when models and toolchains are designed for efficient computation. Closed-loop visual navigation with a compact convolutional network runs on a twenty-seven gram platform using an ultra low power microcontroller at milliwatt budgets and multi-hertz rates, which shows that end-to-end policies can fit tiny vehicles [150]. Model compression and distillation extend this approach. Tiny PULP DroNets compress steering and collision predictors by large factors in parameters and operations with similar flight behaviour on PULP microcontrollers, which enables multitasking under tight memory and compute [151]. A simple and robust baseline remains widely used. DroNet maps monocular images to steering and collision probability using imitation from driving data and transfers to micro aerial vehicles in urban scenes, which popularized reactive CNNs for navigation in complex environments [152]. Data collection strategies also matter. Learning from large negative datasets created by deliberate crashes produces policies that avoid obstacles without expert labels, which offers a cost-effective route to robust behaviour [153].

3.3.2. Spiking Neural Networks for Quadcopters

Fully Neuromorphic Vision to Control

A fully neuromorphic pipeline can map raw event streams to control with only spiking computation along the path from vision to action. A recent system learns spiking ego-motion self-supervised on real events and trains a spiking decoding layer with evolutionary search in simulation to output control. Free flight with only neuromorphic processing validates an end-to-end spiking stack for autonomous drones at scale [154].

Neuromorphic low level control and state estimation

Neuromorphic hardware supports ultra low latency feedback and on-chip learning that adapts during flight. Event-driven perception and control implemented as spiking networks run on a neuromorphic processor and achieve high-speed attitude control with learned adaptation, which shows the latency and energy advantages of neuromorphic loops [155]. Minimalist spiking modules replace classical blocks as well. A neuromorphic proportional–integral–derivative controller with fewer than one hundred neurons achieves onboard altitude control on a flying platform with high update frequency, which keeps the control stack fully spiking [156]. Spiking estimation is also feasible onboard the processor. An attitude estimator with roughly 150 neurons trained on flight data runs onboard and reaches accuracy comparable to non-neuromorphic estimators during real flights [157]. Merging estimation and control closes the loop. An end-to-end neuromorphic attitude controller trained by imitation runs at 500 Hz on a micro quadcopter and improves oscillation behaviour with data augmentation, which demonstrates a plausible core for a neuromorphic autopilot [158].

Spiking Perception for High Speed Motion Sensing

Spiking perception models operate directly on asynchronous event streams, which supports fast motion sensing under high dynamics [159, 160]. One line of work trains a convolutional spiking network with surrogate gradients to estimate three degree of freedom angular velocity from events and shows accurate regression on flight data [159]. Another line computes optical flow with spiking architectures on event datasets and attains competitive accuracy with strong efficiency, which enables spiking pipelines for agile odometry and anticipatory control [160].

4

Research Formulation

4.1. Motivation

Quadcopters are highly agile vehicles that require fast and reliable control in order to navigate through complex environments at high speed. Within MAVLab, bioinspired design has driven numerous innovations. The DelFly Nimble, a tailless flapping-wing micro air vehicle, shown in Figure 4.1, demonstrated exceptional agility inspired by fruit fly escape maneuvers and revealed aerodynamic strategies such as torque coupling used in rapid banked turns [161].

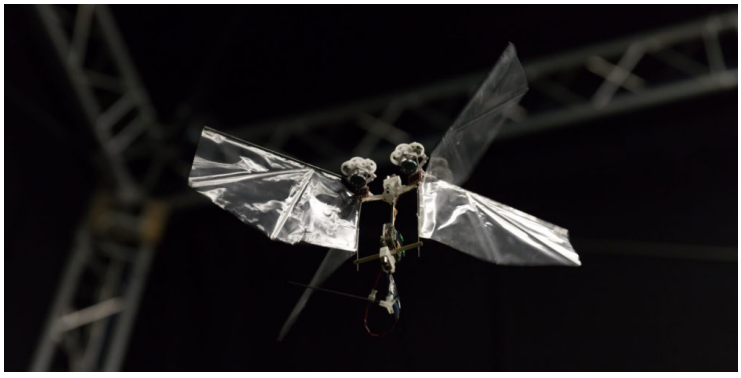


Figure 4.1: DelFly Nimble: a tailless flapping-wing micro air vehicle developed at the TU Delft MAVLab [161].

Building on this tradition, MAVLab has advanced neuromorphic implementations for autonomous flight. A recent study demonstrated a fully neuromorphic drone that coupled an event-based vision system with a SNN running on the Intel Loihi neuromorphic processor, achieving onboard closed-loop control at 200 Hz while consuming orders of magnitude less energy than GPU-based systems [154].

In parallel, MAVLab has also shown the power of ANNs for high-speed quadcopter navigation. End-to-end reinforcement learning for time-optimal quadcopter flight demonstrated that neural policies could autonomously race through a track at speeds approaching the theoretical optimum [6]. Building on this, the One Net to Rule Them All framework showed that a single end-to-end policy could generalize across diverse flight and perception tasks [121]. Together, these studies established strong ANN baselines and confirmed that neural controllers can push the limits of agile autonomous flight.

Although ANNs have achieved remarkable results, their reliance on dense computations and continuous activations makes them energy demanding and less suitable for embedded aerial platforms. SNNs, by contrast, operate with sparse and event-driven communication and are ideally suited for neuromorphic hardware. The motivation for this thesis therefore arises directly from MAVLab's two lines of research: bioinspired and neuromorphic approaches on the one hand, and high-performance ANN-based control on the other.

4.2. Research Objective

Although SNNs have gained increasing attention in recent years, their application in reinforcement learning for control tasks has so far remained narrow. Most existing studies rely on ANN-to-SNN conversion or hybrid models and have not shown that spiking networks can be trained directly to solve demanding continuous control problems as discussed in Section 3.3.2. In particular, no studies to date have investigated the use of SNNs for high-speed quadcopter control. This represents a critical gap in the literature, since quadcopters place strict requirements on real-time stability, performance, precision, and execution efficiency. The aim of this thesis is therefore to demonstrate that a SNN controller can be trained with reinforcement learning to control a quadcopter in high-speed flight, while maintaining comparable performance to an ANN baseline. By doing so, this work seeks to establish whether spiking networks can serve as viable alternatives to artificial ones for agile drone navigation. The main research objective is summarized as follows:

To develop and evaluate a spiking neural network controller for high-speed quadcopter flight, with a focus on spiking network design, training methodology, and benchmarking against ANN baselines in terms of control performance and stability

4.3. Research Questions

To address the research objective, the study is guided by two central questions. The first question focuses on the fundamental challenge of designing and training spiking networks that can learn to fly a quadcopter. Spiking networks process information through discrete temporal events, which makes learning continuous control tasks non-trivial. The central issue is whether spiking networks can be constructed and trained in such a way that they achieve stable and high-performance control comparable to a proven ANN baseline. This requires exploring network architectures and training methods that can successfully leverage spiking dynamics while maintaining reliable real-time behaviour.

Research Question 1

What spiking network architectures and training methods enable stable and high-performance quadcopter control, and how do these controllers compare to a known ANN baseline in terms of performance?

The second question addresses the broader influence of design and training parameters on spiking control performance. The structure of spiking networks, including neuron models, coding strategies, and temporal integration, can have an influence on stability and responsiveness during flight. Training choices such as optimization setup, rollout length, or spike averaging may further shape how well the network generalizes and how efficiently it operates. Understanding how these factors impact performance relative to an ANN baseline is crucial for assessing whether spiking controllers offer practical benefits or introduce new limitations. By systematically examining these trade-offs, the study aims to clarify the conditions under which SNNs can serve as viable alternatives for high-speed quadcopter control.

Research Question 2

How do the design and training parameters of spiking neural networks affect quadcopter control performance compared to a known ANN baseline?

4.4. Research Methodology

To answer the first question, several complementary approaches will be explored to obtain spiking controllers capable of high-speed quadcopter flight. The first avenue is ANN-to-SNN conversion, where a trained ANN is mapped onto a spiking network to test whether suitable performance can be retained. This approach is chosen as the starting point because ANN-to-SNN conversion is a well-documented and widely used method in the SNN literature. It provides a reliable baseline for evaluation and allows us to quickly determine whether spiking dynamics can approximate

the behaviour of an already successful ANN policy. Performance may also be further improved with additional supervised learning methods to match the output of the SNN to the ANN.

The second avenue is reinforcement learning with PPO, where both the spiking network and the training algorithm must be adapted to enable effective policy optimization. PPO is selected because it is a robust, stable, and well-known algorithm for continuous control, and it has become a standard method in reinforcement learning research. However, there are almost no prior studies that apply PPO directly to spiking networks, which makes this approach riskier but also potentially more rewarding. If successful, it would provide one of the first demonstrations of end-to-end reinforcement learning with PPO for spiking policies in high-speed quadcopter control.

Different methods will be tested for their suitability in learning the task, with the most successful approach ultimately chosen as the main method for further analysis. Across these approaches, the performance of spiking networks will be compared to ANN baselines trained under identical conditions, ensuring a fair and systematic benchmark.

A caveat of the second question is that it assumes a positive outcome for the first: namely, that training spiking controllers for high-speed quadcopter flight is feasible. The choice of parameters to analyze will also depend strongly on the selected architecture and learning method. In addition, this question implicitly assumes that a strategy that works in simulation can also transfer to real flight. If this proves to be the case, the scope of the research can be extended to include conclusions about real-world flight performance. Nonetheless, even in the absence of hardware validation, generalizations can be made about the methodology for addressing the second question. It will be tackled by systematically varying design and training parameters of the spiking controllers. Structural factors such as neuron models, spike coding strategies, and temporal integration mechanisms will be investigated to determine their impact on flight stability and control quality. Training-related factors such as optimization setup, rollout length, and spike coding will also be varied to assess their effect on learning dynamics and final performance. These experiments will provide insight into how design and training parameters influence stability, robustness, execution frequency, and overall performance, thereby clarifying the trade-offs between spiking and artificial networks.

All experiments will first be carried out in simulation, where conditions and disturbances can be controlled and repeated consistently. Once promising spiking controllers are identified, they will be deployed on real quadcopter hardware for validation in practical flight scenarios. This staged approach, from ANN-to-SNN conversion and supervised training, to direct reinforcement learning, to parameter analysis and hardware validation, ensures that the methodology addresses both the feasibility of training spiking controllers and their potential advantages over ANN baselines.

4.5. Research Planning

The research will be carried out in several phases that build on each other. The first phase is dedicated to a thorough literature review, covering neuromorphic computing, SNNs, ANNs, and reinforcement learning, with a particular focus on training methods for SNNs and their application to drones or other robotic control tasks. This review establishes the theoretical foundation and clarifies the state of the art.

