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

M.F. van Breukelen Castillo[*], R. Ferede, R.W. Vos, C. De Wagter G.C.H.E. de Croon [†]
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, machine learning has greatly advanced autonomous systems, from smartphones and self-driving vehicles to large language models. This progress comes at a cost: deep networks are energy-intensive, demanding substantial resources for training and inference. As their use grows, there is a pressing need for energy-efficient models that can deliver real-time performance on embedded systems. This challenge is acute in micro aerial vehicles (MAVs), which require fast, low-latency control under strict power and hardware limits. Tasks such as drone racing or agile navigation push the limits of onboard computation, as controllers must run at high frequencies on lightweight, battery-powered platforms. Although artificial neural networks (ANNs) have shown strong control capabilities [1, 2], their poor energy efficiency remains a critical 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 generated only when a neuron's membrane potential crosses a threshold. As a result, large parts of the network remain inactive, and computation scales with the number of spikes rather than the number of neurons. This enables low-power computation with minimal latency and can yield orders-of-magnitude energy savings compared to ANNs that update all neurons continuously [3]. These properties make SNNs attractive for real-time control on embedded platforms when deployed on neuromorphic hardware such as Intel's Loihi.

Despite this promise, SNNs remain underexplored in closed-loop robotic control. Prior work has examined their use in vision [4], neuromorphic sensing [5], and motor tasks [6], but not in high-speed quadcopter control. Reinforcement learning (RL) provides a natural framework, as agents learn control policies through trial-and-error interaction with an environment to maximise long-term reward [7]. Recent methods have enabled SNNs to be trained with surrogate gradients [8, 9], yet applications have mostly been confined to low-dimensional benchmarks, and robust real-time deployment is still lacking. Initial demonstrations show progress: Paredes-Vallés et al. implemented a spiking vision-to-control pipeline on Loihi, enabling autonomous drone flight at 200 Hz with microjoule-level inference cost [10], and Stroobants et al. achieved spiking attitude control on a Crazyflie at 500 Hz [11]. These works highlight the potential of neuromorphic approaches, but none address high-speed, continuous quadcopter control directly with fully spiking networks.

In this work we present the first implementation of a fully spiking actor-critic architecture for continuous quadcopter control in an agile gate navigation task. Learning is

[*]ORCID: (0009-0001-9337-7960)
[†]Email address(es): michaelvanbreukelen509@gmail.com , {R.Ferede; R.W.Vos, C.deWagter, G.C.H.E.deCroon}@tudelft.nl
Code=https://github.com/michael2992/msc_spiking_quadcopter_control

carried out with Proximal Policy Optimization (PPO), a reinforcement learning algorithm that updates the policy by sampling trajectories, evaluating performance, and making incremental changes while constraining updates to remain stable [12]. The network is built from Leaky Integrate-and-Fire (LIF) neurons, a standard spiking model in which inputs accumulate in a membrane potential that decays over time and trigger a spike when the threshold is crossed. To obtain continuous commands from discrete spikes, we apply spike-rate decoding, where spike rates are calculated over a short window and mapped to motor actions. The SNN architecture is matched in structure and parameter count to an ANN baseline [13], ensuring fair comparison.

Our results show that the SNN not only achieves strong performance in simulation but also outperforms the ANN in real-world flight tests, demonstrating higher robustness, greater average velocity, and more gates traversed within a 12-second interval. We also analyse how the number of spike-integration cycles affects control performance and latency. This work bridges the gap between neuromorphic computing and reinforcement learning, providing a foundation for 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:

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

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:

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

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

$$\dot{\omega}_i = \frac{\omega_{ci} - \omega_i}{\tau} \quad (4)$$

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:

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

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

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

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

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]:

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

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 [9]. 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 [9]. This is also exploited by rate decoding the output, as explained further in subsection 2.4 along side Figure 1 and how our implementation differs from a traditional ANN.

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. The network is composed of three fully connected layers of LIF neurons, parametrized by weight matrices W. 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 with the output being binary spikes $s \in \{0, 1\}^N$, with N denoting the number of neurons in the hidden layer. 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 $\bar{\mathbf{S}} \in [0, 1]^N$, for the N 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 [9]. Specifically, we use the derivative of a shifted arctangent function as the surrogate, adapted from [8]:

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

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]. 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 [18]. 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, as described in Figure 1. 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 which also ensures that subsequent benchmarking between the ANN and SNN is both fair and meaningful. The resulting architecture comprises three fully connected layers of LIF neurons, each containing 64 units as shown in Figure 2.