After the literature study, the project moves to testing and implementation of the two main approaches identified in the methodology. The first is ANN-to-SNN conversion. This method will be implemented and tested on the gate navigation task to evaluate whether a converted network can deliver satisfactory performance. In addition, possible extensions will be explored, such as augmenting the conversion with supervised learning techniques or other training refinements to improve stability and accuracy. The second approach is direct reinforcement learning of spiking networks using PPO. While more challenging and uncertain, this method also has the potential for higher impact, as little to no prior work has investigated PPO training of fully spiking policies. For this reason, the research plan begins with the more established conversion-based approach and only then advances to the more experimental reinforcement learning method.

Once both approaches have been tested, a decision will be made regarding which method provides the best performance in simulation and whether an SNN can in fact learn to fly with results comparable to the ANN baseline. If successful, the next steps will focus on adapting the spiking policy for transfer from simulation to reality. This includes addressing the sim-to-real gap and ensuring that the controller can run efficiently on embedded hardware, culminating in real-world quadcopter flight tests, and analysis of performance of the SNN in comparison to the baseline ANN.

As one of the later stages, evaluation of the gathered data on the SNN and ANN will be performed which will then lead into the final reporting section for the thesis. A timeline of the planned phases is given in the Gantt chart in Figure 4.2. The tasks have been split up into the 4 nominal stages according to the MSc Thesis guidelines, namely Kick-off phase, Mid-term phase and Green-light phase and Finalisation Phase

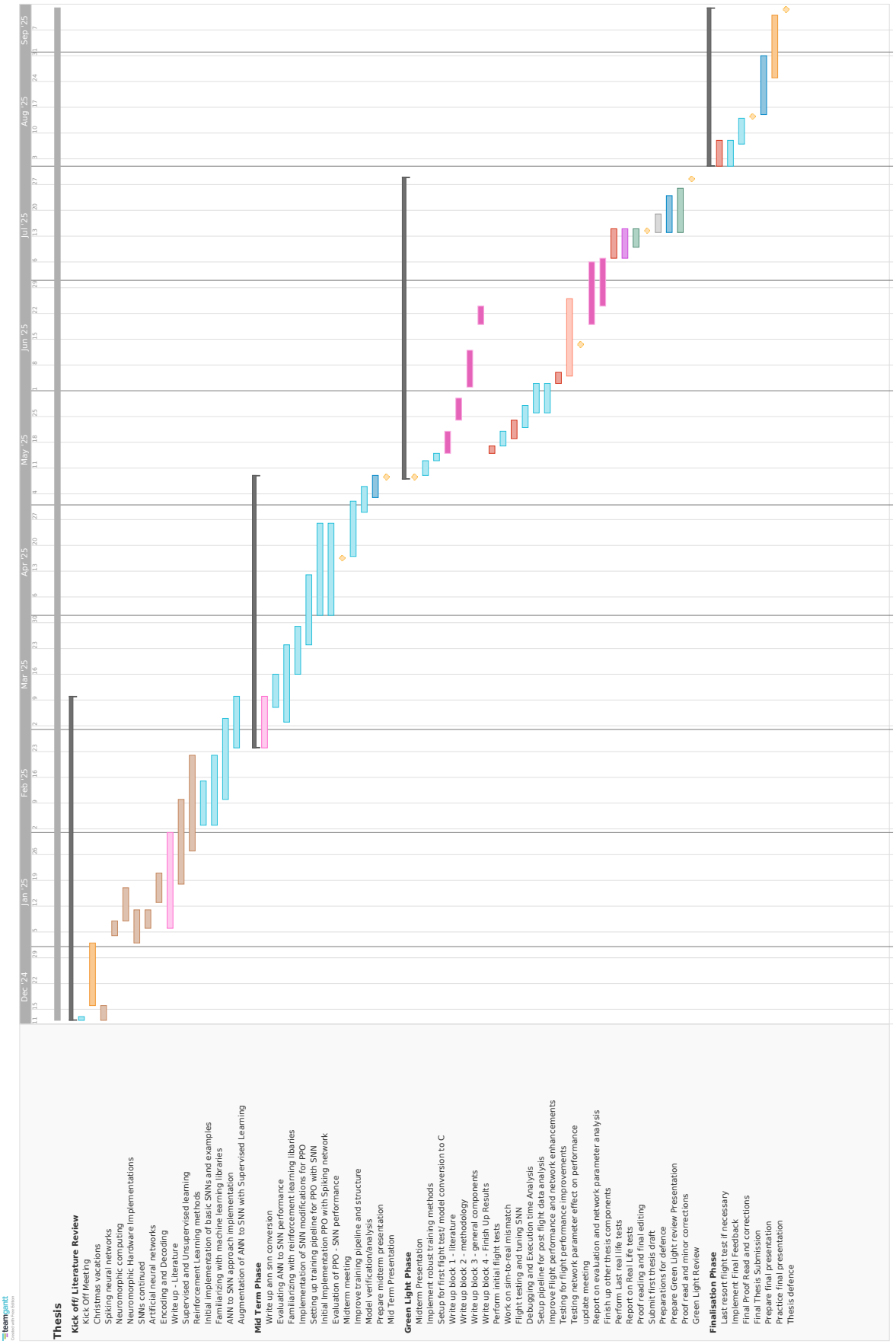


Figure 4.2: Gantt Chart for the thesis timeline showing all major tasks and milestones until the defence. The main colours indicate the following, Brown:Literature review, Pink: Write-up, Blue: Programming/Analysis, Red: Flight Tests, Green: Proof reading.

5

Preliminary Work

5.1. ANN-to-SNN and Supervised Learning Methods

Before adopting reinforcement learning with PPO as the main method, several conversion-based approaches were explored. These followed the well-established line of ANN-to-SNN conversion methods, which exploit the equivalence between ReLU activations and the firing rate of leaky integrate-and-fire (LIF) neurons under rate coding [43]. In theory, this correspondence enables a direct transfer of weights from a trained ANN into an SNN with the same architecture, given that the LIF neurons are functionally equivalent. Given the exploratory nature of this stage, not all approaches were fully developed or explored in depth, leaving several open research directions for the future. Because the emphasis was on finding a minimum viable approach to solving the problem of SNN control for high-speed drones, methods that showed little potential after significant experimentation were set aside in favor of more promising directions.

Direct weight injection

The first approach was to copy the trained weights of an ANN baseline policy into an SNN policy of identical structure. The weights of the fully connected layers of the ANN were mapped directly onto the corresponding layers of the SNN. Unlike ReLU, LIF neurons do not have an explicit bias term. Instead, this bias is absorbed as a constant input current. Input values were therefore passed to the LIF neurons through direct current injection, with the bias added as an offset to the input. The LIF neurons were parameterized with a threshold $U_{threshold} = 1$ and a leak parameter $\beta = 0.999$.

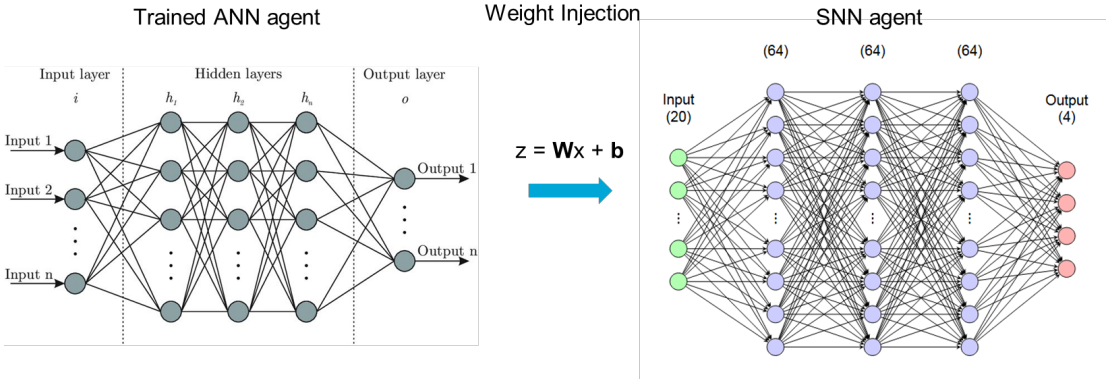
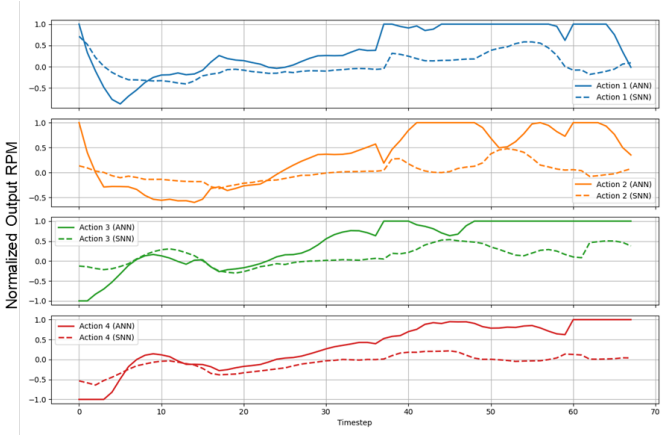


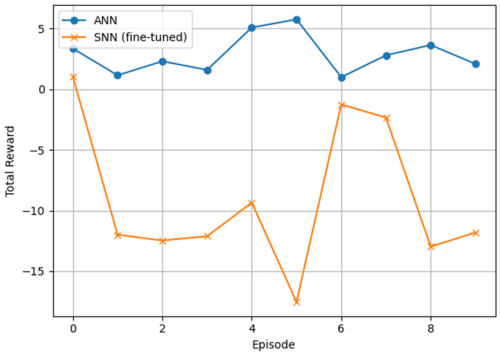
Figure 5.1: Representation of direct weight injection from a trained ANN into an SNN with the same architecture using rate decoding [162].

The expectation was that the LIF neurons with rate decoding would replicate the ANN activations, leading to matched outputs between the two models. Initial evaluation, as shown in Figure 5.2, indicated that the outputs aligned only for a short period before diverging toward the end of the episode. Each episode lasted 70 timesteps, and with a control update rate of 100 Hz this corresponded to less than a second, as the episode terminated when a crash occurred. When deployed in the quadcopter environment, the converted SNN achieved very low rewards and crashed

rapidly for most episodes, confirming that direct weight transfer was insufficient on its own. The drone almost always crashed immediately, although it consistently moved toward the gate. In the best cases, only a single gate was crossed. The exact reason behind the discrepancy between the SNN and ANN outputs despite the functional equivalence between the ReLU and the LIF neurons is still unknown and remains a point of future investigation. A likely cause is the treatment of the bias term, which may deviate from the ideal mapping since the way a constant input current shifts LIF activation differs from how a bias term affects ReLU activations.



(a) Normalized output RPM for the ANN and SNN with weight injection for a single episode over several timesteps until termination.



(b) Total accumulated reward for trained ANN and SNN after weight injection over 10 episodes.

Figure 5.2: Initial comparison results of the SNN with **injected weights** from a trained ANN model.

Fine-tuning output heads

A second attempt drew inspiration from supervised learning. After weight injection, only the actor output heads of the SNN were fine-tuned to minimize the error between the ANN and SNN outputs over longer time horizons. The training objective was the mean squared error (MSE) between the motor RPM predictions of the two networks,

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N} \sum_{i=1}^{N} \left(y_i^{\text{ANN}} - y_i^{\text{SNN}} \right)^2, \tag{5.1}$$

where y_i^{ANN} and y_i^{SNN} denote the motor RPM outputs of the ANN and SNN for sample i, and N is the batch size. Optimization was carried out using Adam [163], which maintains exponential moving averages of the first and second moments of the gradients. For parameters θ, gradient $g_t = \nabla_\theta \mathcal{L}_t$ at step t, and learning rate α, the update rules are

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \tag{5.2}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2, \tag{5.3}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \tag{5.4}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}, \tag{5.5}$$

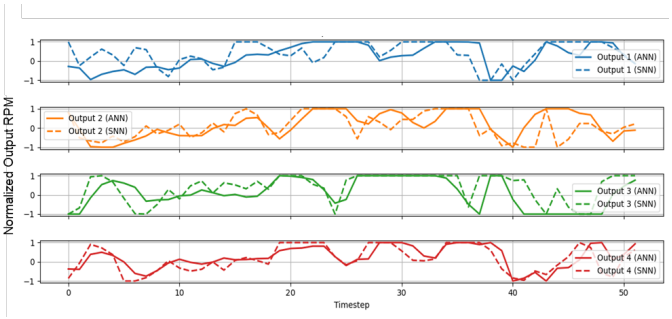
$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \tag{5.6}$$

where m_t and v_t are biased moment estimates, $\hat{m}_t$ and $\hat{v}_t$ are their bias-corrected forms, and β_1, β_2 are decay coefficients.

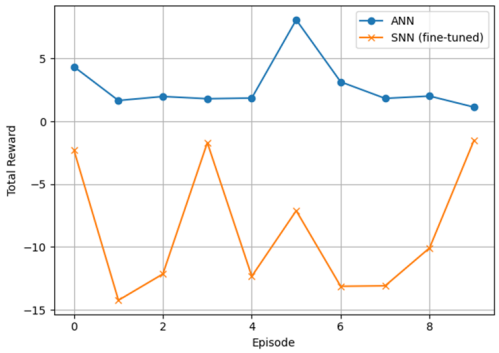
Figure 5.3 shows that this adjustment improved output RPM alignment between the two policies. Cumulative rewards per episode were slightly higher than with direct injection, but performance remained unreliable. At this level of reward, the simulated models could typically fly through one or two gates at most before crashing.

Full-network fine-tuning

To allow broader adaptation, the entire network was fine-tuned instead of only the output heads. The setup again used the Adam optimizer with MSE loss between ANN and SNN motor outputs. As shown in Figure 5.4, this strategy

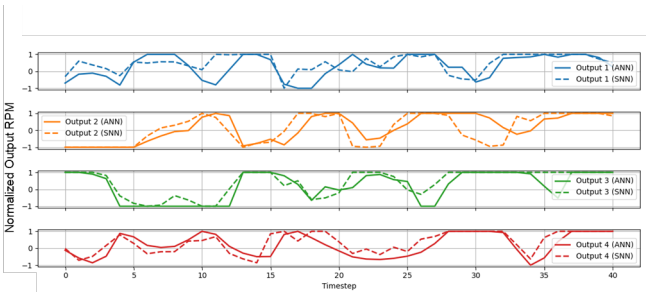


(a) Normalized output RPM for the ANN and SNN with weight injection and output head fine-tuning for a single episode.

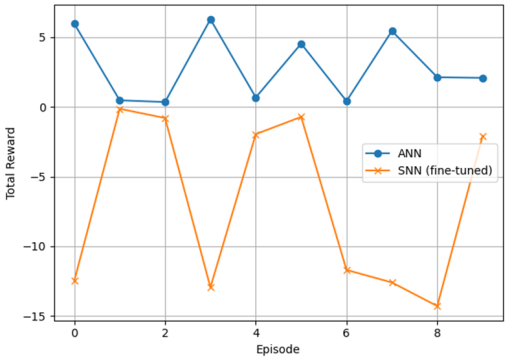


(b) Total accumulated reward for trained ANN and SNN after weight injection and output head fine-tuning over 10 episodes.

Figure 5.3: Comparison results of the SNN with **injected weights and output head fine-tuning** from a trained ANN model.



(a) Normalized output RPM for the ANN and SNN with weight injection and full network fine-tuning for a single episode.



(b) Total accumulated reward for trained ANN and SNN after weight injection and full network fine-tuning over 10 episodes.

Figure 5.4: Comparison results of the SNN with **injected weights and full network fine-tuning** from a trained ANN model.

offered only marginal improvement compared to output-head fine-tuning. Output RPMs aligned similarly, and average rewards were only slightly higher, with some still close to zero. In the best cases, the SNN managed to pass three or four gates when simulated, but struggled to stay airborne afterwards.

Transition toward reinforcement learning

The supervised conversion approaches confirmed that ANN-to-SNN transfer was feasible for short horizons but did not provide stable long-term control in a closed-loop setting. The likely cause of poor performance over longer horizons was the accumulation of output RPM errors. Since the SNN did not perfectly match the outputs of the trained ANN, it incurred a small error at each timestep. While this error was tolerable for the first few gates, it compounded over time, and the mismatch between SNN and ANN grew. Once the SNN entered states that were not represented in its training data, performance degraded sharply and crashes followed. Cumulative rewards were sometimes lower than with output-head training alone, and the computational cost of simulating multiple episodes further limited the experiments.

This analysis also highlighted a key limitation of supervised learning, namely that the objective function, in this case the loss, was not representative of actual task performance. Optimizing the loss did not necessarily translate into better control. For this reason, analyzing output RPMs was less informative than simulating the trained agents directly. With this preliminary analysis we also partially addressed the first research question, as the initial experiments showed

that ANN-to-SNN conversion with supervised learning was not promising enough for a fully spiking controller for high-speed quadcopters.

The instability, frequent crashes, and computational demands motivated a pivot toward direct reinforcement learning with PPO. This method does not rely on an ANN teacher but instead optimizes the spiking policy directly through interaction with the environment, which became the main research direction of this thesis.

Part III

Limitations and Recommendations for Future Research

6

Limitations

Although the spiking policy trained with PPO completes the high speed racing task, the strength of the conclusions is shaped by the way the problem was framed, the measurements that were taken, and the hardware that executed the controller. Reinforcement learning outcomes can vary across random seeds, which affects how much confidence we can place in single run curves and final scores [164]. Energy efficiency is a central motivation for spiking computation and must be verified with on chip measurements on neuromorphic hardware rather than inferred from theory alone [3]. The specific choices that made training stable, such as LIF neurons, surrogate gradients, and rate decoding, also influence expressiveness and latency [47]. The sections below provides and overview of the limitations of the study, which motivate many of the future reccommendations in Chapter 7

Methodological Limitations

The evaluation is based on a limited number of training runs due to the long training times of SNN models. As a result, performance curves and convergence results are derived from single representative runs rather than averaged across multiple seeds. In reinforcement learning, variance across seeds can be significant, and more repetitions would be required for statistically robust conclusions [164].

Architectural and Model

The SNN architecture is constrained to three hidden layers of 64 LIF neurons each. This shallow feed-forward structure may lack the representational capacity of deeper or recurrent networks, limiting long-horizon planning. Moreover, the use of rate decoding over a fixed number of cycles introduces quantization effects in the latent output. For example, with $C = 5$ cycles, each neuron's average firing rate is limited to six possible values $\{0, \frac{1}{5}, \ldots, 1\}$, which can severely impact the resolution of the subsequently generated motor output RPM. In addition, only a narrow range of cycle counts was tested, which limits the generality of the conclusions on the trade-off between control resolution and computational cost. The conversion pipeline also did not explicitly optimize for energy efficiency, which is a central motivation for deploying SNNs on neuromorphic hardware [3].

Training and Evaluation

Training time scales linearly with the number of cycles due to repeated forward passes per input. The 10-cycle model, for example, required over ten hours to train to convergence, compared to 33 minutes for the ANN baseline. This severely restricted the extent of hyperparameter tuning, such as exploration of learning rates, discount factors, or PPO-specific clip ranges, which may have yielded higher-performing policies. Furthermore, while PPO is stable and well-documented, it is not clear whether alternative RL algorithms (e.g., TD3 or SAC) could be more efficient or well suited for spiking networks.

Hardware and implementation

The current C implementation of the SNN defines layers and neuron dynamics manually, without optimised vectorized operations. This results in slower execution compared to what could be achieved with matrix multiplication libraries on embedded processors. Inference-time measurements were only inferred indirectly through observed control update frequencies, not profiled at the function-call level, making it difficult to pinpoint computational bottlenecks. Most importantly, no neuromorphic hardware was employed in this study. Thus, while the energy efficiency of SNNs is a primary motivation, the claimed efficiency remains theoretical without direct measurement on event-driven processors such as Loihi.

7

Future Work and Research Directions

This research identified several promising directions for extending the study of spiking neural networks in quadcopter control. The recommendations can be grouped into hardware optimization, neuromorphic deployment, RL design, neuron and encoding analysis, architectural changes, and further investigation of ANN-to-SNN conversion.

Hardware Optimisation and Performance Analysis

In this work the spiking policy was trained in Python, but deployment on the drone required manual translation into C code. Each layer was implemented explicitly with scalar multiplications, after which the code was compiled and executed onboard. A clear next step is to optimise this implementation by using vectorised or matrix-based operations, which are expected to reduce inference latency and improve throughput on embedded processors.

Complementary to this, an in-depth analysis of execution time on the hardware is needed. The current evaluation relied mostly on observed update frequencies from actual flight logs, which should in theory provide an accurate estimate of the actual inference time of the network. However, direct timing measurements at the processor level, implemented in the C code, would yield more accurate results. Such measurements should also examine the effect of cycle count on execution time. Although a linear scaling is expected, the results from real flights point to the fact that inference time saturates after a certain number of cycles of the network, as increasing cycle count did not increase the observed update frequency. This behaviour may be linked to the scheduling strategy of the onboard processor, which motivates a more detailed study of scheduler behaviour and its impact on the consistency of SNN execution.

Deployment on Neuromorphic Hardware

Another important and interesting next step is to adapt the controller for deployment on neuromorphic processors such as Loihi or TrueNorth [3, 17]. These platforms impose strict resource constraints and typically do not support floating-point arithmetic. This makes quantisation of weights and biases necessary. Both as an initial technique, post-training quantisation and quantisation-aware training should be systematically evaluated to determine their effect on task performance. In addition, further adaptations may be required, such as limiting connectivity and adapting spike timing resolution to match the hardware. Deployment on neuromorphic devices would make it possible to directly measure energy consumption per inference and to validate the anticipated efficiency benefits of neuromorphic computation. To maximise these gains, it is advisable to design and train spiking networks with energy efficiency as a primary objective, thereby aligning the controller with the strengths of neuromorphic platforms. This would advance the research in a straight direction towards the initial motive and inspiration for this thesis.

Reinforcement Learning Setup, Hyperparameters and Reward Design

The RL framework also presents opportunities for improvement. First of all, the reward function used in this study did not fully capture the true task objective, which is to maximise the number of gates passed within a fixed time interval, in other words, a faster completion of the track. Future work should investigate reward shaping strategies that explicitly encourage task completion and high-speed flight, given that our reward function focused largely on rewarding distance towards the next target gate. Beyond reward design, the influence of hyperparameters of PPO on training stability and convergence should be further assessed. This is also involves analysing the training parameter curves to better understand how the algorithm can be optimized to produce better performance for the same model. It is also recommended, as with any RL approach, that results should be averaged over a larger number of training runs to reduce variance and provide statistically reliable benchmarks of performance.

Neuron Parameters and Coding Methods

The dynamics of spiking neurons strongly depend on their parametrization. Future work should examine how variations in the LIF threshold, bias, and leak affect training performance and real flight tests. Small adjustments in these values may lead to significant differences in convergence speed or control precision. In addition, alternative encoding methods should be explored. The choice of encoding has a direct impact on spike efficiency, latency, and accuracy, and systematic comparisons could reveal trade-offs that were not captured by the current setup.

Architectural Explorations

Architectural modifications represent another promising avenue. Increasing the depth or width of the spiking network may allow for richer representations of the task dynamics, though this comes at the cost of higher inference complexity. Hybrid actor-critic structures, where the actor is implemented as an SNN and the critic as a conventional ANN, could provide a balance between efficiency and learning stability. More generally, exploring architectures that are explicitly optimised for energy efficiency could be advantageous, especially when considering eventual deployment on neuromorphic processors.

Further Analysis of ANN-to-SNN Conversion

Finally, the ANN-to-SNN conversion approach explored in the early stages of this work remains an open research question. In principle, the functional equivalence between rate-decoded LIF neurons and ReLU activations suggests that injected weights should produce identical outputs. However, this study observed consistent mismatches between the ANN baseline and the converted SNN. The source of this discrepancy is not yet understood. Future work should perform a systematic analysis of the factors contributing to this error, such as the leak parameter, encoding precision, or numerical approximations in the conversion pipeline. More comprehensive experiments are needed to determine whether ANN-to-SNN conversion can be established as a reliable alternative to direct training for spiking controllers.

Part IV

Closure

8

Conclusion

This thesis presented, for the first time, a fully *spiking* actor–critic controller for continuous quadcopter flight, trained end-to-end with reinforcement learning to perform high-speed, agile manoeuvrers. In simulation, the spiking policy learned to control the drone through an eight-shaped gate navigation task, and it successfully transferred to real-world flight tests while preserving performance. Notably, the spiking controller's flight performance was superior to a standard artificial neural network based controller, provided by Ferede et. al[6], despite being trained on equal conditions. This result indicates that a spiking agent can achieve the effectiveness of conventional drone controllers even on demanding tasks. This marks a key contribution: despite relying on discrete spike signals throughout the network, our spiking agent matched the agility and performance expected from state-of-the-art artificial neural network policies [6]. We thereby demonstrate that neuromorphic control can attain high-speed flight performance on par with traditional neural networks.

The answer and link back to our initial research questions are embedded in the specific set of coupled design choices for the spiking policy. For RQ1 on architectures and training methods that enable high speed flight, we used a compact feed-forward spiking actor-critic with leaky integrate and fire neurons and layer sizes matched to a proven artificial neural network baseline. Training used Proximal Policy Optimisation. Surrogate gradients smoothed the spike non-linearity in the spiking neurons to preserve gradient flow during training through backpropagation. Inputs were passed as state vectors and outputs were decoded as short window spike rates to produce continuous motor commands. These decisions produced stable learning from scratch without artificial neural network to spiking neural network conversion and yielded a policy that matched the agility of the artificial neural network reference in simulation and real flight [8, 47, 43, 44].

For RQ2 on how design and training parameters influence performance, we quantified how the spike integration window used for decoding actions governs the latency and control precision trade off by varying the number of spike cycles used to decode each control output. Each action is formed by integrating spikes over a short window of cycles, so the cycle count sets the effective rate resolution and therefore the granularity of the latent action space and by consequence the output RPM. Using few cycles minimized inference latency and computational cost but produced coarser decoded outputs and a drop in training and flight performance. Increasing the cycle count improved training performance, output resolution, and reward, at the cost of added latency and a lower achievable control update frequency on embedded hardware. A moderate cycle count balanced reward, average velocity, and gates passed, which yields a practical rule for configuring real time spiking control on resource constrained platforms. Together these choices inform how to tune latency and performance for a given task [8, 43, 44, 6].

In conclusion, this work bridges the gap between neuromorphic computing and advanced quadcopter control and robotics. We have demonstrated that brain-inspired spiking neural networks, trained with modern reinforcement learning techniques, can control a micro aerial vehicle with superior performance to conventional ANN controllers. This achievement opens the door to leveraging neuromorphic processors for real-time, on-board flight control—potentially enabling a new generation of highly energy-efficient, high-performance autonomous drones. By showing that an SNN can meet the demands of high-speed flight while offering low-power, high-frequency operation [154], we pave the way for generalizing this approach to other robotic platforms and scenarios, where efficiency and agility are paramount.

References

[1] OpenAI. *ChatGPT Image Generation: Scene description*. AI-generated image; prompt: "Generate an realistic portrait image for a report of a drone and a neuromorphic chip". Aug. 31, 2025.

[2] Jaime Sevilla et al. "Compute Trends Across Three Eras of Machine Learning". en. In: *2022 International Joint Conference on Neural Networks (IJCNN)*. arXiv:2202.05924 [cs]. July 2022, pp. 1–8. DOI: 10.1109/IJCNN55064.2022.9891914. URL: http://arxiv.org/abs/2202.05924 (visited on 09/09/2025).

[3] Mike Davies et al. "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning". en. In: *IEEE Micro* 38.1 (Jan. 2018), pp. 82–99. DOI: 10.1109/MM.2018.112130359. URL: https://ieeexplore.ieee.org/document/8259423/.

[4] Luca Zanatta et al. "Exploring spiking neural networks for deep reinforcement learning in robotic tasks". en. In: *Scientific Reports* 14.1 (Dec. 2024). Publisher: Nature Publishing Group, p. 30648. DOI: 10.1038/s41598-024-77779-8. URL: https://www.nature.com/articles/s41598-024-77779-8.

[5] Emre O. Neftci et al. *Surrogate Gradient Learning in Spiking Neural Networks*. en. arXiv:1901.09948 [cs]. May 2019. DOI: 10.48550/arXiv.1901.09948. URL: http://arxiv.org/abs/1901.09948.

[6] Robin Ferede et al. "End-to-end neural network based optimal quadcopter control". In: *Robotics and Autonomous Systems* 172 (Feb. 2024), p. 104588. DOI: 10.1016/j.robot.2023.104588. URL: https://www.sciencedirect.com/science/article/pii/S0921889023002270.

[7] Yunlong Song et al. "Reaching the Limit in Autonomous Racing: Optimal Control versus Reinforcement Learning". In: *Science Robotics* 8.82 (Sept. 2023). arXiv:2310.10943 [cs], eadg1462. DOI: 10.1126/scirobotics.adg1462. URL: http://arxiv.org/abs/2310.10943.

[8] John Schulman et al. *Proximal Policy Optimization Algorithms*. arXiv:1707.06347 [cs]. Aug. 2017. DOI: 10.48550/arXiv.1707.06347. URL: http://arxiv.org/abs/1707.06347.

[9] Allen Nutman et al. "Rapid emergence of life shown by discovery of 3,700-million-year-old microbial structures". en. In: (Jan. 2016). Publisher: University of Wollongong. DOI: 10.1038/nature19355']. URL: https://ro.uow.edu.au/articles/journal_contribution/Rapid_emergence_of_life_shown_by_discovery_of_3_700-million-year-old_microbial_structures/27787500/1.

[10] *Codex on the Flight of Birds*. Italian. place of publication not identified: publisher not identified, Jan. 1505. URL: https://www.loc.gov/resource/gdcwdl.wdl_19477/.

[11] *High Speed Train Inspired by the Kingfisher — Innovation — AskNature*. en-US. URL: https://asknature.org/innovation/high-speed-train-inspired-by-the-kingfisher/.

[12] Carver A. Mead et al. "A silicon model of early visual processing". In: *Neural Networks* 1.1 (Jan. 1988), pp. 91–97. DOI: 10.1016/0893-6080(88)90024-X. URL: https://www.sciencedirect.com/science/article/pii/089360808890024X.

[13] Wolfgang Maass. "Networks of spiking neurons: The third generation of neural network models". en. In: *Neural Networks* 10.9 (Dec. 1997), pp. 1659–1671. DOI: 10.1016/S0893-6080(97)00011-7. URL: https://linkinghub.elsevier.com/retrieve/pii/S0893608097000117.

[14] Wulfram Gerstner et al. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge: Cambridge University Press, 2002. DOI: 10.1017/CBO9780511815706. URL: https://www.cambridge.org/core/books/spiking-neuron-models/76A3FC77EC2D24CDD91E29EBB23ADB0B.

[15] Xiwen Luo et al. "Encrypted Spiking Neural Networks Based on Adaptive Differential Privacy Mechanism". en. In: *Entropy* 27.4 (Apr. 2025). Publisher: Multidisciplinary Digital Publishing Institute, p. 333. DOI: 10.3390/e27040333. URL: https://www.mdpi.com/1099-4300/27/4/333.

[16] Giacomo Indiveri et al. “Memory and information processing in neuromorphic systems”. en. In: *Proceedings of the IEEE* 103.8 (Aug. 2015). arXiv:1506.03264 [cs], pp. 1379–1397. DOI: 10.1109/JPROC.2015.2444094. URL: http://arxiv.org/abs/1506.03264.

[17] Filipp Akopyan et al. “TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.10 (Oct. 2015), pp. 1537–1557. DOI: 10.1109/TCAD.2015.2474396. URL: https://ieeexplore.ieee.org/document/7229264.

[18] C. Mead. “Neuromorphic electronic systems”. In: *Proceedings of the IEEE* 78.10 (Oct. 1990), pp. 1629–1636. DOI: 10.1109/5.58356. URL: https://ieeexplore.ieee.org/document/58356.

[19] Jamal Lottier Molin et al. “Neuromorphic Circuits and Systems: From Neuron Models to Integrate-and-Fire Arrays”. en. In: *Handbook of Neuroengineering*. Springer, Singapore, 2023, pp. 1455–1480. DOI: 10.1007/978-981-16-5540-1_42. URL: https://link.springer.com/rwe/10.1007/978-981-16-5540-1_42.

[20] K.A. Boahen. “Point-to-point connectivity between neuromorphic chips using address events”. en. In: *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* 47.5 (May 2000), pp. 416–434. DOI: 10.1109/82.842110. URL: http://ieeexplore.ieee.org/document/842110/.

[21] Patrick Lichtsteiner et al. “A 128$\times$128 120 dB 15 μs Latency Asynchronous Temporal Contrast Vision Sensor”. en. In: *IEEE Journal of Solid-State Circuits* 43.2 (2008), pp. 566–576. DOI: 10.1109/JSSC.2007.914337. URL: http://ieeexplore.ieee.org/document/4444573/.

[22] Paul A. Merolla et al. “A million spiking-neuron integrated circuit with a scalable communication network and interface”. In: *Science* 345.6197 (Aug. 2014). Publisher: American Association for the Advancement of Science, pp. 668–673. DOI: 10.1126/science.1254642. URL: https://www.science.org/doi/10.1126/science.1254642.

[23] Steve B. Furber et al. “The SpiNNaker Project”. In: *Proceedings of the IEEE* 102.5 (May 2014), pp. 652–665. DOI: 10.1109/JPROC.2014.2304638. URL: https://ieeexplore.ieee.org/document/6750072.

[24] Christian Pehle et al. *The BrainScaleS-2 accelerated neuromorphic system with hybrid plasticity*. arXiv:2201.11063 [cs]. Feb. 2022. DOI: 10.48550/arXiv.2201.11063. URL: http://arxiv.org/abs/2201.11063.

[25] Warren S. McCulloch et al. “A logical calculus of the ideas immanent in nervous activity”. en. In: *The bulletin of mathematical biophysics* 5.4 (Dec. 1943), pp. 115–133. DOI: 10.1007/BF02478259. URL: https://doi.org/10.1007/BF02478259.

[26] F. Rosenblatt. “The perceptron: A probabilistic model for information storage and organization in the brain”. In: *Psychological Review* 65.6 (1958). Place: US Publisher: American Psychological Association, pp. 386–408. DOI: 10.1037/h0042519.

[27] Perceptron Mitchell p87 Machine Learning. *Nederlands: Rosenblatt’s Perceptron*. Oct. 2012. URL: https://commons.wikimedia.org/wiki/File:Rosenblattperceptron.png.

[28] Marvin Minsky et al. *Perceptrons: An Introduction to Computational Geometry*. en. The MIT Press, Sept. 2017. DOI: 10.7551/mitpress/11301.001.0001. URL: https://direct.mit.edu/books/monograph/3132/PerceptronsAn-Introduction-to-Computational.

[29] Jon Agar. “What is science for? The Lighthill report on artificial intelligence reinterpreted”. en. In: *The British Journal for the History of Science* 53.3 (Sept. 2020), pp. 289–310. DOI: 10.1017/S0007087420000230. URL: https://www.cambridge.org/core/journals/british-journal-for-the-history-of-science/article/what-is-science-for-the-lighthill-report-on-artificial-intelligence-reinterpreted/61B13B32988D6A8C58CF8AADD4777789.

[30] Zhen “Leo” Liu. “Deep Learning”. en. In: *Artificial Intelligence for Engineers: Basics and Implementations*. Ed. by Zhen ”Leo” Liu. Cham: Springer Nature Switzerland, 2025, pp. 191–220. DOI: 10.1007/978-3-031-75953-6_8. URL: https://doi.org/10.1007/978-3-031-75953-6_8.

[31] Kunihiko Fukushima. “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”. en. In: *Biological Cybernetics* 36.4 (Apr. 1980), pp. 193–202. DOI: 10.1007/BF00344251. URL: http://link.springer.com/10.1007/BF00344251.

[32] J J Hopfield. “Neural networks and physical systems with emergent collective computational abilities.” In: *Proceedings of the National Academy of Sciences of the United States of America* 79.8 (Apr. 1982), pp. 2554–2558. DOI: 10.1073/pnas.79.8.2554. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC346238/.

[33] Zachary C. Lipton et al. *A Critical Review of Recurrent Neural Networks for Sequence Learning*. arXiv:1506.00019 [cs]. Oct. 2015. DOI: 10.48550/arXiv.1506.00019. URL: http://arxiv.org/abs/1506.00019.

[34] David E. Rumelhart et al. “Learning representations by back-propagating errors”. en. In: *Nature* 323.6088 (Oct. 1986). Publisher: Nature Publishing Group, pp. 533–536. DOI: 10.1038/323533a0. URL: https://www.nature.com/articles/323533a0.

[35] Yann LeCun et al. “Handwritten Digit Recognition with a Back-Propagation Network”. In: *Advances in Neural Information Processing Systems*. Vol. 2. Morgan-Kaufmann, 1989. URL: https://proceedings.neurips.cc/paper/1989/hash/53c3bce66e43be4f209556518c2fcb54-Abstract.html.

[36] G. Cybenko. “Approximation by superpositions of a sigmoidal function”. en. In: *Mathematics of Control, Signals and Systems* 2.4 (Dec. 1989), pp. 303–314. DOI: 10.1007/BF02551274. URL: https://doi.org/10.1007/BF02551274.

[37] Sepp Hochreiter. “The vanishing gradient problem during learning recurrent neural nets and problem solutions”. In: *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.* 6.2 (Apr. 1998), pp. 107–116. DOI: 10.1142/S0218488598000094. URL: https://doi.org/10.1142/S0218488598000094.

[38] Weijiang Feng et al. *Audio visual speech recognition with multimodal recurrent neural networks*. Pages: 688. May 2017. DOI: 10.1109/IJCNN.2017.7965918.

[39] Vincenzo Lomonaco. “Deep learning for computer vision: a comparison between convolutional neural networks and hierarchical temporal memories on object recognition tasks”. PhD thesis. Sept. 2015.

[40] Sepp Hochreiter et al. “Long Short-Term Memory”. In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[41] Rajat Raina et al. “Large-scale deep unsupervised learning using graphics processors”. In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML ’09. New York, NY, USA: Association for Computing Machinery, June 2009, pp. 873–880. DOI: 10.1145/1553374.1553486. URL: https://doi.org/10.1145/1553374.1553486.

[42] Jia Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. ISSN: 1063-6919. June 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848. URL: https://ieeexplore.ieee.org/document/5206848.

[43] Bodo Rueckauer et al. “Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification”. English. In: *Frontiers in Neuroscience* 11 (Dec. 2017). Publisher: Frontiers. DOI: 10.3389/fnins.2017.00682. URL: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2017.00682/full.

[44] Abhronil Sengupta et al. “Going Deeper in Spiking Neural Networks: VGG and Residual Architectures”. English. In: *Frontiers in Neuroscience* 13 (Mar. 2019). Publisher: Frontiers. DOI: 10.3389/fnins.2019.00095. URL: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2019.00095/full.

[45] Friedemann Zenke et al. “The Remarkable Robustness of Surrogate Gradient Learning for Instilling Complex Function in Spiking Neural Networks”. In: *Neural Computation* 33.4 (Mar. 2021), pp. 899–925. DOI: 10.1162/neco_a_01367. URL: https://doi.org/10.1162/neco_a_01367.

[46] Sumit Bam Shrestha et al. *SLAYER: Spike Layer Error Reassignment in Time*. arXiv:1810.08646 [cs]. Sept. 2018. DOI: 10.48550/arXiv.1810.08646. URL: http://arxiv.org/abs/1810.08646.

[47] Jason K. Eshraghian et al. *Training Spiking Neural Networks Using Lessons From Deep Learning*. arXiv:2109.12894 [cs]. Aug. 2023. DOI: 10.48550/arXiv.2109.12894. URL: http://arxiv.org/abs/2109.12894.

[48] Wei Fang et al. *SpikingJelly: An open-source machine learning infrastructure platform for spike-based intelligence*. en. arXiv:2310.16620 [cs]. Oct. 2023. DOI: 10.48550/arXiv.2310.16620. URL: http://arxiv.org/abs/2310.16620.

[49] Sadique Sheik et al. *SINABS: A simple Pytorch based SNN library specialised for Speck*. Sept. 2023. DOI: 10.5281/zenodo.8385545. URL: https://zenodo.org/records/8385545.

[50] Denis Alevi et al. "Brian2CUDA: Flexible and Efficient Simulation of Spiking Neural Network Models on GPUs". English. In: *Frontiers in Neuroinformatics* 16 (Oct. 2022). Publisher: Frontiers. DOI: 10.3389/fninf.2022.883700. URL: https://www.frontiersin.org/journals/neuroinformatics/articles/10.3389/fninf.2022.883700/full.

[51] Daniel Rasmussen. *NengoDL: Combining deep learning and neuromorphic modelling methods*. arXiv:1805.11144 [cs]. Mar. 2019. DOI: 10.48550/arXiv.1805.11144. URL: http://arxiv.org/abs/1805.11144.

[52] James C. Knight et al. "PyGeNN: A Python Library for GPU-Enhanced Neural Networks". English. In: *Frontiers in Neuroinformatics* 15 (Apr. 2021). Publisher: Frontiers. DOI: 10.3389/fninf.2021.659005. URL: https://www.frontiersin.org/journals/neuroinformatics/articles/10.3389/fninf.2021.659005/full.

[53] Maarten H. P. Kole et al. "Signal Processing in the Axon Initial Segment". In: *Neuron* 73.2 (Jan. 2012), pp. 235–247. DOI: 10.1016/j.neuron.2012.01.007. URL: https://www.sciencedirect.com/science/article/pii/S0896627312000505.

[54] A. L. Hodgkin et al. "A quantitative description of membrane current and its application to conduction and excitation in nerve". en. In: *The Journal of Physiology* 117.4 (1952). _eprint: https://physoc.onlinelibrary.wiley.com/doi/pdf/10.1113/jphysiol.1952.sp004764, pp. 500–544. DOI: 10.1113/jphysiol.1952.sp004764. URL: https://onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1952.sp004764.

[55] Wulfram Gerstner et al. *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. en. 1st ed. Cambridge University Press, July 2014. DOI: 10.1017/CBO9781107447615. URL: https://www.cambridge.org/core/product/identifier/9781107447615/type/book.

[56] Kashu Yamazaki et al. "Spiking Neural Networks and Their Applications: A Review". en. In: *Brain Sciences* 12.7 (July 2022). Publisher: Multidisciplinary Digital Publishing Institute, p. 863. DOI: 10.3390/brainsci12070863. URL: https://www.mdpi.com/2076-3425/12/7/863.

[57] Chethan M. Parameshwara et al. *SpikeMS: Deep Spiking Neural Network for Motion Segmentation*. Pages: 3420. Sept. 2021. DOI: 10.1109/IROS51168.2021.9636506.

[58] W Rall. "Distinguishing theoretical synaptic potentials computed for different soma-dendritic distributions of synaptic input." In: *Journal of Neurophysiology* 30.5 (Sept. 1967). Publisher: American Physiological Society, pp. 1138–1168. DOI: 10.1152/jn.1967.30.5.1138. URL: https://journals.physiology.org/doi/abs/10.1152/jn.1967.30.5.1138.

[59] Guo-qiang Bi et al. "Synaptic Modifications in Cultured Hippocampal Neurons: Dependence on Spike Timing, Synaptic Strength, and Postsynaptic Cell Type". en. In: *Journal of Neuroscience* 18.24 (Dec. 1998). Publisher: Society for Neuroscience Section: ARTICLE, pp. 10464–10472. DOI: 10.1523/JNEUROSCI.18-24-10464.1998. URL: https://www.jneurosci.org/content/18/24/10464.

[60] Peter Dayan et al. *Theoretical neuroscience: computational and mathematical modeling of neural systems*. en. Computational neuroscience. Cambridge, Mass.: MIT Press, 2001.

[61] Gaspardgpy. *English: Synaptic weight update using biological spike-timing dependent plasticity (STDP)*. Jan. 2023. URL: https://commons.wikimedia.org/wiki/File:Stdp_biological.svg.

[62] Richard FitzHugh. "Impulses and Physiological States in Theoretical Models of Nerve Membrane". In: *Biophysical Journal* 1.6 (July 1961), pp. 445–466. DOI: 10.1016/s0006-3495(61)86902-6. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1366333/.

[63] J. Nagumo et al. "An Active Pulse Transmission Line Simulating Nerve Axon". In: *Proceedings of the IRE* 50.10 (Oct. 1962), pp. 2061–2070. DOI: 10.1109/JRPROC.1962.288235. URL: https://ieeexplore.ieee.org/document/4066548.

[64] C. Morris et al. “Voltage oscillations in the barnacle giant muscle fiber”. en. In: *Biophysical Journal* 35.1 (July 1981), pp. 193–213. DOI: 10.1016/S0006-3495(81)84782-0. URL: https://linkinghub.elsevier.com/retrieve/pii/S0006349581847820.

[65] A. N. Burkitt. “A Review of the Integrate-and-fire Neuron Model: I. Homogeneous Synaptic Input”. en. In: *Biological Cybernetics* 95.1 (July 2006), pp. 1–19. DOI: 10.1007/s00422-006-0068-6. URL: https://doi.org/10.1007/s00422-006-0068-6.

[66] Nicolas Brunel et al. “Lapicque’s 1907 paper: from frogs to integrate-and-fire”. eng. In: *Biological Cybernetics* 97.5-6 (Dec. 2007), pp. 337–339. DOI: 10.1007/s00422-007-0190-0.

[67] L. F. Abbott. “Lapicque’s introduction of the integrate-and-fire model neuron (1907)”. eng. In: *Brain Research Bulletin* 50.5-6 (1999), pp. 303–304. DOI: 10.1016/s0361-9230(99)00161-6.

[68] Bard Ermentrout. “Type I Membranes, Phase Resetting Curves, and Synchrony”. en. In: *Neural Computation* 8.5 (July 1996), pp. 979–1001. DOI: 10.1162/neco.1996.8.5.979. URL: https://direct.mit.edu/neco/article/8/5/979-1001/5998.

[69] E.M. Izhikevich. “Simple model of spiking neurons”. en. In: *IEEE Transactions on Neural Networks* 14.6 (Nov. 2003), pp. 1569–1572. DOI: 10.1109/TNN.2003.820440. URL: http://ieeexplore.ieee.org/document/1257420/.

[70] Eugene M. Izhikevich. *Dynamical systems in neuroscience: the geometry of excitability and bursting*. en. Computational neuroscience. Cambridge, Mass: MIT Press, 2007.

[71] Romain Brette et al. “Adaptive exponential integrate-and-fire model as an effective description of neuronal activity”. eng. In: *Journal of Neurophysiology* 94.5 (Nov. 2005), pp. 3637–3642. DOI: 10.1152/jn.00686.2005.

[72] Richard Naud et al. “Firing patterns in the adaptive exponential integrate-and-fire model”. In: *Biological Cybernetics* 99.4 (2008), pp. 335–347. DOI: 10.1007/s00422-008-0264-7. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2798047/.

[73] Wulfram Gerstner. “Coding properties of spiking neurons: reverse and cross-correlations”. en. In: *Neural Networks* 14.6-7 (July 2001), pp. 599–610. DOI: 10.1016/S0893-6080(01)00053-3. URL: https://linkinghub.elsevier.com/retrieve/pii/S0893608001000533.

[74] Tong Bu et al. *Optimal ANN-SNN Conversion for High-accuracy and Ultra-low-latency Spiking Neural Networks*. arXiv:2303.04347 [cs]. Mar. 2023. DOI: 10.48550/arXiv.2303.04347. URL: http://arxiv.org/abs/2303.04347.

[75] Peter U. Diehl et al. “Unsupervised learning of digit recognition using spike-timing-dependent plasticity”. English. In: *Frontiers in Computational Neuroscience* 9 (Aug. 2015). Publisher: Frontiers. DOI: 10.3389/fncom.2015.00099. URL: https://www.frontiersin.org/journals/computational-neuroscience/articles/10.3389/fncom.2015.00099/full.

[76] Saeed Reza Kheradpisheh et al. “S4NN: temporal backpropagation for spiking neural networks with one spike per neuron”. In: *International Journal of Neural Systems* 30.06 (June 2020). arXiv:1910.09495 [cs], p. 2050027. DOI: 10.1142/S0129065720500276. URL: http://arxiv.org/abs/1910.09495.

[77] Lina Bonilla et al. “Analyzing time-to-first-spike coding schemes: A theoretical approach”. English. In: *Frontiers in Neuroscience* 16 (Sept. 2022). Publisher: Frontiers. DOI: 10.3389/fnins.2022.971937. URL: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2022.971937/full.

[78] Siying Liu et al. “First-spike coding promotes accurate and efficient spiking neural networks for discrete events with rich temporal structures”. English. In: *Frontiers in Neuroscience* 17 (Oct. 2023). Publisher: Frontiers. DOI: 10.3389/fnins.2023.1266003. URL: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2023.1266003/full.

[79] Simon Thorpe et al. “Rank Order Coding”. en. In: *Computational Neuroscience: Trends in Research, 1998*. Ed. by James M. Bower. Boston, MA: Springer US, 1998, pp. 113–118. DOI: 10.1007/978-1-4615-4831-7_19. URL: https://doi.org/10.1007/978-1-4615-4831-7_19.

[80] J. Gautrais et al. “Rate coding versus temporal order coding: a theoretical approach”. eng. In: *Bio Systems* 48.1-3 (1998), pp. 57–65. DOI: 10.1016/s0303-2647(98)00050-1.

[81] Sander M. Bohte. “The evidence for neural information processing with precise spike-times: A survey”. en. In: *Natural Computing* 3.2 (June 2004), pp. 195–206. DOI: 10.1023/B:NACO.0000027755.02868.60. URL: https://link.springer.com/10.1023/B:NACO.0000027755.02868.60.

[82] Saeed Reza Kheradpisheh et al. “STDP-based spiking deep convolutional neural networks for object recognition”. In: *Neural Networks* 99 (Mar. 2018), pp. 56–67. DOI: 10.1016/j.neunet.2017.12.005. URL: https://www.sciencedirect.com/science/article/pii/S0893608017302903.

[83] Sander M. Bohte et al. “Error-backpropagation in temporally encoded networks of spiking neurons”. en. In: *Neurocomputing* 48.1-4 (Oct. 2002), pp. 17–37. DOI: 10.1016/S0925-2312(01)00658-0. URL: https://linkinghub.elsevier.com/retrieve/pii/S0925231201006580.

[84] Eugene M. Izhikevich. “NEURAL EXCITABILITY, SPIKING AND BURSTING”. en. In: *International Journal of Bifurcation and Chaos* 10.06 (June 2000), pp. 1171–1266. DOI: 10.1142/S0218127400000840. URL: https://www.worldscientific.com/doi/abs/10.1142/S0218127400000840.

[85] Hugo G. Eyherabide et al. “Bursts generate a non-reducible spike-pattern code”. English. In: *Frontiers in Neuroscience* 3 (May 2009). Publisher: Frontiers. DOI: 10.3389/neuro.01.002.2009. URL: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/neuro.01.002.2009/full.

[86] Christoph Kayser et al. “Spike-phase coding boosts and stabilizes information carried by spatial and temporal spike patterns”. eng. In: *Neuron* 61.4 (Feb. 2009), pp. 597–608. DOI: 10.1016/j.neuron.2009.01.008.

[87] Timothée Masquelier et al. “Oscillations, phase-of-firing coding, and spike timing-dependent plasticity: an efficient learning scheme”. eng. In: *The Journal of Neuroscience: The Official Journal of the Society for Neuroscience* 29.43 (Oct. 2009), pp. 13484–13493. DOI: 10.1523/JNEUROSCI.2207-09.2009.

[88] Wenzhe Guo et al. “Neural Coding in Spiking Neural Networks: A Comparative Study for Robust Neuromorphic Systems”. English. In: *Frontiers in Neuroscience* 15 (Mar. 2021). Publisher: Frontiers. DOI: 10.3389/fnins.2021.638474. URL: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2021.638474/full.

[89] R. S. Zemel et al. “Probabilistic interpretation of population codes”. eng. In: *Neural Computation* 10.2 (Feb. 1998), pp. 403–430. DOI: 10.1162/089976698300017818.

[90] Nitin Rathi et al. *DIET-SNN: Direct Input Encoding With Leakage and Threshold Optimization in Deep Spiking Neural Networks*. arXiv:2008.03658 [cs]. Dec. 2020. DOI: 10.48550/arXiv.2008.03658. URL: http://arxiv.org/abs/2008.03658.

[91] Yuhang Li et al. “A Free Lunch From ANN: Towards Efficient, Accurate Spiking Neural Networks Calibration”. en. In: *Proceedings of the 38th International Conference on Machine Learning*. ISSN: 2640-3498. PMLR, July 2021, pp. 6316–6325. URL: https://proceedings.mlr.press/v139/li21d.html.

[92] Qingyan Meng et al. “Training much deeper spiking neural networks with a small number of time-steps”. In: *Neural Networks* 153 (Sept. 2022), pp. 254–268. DOI: 10.1016/j.neunet.2022.06.001. URL: https://www.sciencedirect.com/science/article/pii/S0893608022002064.

[93] Haiyan Jiang et al. “A Unified Optimization Framework of ANN-SNN Conversion: Towards Optimal Mapping from Activation Values to Firing Rates”. en. In: *Proceedings of the 40th International Conference on Machine Learning*. ISSN: 2640-3498. PMLR, July 2023, pp. 14945–14974. URL: https://proceedings.mlr.press/v202/jiang23a.html.

[94] Guillermo Gallego et al. “Event-based Vision: A Survey”. en. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.1 (Jan. 2022). arXiv:1904.08405 [cs], pp. 154–180. DOI: 10.1109/TPAMI.2020.3008413. URL: http://arxiv.org/abs/1904.08405.

[95] Peter O’Connor et al. *Sigma Delta Quantized Networks*. arXiv:1611.02024 [cs]. Nov. 2016. DOI: 10.48550/arXiv.1611.02024. URL: http://arxiv.org/abs/1611.02024.

[96] Guillaume Bellec et al. “A solution to the learning dilemma for recurrent networks of spiking neurons”. en. In: *Nature Communications* 11.1 (July 2020). Publisher: Nature Publishing Group, p. 3625. DOI: 10.1038/s41467-020-17236-y. URL: https://www.nature.com/articles/s41467-020-17236-y.

[97] Amirhossein Tavanaei et al. “Deep Learning in Spiking Neural Networks”. In: *Neural Networks* 111 (Mar. 2019). arXiv:1804.08150 [cs], pp. 47–63. DOI: 10.1016/j.neunet.2018.12.002. URL: http://arxiv.org/abs/1804.08150.

[98] Ian Goodfellow et al. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[99] Friedemann Zenke et al. “SuperSpike: Supervised learning in multi-layer spiking neural networks”. In: *Neural Computation* 30.6 (June 2018). arXiv:1705.11146 [q-bio], pp. 1514–1541. DOI: 10.1162/neco_a_01086. URL: http://arxiv.org/abs/1705.11146.

[100] Yujie Wu et al. “Spatio-Temporal Backpropagation for Training High-Performance Spiking Neural Networks”. English. In: *Frontiers in Neuroscience* 12 (May 2018). Publisher: Frontiers. DOI: 10.3389/fnins.2018.00331. URL: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2018.00331/full.

[101] Xinqiao Jiang et al. “Energy-Efficient and High-Performance Ship Classification Strategy Based on Siamese Spiking Neural Network in Dual-Polarized SAR Images”. In: *Remote Sensing* 15 (Oct. 2023), p. 4966. DOI: 10.3390/rs15204966.

[102] Chen-Yu Lee et al. *Deeply-Supervised Nets*. arXiv:1409.5185 [stat]. Sept. 2014. DOI: 10.48550/arXiv.1409.5185. URL: http://arxiv.org/abs/1409.5185.

[103] Timothy P. Lillicrap et al. *Random feedback weights support learning in deep neural networks*. arXiv:1411.0247 [q-bio]. Nov. 2014. DOI: 10.48550/arXiv.1411.0247. URL: http://arxiv.org/abs/1411.0247.

[104] Arild Nøkland. *Direct Feedback Alignment Provides Learning in Deep Neural Networks*. arXiv:1609.01596 [stat]. Dec. 2016. DOI: 10.48550/arXiv.1609.01596. URL: http://arxiv.org/abs/1609.01596.

[105] Robert Gütig et al. “The tempotron: a neuron that learns spike timing–based decisions”. en. In: *Nature Neuroscience* 9.3 (Mar. 2006). Publisher: Nature Publishing Group, pp. 420–428. DOI: 10.1038/nn1643. URL: https://www.nature.com/articles/nn1643.

[106] Răzvan Florian. “The chronotron: a neuron that learns to fire temporally-precise spike patterns”. en. In: *Nature Precedings* (Nov. 2010). Publisher: Nature Publishing Group, pp. 1–1. DOI: 10.1038/npre.2010.5190.1. URL: https://www.nature.com/articles/npre.2010.5190.1.

[107] Jonathan D. Victor et al. *Metric-space analysis of spike trains: theory, algorithms, and application*. arXiv:q-bio/0309031. Oct. 1998. DOI: 10.48550/arXiv.q-bio/0309031. URL: http://arxiv.org/abs/q-bio/0309031.

[108] M. C. van Rossum. “A novel spike distance”. eng. In: *Neural Computation* 13.4 (Apr. 2001), pp. 751–763. DOI: 10.1162/089976601300014321.

[109] D. O. Hebb. *The organization of behavior; a neuropsychological theory*. The organization of behavior; a neuropsychological theory. Pages: xix, 335. Oxford, England: Wiley, 1949.

[110] Natalia Caporale et al. “Spike timing-dependent plasticity: a Hebbian learning rule”. eng. In: *Annual Review of Neuroscience* 31 (2008), pp. 25–46. DOI: 10.1146/annurev.neuro.31.060407.125639.

[111] Timothée Masquelier. “Relative spike time coding and STDP-based orientation selectivity in the early visual system in natural continuous and saccadic vision: a computational model”. eng. In: *Journal of Computational Neuroscience* 32.3 (June 2012), pp. 425–441. DOI: 10.1007/s10827-011-0361-9.

[112] Friedemann Zenke et al. “Synaptic Plasticity in Neural Networks Needs Homeostasis with a Fast Rate Detector”. en. In: *PLOS Computational Biology* 9.11 (Nov. 2013). Publisher: Public Library of Science, e1003330. DOI: 10.1371/journal.pcbi.1003330. URL: https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1003330.

[113] Jean-Pascal Pfister et al. “Triplets of spikes in a model of spike timing-dependent plasticity”. eng. In: *The Journal of Neuroscience: The Official Journal of the Society for Neuroscience* 26.38 (Sept. 2006), pp. 9673–9682. DOI: 10.1523/JNEUROSCI.1425-06.2006.

[114] Richard S. Sutton et al. *Reinforcement learning: An introduction, 2nd ed*. Reinforcement learning: An introduction, 2nd ed. Pages: xxii, 526. Cambridge, MA, US: The MIT Press, 2018.

[115] Richard S Sutton et al. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Advances in Neural Information Processing Systems*. Vol. 12. MIT Press, 1999. URL: https://papers.nips.cc/paper_files/paper/1999/hash/464d828b85b0bed98e80ade0a5c43b0f-Abstract.html.

[116] Chen Tang et al. *Deep Reinforcement Learning for Robotics: A Survey of Real-World Successes*. arXiv:2408.03539 [cs]. Sept. 2024. DOI: 10.48550/arXiv.2408.03539. URL: http://arxiv.org/abs/2408.03539.

[117] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533.

[118] Dmitry Kalashnikov et al. *QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation*. arXiv:1806.10293 [cs]. Nov. 2018. DOI: 10.48550/arXiv.1806.10293. URL: http://arxiv.org/abs/1806.10293.

[119] John Schulman et al. *Trust Region Policy Optimization*. en. arXiv:1502.05477 [cs]. Apr. 2017. DOI: 10.48550/arXiv.1502.05477. URL: http://arxiv.org/abs/1502.05477.

[120] Elia Kaufmann et al. "Champion-level drone racing using deep reinforcement learning". In: *Nature* 620.7976 (2023), pp. 982–987. DOI: 10.1038/s41586-023-06419-4. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10468397/.

[121] Robin Ferede et al. *One Net to Rule Them All: Domain Randomization in Quadcopter Racing Across Different Platforms*. arXiv:2504.21586 [cs]. Apr. 2025. DOI: 10.48550/arXiv.2504.21586. URL: http://arxiv.org/abs/2504.21586.

[122] William Koch et al. *Neuroflight: Next Generation Flight Control Firmware*. arXiv:1901.06553 [cs]. Sept. 2019. DOI: 10.48550/arXiv.1901.06553. URL: http://arxiv.org/abs/1901.06553.

[123] William Koch et al. "Reinforcement Learning for UAV Attitude Control". en. In: *ACM Transactions on Cyber-Physical Systems* 3.2 (Apr. 2019), pp. 1–21. DOI: 10.1145/3301273. URL: https://dl.acm.org/doi/10.1145/3301273.

[124] Nikita Rudin et al. *Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning*. arXiv:2109.11978 [cs]. Aug. 2022. DOI: 10.48550/arXiv.2109.11978. URL: http://arxiv.org/abs/2109.11978.

[125] Jemin Hwangbo et al. "Learning agile and dynamic motor skills for legged robots". In: *Science Robotics* 4.26 (Jan. 2019). Publisher: American Association for the Advancement of Science, eaau5872. DOI: 10.1126/scirobotics.aau5872. URL: https://www.science.org/doi/10.1126/scirobotics.aau5872.

[126] OpenAI et al. *Solving Rubik's Cube with a Robot Hand*. arXiv:1910.07113 [cs]. Oct. 2019. DOI: 10.48550/arXiv.1910.07113. URL: http://arxiv.org/abs/1910.07113.

[127] Timothy P. Lillicrap et al. *Continuous control with deep reinforcement learning*. arXiv:1509.02971 [cs]. July 2019. DOI: 10.48550/arXiv.1509.02971. URL: http://arxiv.org/abs/1509.02971.

[128] David Silver et al. "Deterministic policy gradient algorithms". In: *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*. ICML'14. Beijing, China: JMLR.org, June 2014, pp. I–387–I–395.

[129] Scott Fujimoto et al. *Addressing Function Approximation Error in Actor-Critic Methods*. arXiv:1802.09477 [cs]. Oct. 2018. DOI: 10.48550/arXiv.1802.09477. URL: http://arxiv.org/abs/1802.09477.

[130] Tuomas Haarnoja et al. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. arXiv:1801.01290 [cs]. Aug. 2018. DOI: 10.48550/arXiv.1801.01290. URL: http://arxiv.org/abs/1801.01290.

[131] Tuomas Haarnoja et al. *Learning to Walk via Deep Reinforcement Learning*. arXiv:1812.11103 [cs]. June 2019. DOI: 10.48550/arXiv.1812.11103. URL: http://arxiv.org/abs/1812.11103.

[132] Marc Peter Deisenroth et al. “PILCO: a model-based and data-efficient approach to policy search”. In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. ICML’11. Madison, WI, USA: Omnipress, June 2011, pp. 465–472.

[133] Michael Janner et al. *When to Trust Your Model: Model-Based Policy Optimization*. arXiv:1906.08253 [cs]. Nov. 2021. DOI: 10.48550/arXiv.1906.08253. URL: http://arxiv.org/abs/1906.08253.

[134] Zexiang Yi et al. “Learning rules in spiking neural networks: A survey”. en. In: *Neurocomputing* (Jan. 2023). URL: https://openreview.net/forum?id=RkKPJaJAKn.

[135] Nicolas Frémaux et al. “Reinforcement Learning Using a Continuous Time Actor-Critic Framework with Spiking Neurons”. en. In: *PLOS Computational Biology* 9.4 (Apr. 2013). Publisher: Public Library of Science, e1003024. DOI: 10.1371/journal.pcbi.1003024. URL: https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1003024.

[136] Nicolas Frémaux et al. “Neuromodulated Spike-Timing-Dependent Plasticity, and Theory of Three-Factor Learning Rules”. eng. In: *Frontiers in Neural Circuits* 9 (2015), p. 85. DOI: 10.3389/fncir.2015.00085.

[137] Guangzhi Tang et al. *Deep Reinforcement Learning with Population-Coded Spiking Neural Network for Continuous Control*. arXiv:2010.09635 [cs]. Oct. 2020. DOI: 10.48550/arXiv.2010.09635. URL: http://arxiv.org/abs/2010.09635.

[138] Guangzhi Tang et al. *Reinforcement co-Learning of Deep and Spiking Neural Networks for Energy-Efficient Mapless Navigation with Neuromorphic Hardware*. arXiv:2003.01157 [cs]. July 2020. DOI: 10.48550/arXiv.2003.01157. URL: http://arxiv.org/abs/2003.01157.

[139] Mahmoud Akl et al. “Porting Deep Spiking Q-Networks to neuromorphic chip Loihi”. In: *International Conference on Neuromorphic Systems 2021*. ICONS 2021. New York, NY, USA: Association for Computing Machinery, Oct. 2021, pp. 1–7. DOI: 10.1145/3477145.3477159. URL: https://doi.org/10.1145/3477145.3477159.

[140] Yang Wang et al. “Event-Enhanced Multi-Modal Spiking Neural Network for Dynamic Obstacle Avoidance”. In: *Proceedings of the 31st ACM International Conference on Multimedia*. arXiv:2310.02361 [cs]. Oct. 2023, pp. 3138–3148. DOI: 10.1145/3581783.3612147. URL: http://arxiv.org/abs/2310.02361.

[141] Genghang Zhuang et al. “An Energy-Efficient Lane-Keeping System Using 3D LiDAR Based on Spiking Neural Network”. In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Oct. 2023). Conference Name: 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) ISBN: 9781665491907 Place: Detroit, MI, USA Publisher: IEEE, pp. 4763–4769. DOI: 10.1109/IROS55552.2023.10342044. URL: https://ieeexplore.ieee.org/document/10342044/.

[142] Mahmoud Akl et al. “Toward robust and scalable deep spiking reinforcement learning”. English. In: *Frontiers in Neurorobotics* 16 (Jan. 2023). Publisher: Frontiers. DOI: 10.3389/fnbot.2022.1075647. URL: https://www.frontiersin.org/journals/neurorobotics/articles/10.3389/fnbot.2022.1075647/full.

[143] Antonio Loquercio et al. “Learning High-Speed Flight in the Wild”. In: *Science Robotics* 6.59 (Oct. 2021). arXiv:2110.05113 [cs], eabg5810. DOI: 10.1126/scirobotics.abg5810. URL: http://arxiv.org/abs/2110.05113.

[144] Elia Kaufmann et al. *Deep Drone Acrobatics*. arXiv:2006.05768 [cs]. June 2020. DOI: 10.48550/arXiv.2006.05768. URL: http://arxiv.org/abs/2006.05768.

[145] Philipp Foehn et al. “Agilicious: Open-Source and Open-Hardware Agile Quadrotor for Vision-Based Flight”. In: *Science Robotics* 7.67 (June 2022). arXiv:2307.06100 [cs], eabl6259. DOI: 10.1126/scirobotics.abl6259. URL: http://arxiv.org/abs/2307.06100.

[146] Yunlong Song et al. *Learning Perception-Aware Agile Flight in Cluttered Environments*. arXiv:2210.01841 [cs]. Mar. 2023. DOI: 10.48550/arXiv.2210.01841. URL: http://arxiv.org/abs/2210.01841.

[147] Davide Falanga et al. “Dynamic obstacle avoidance for quadrotors with event cameras”. eng. In: *Science Robotics* 5.40 (Mar. 2020), eaaz9712. DOI: 10.1126/scirobotics.aaz9712.

[148] Nitin J. Sanket et al. *EVDodgeNet: Deep Dynamic Obstacle Dodging with Event Cameras*. arXiv:1906.02919 [cs]. Mar. 2020. DOI: 10.48550/arXiv.1906.02919. URL: http://arxiv.org/abs/1906.02919.

[149] Giovanni Cioffi et al. *Learned Inertial Odometry for Autonomous Drone Racing*. arXiv:2210.15287 [cs]. Feb. 2023. DOI: 10.48550/arXiv.2210.15287. URL: http://arxiv.org/abs/2210.15287.

[150] *Ultra Low Power Deep-Learning-powered Autonomous Nano Drones*. URL: https://infoscience.epfl.ch/entities/publication/6268c8b5-29cf-4a42-b733-49ab8e448746.

[151] Lorenzo Lamberti et al. *Tiny-PULP-Dronets: Squeezing Neural Networks for Faster and Lighter Inference on Multi-Tasking Autonomous Nano-Drones*. arXiv:2407.02405 [cs]. July 2024. DOI: 10.48550/arXiv.2407.02405. URL: http://arxiv.org/abs/2407.02405.

[152] Antonio Loquercio et al. "DroNet: Learning to Fly by Driving". In: *IEEE Robotics and Automation Letters* 3.2 (Apr. 2018), pp. 1088–1095. DOI: 10.1109/LRA.2018.2795643. URL: https://ieeexplore.ieee.org/document/8264734.

[153] Dhiraj Gandhi et al. *Learning to Fly by Crashing*. arXiv:1704.05588 [cs]. Apr. 2017. DOI: 10.48550/arXiv.1704.05588. URL: http://arxiv.org/abs/1704.05588.

[154] Federico Paredes-Vallés et al. *Fully neuromorphic vision and control for autonomous drone flight*. arXiv:2303.08778 [cs]. Mar. 2023. DOI: 10.48550/arXiv.2303.08778. URL: http://arxiv.org/abs/2303.08778.

[155] Antonio Vitale et al. "Event-driven Vision and Control for UAVs on a Neuromorphic Chip". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. ISSN: 2577-087X. May 2021, pp. 103–109. DOI: 10.1109/ICRA48506.2021.9560881. URL: https://ieeexplore.ieee.org/document/9560881.

[156] Stein Stroobants et al. "Design and implementation of a parsimonious neuromorphic PID for onboard altitude control for MAVs using neuromorphic processors". en. In: *Proceedings of the International Conference on Neuromorphic Systems 2022*. Knoxville TN USA: ACM, July 2022, pp. 1–7. DOI: 10.1145/3546790.3546799. URL: https://dl.acm.org/doi/10.1145/3546790.3546799.

[157] Stein Stroobants et al. "Neuromorphic computing for attitude estimation onboard quadrotors". In: *Neuromorphic Computing and Engineering* 2.3 (Sept. 2022). arXiv:2304.08802 [cs], p. 034005. DOI: 10.1088/2634-4386/ac7ee0. URL: http://arxiv.org/abs/2304.08802.

[158] Stein Stroobants et al. *Neuromorphic Attitude Estimation and Control*. Issue: arXiv:2411.13945 arXiv:2411.13945 [cs]. Nov. 2024. DOI: 10.48550/arXiv.2411.13945. URL: http://arxiv.org/abs/2411.13945.

[159] Mathias Gehrig et al. *Event-Based Angular Velocity Regression with Spiking Networks*. arXiv:2003.02790 [cs]. Mar. 2020. DOI: 10.48550/arXiv.2003.02790. URL: http://arxiv.org/abs/2003.02790.

[160] Germain Haessig et al. *Spiking Optical Flow for Event-based Sensors Using IBM's TrueNorth Neurosynaptic System*. arXiv:1710.09820 [cs]. Oct. 2017. DOI: 10.48550/arXiv.1710.09820. URL: http://arxiv.org/abs/1710.09820.

[161] Matěj Karásek et al. "A tailless aerial robotic flapper reveals that flies use torque coupling in rapid banked turns". In: *Science* 361.6407 (Sept. 2018). Publisher: American Association for the Advancement of Science, pp. 1089–1094. DOI: 10.1126/science.aat0350. URL: https://www.science.org/doi/10.1126/science.aat0350.

[162] *Neuron Activation: Brain Insights for AI Evolution*. en-US. URL: https://viso.ai/deep-learning/neuron-activation/.

[163] Diederik P. Kingma et al. *Adam: A Method for Stochastic Optimization*. arXiv:1412.6980 [cs]. Jan. 2017. DOI: 10.48550/arXiv.1412.6980. URL: http://arxiv.org/abs/1412.6980.

[164] Rishabh Agarwal et al. *Deep Reinforcement Learning at the Edge of the Statistical Precipice*. arXiv:2108.13264 [cs]. Jan. 2022. DOI: 10.48550/arXiv.2108.13264. URL: http://arxiv.org/abs/2108.13264 (visited on 09/08/2025).