2.6 Training procedure and randomization

We train the SNN policy using the PPO algorithm [12], implemented via the Stable-Baselines3 reinforcement learning library [19]. 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 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 $x_0,\ y_0 \sim \mathcal{U}(-5,\ 5)$ m and $z_0 \sim \mathcal{U}(-3,\ 0)$ m. Linear velocities are initialized from $v_x,\ v_y,\ v_z \sim \mathcal{U}(-0.5,\ 0.5)$ 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 $p,\ q,\ r \sim \mathcal{U}(-0.1,\ 0.1)$ rad/s, and each motor is initialized with an angular velocity $w_i \sim \mathcal{U}(-1,\ 1)$ 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

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

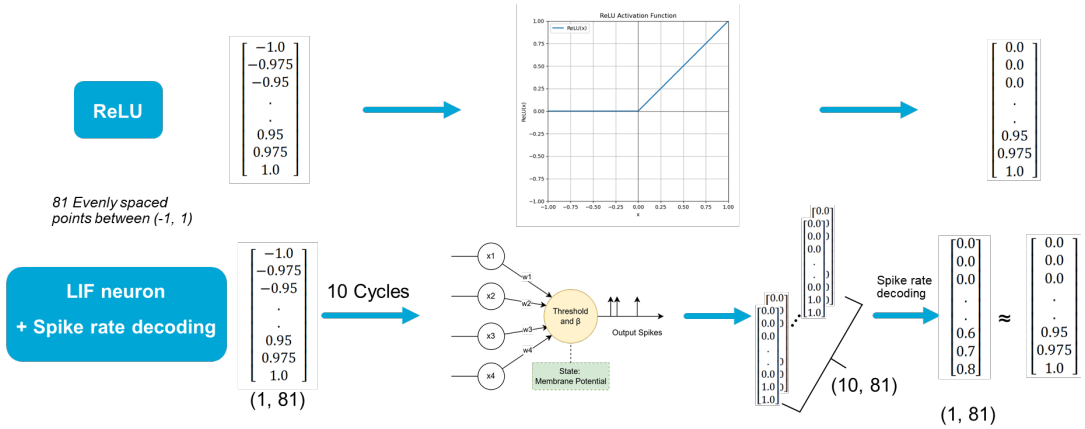

Figure 1: Comparative schematic showing how a LIF neuron with rate decoding over several cycles is similar to the ReLU activation function for an example input X of 81 evenly spaced points between [-1, 1].

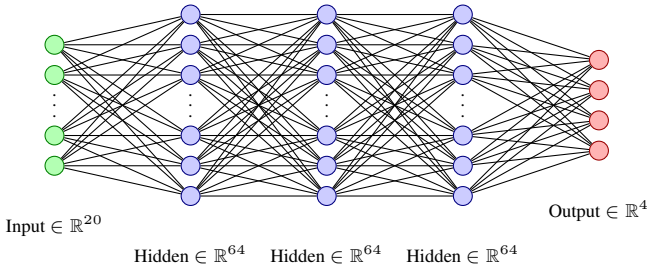

Figure 2: Schematic of the spiking neural **actor** network architecture. Each hidden layer contains 64 neurons. The value network is identical with the exception of the size of the output dimension which in contrast is 1

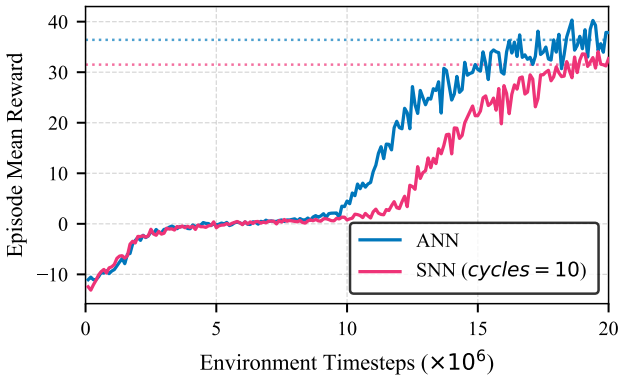

Figure 3: Training of an ANN compared to a SNN policy with 10 cycles over 20 million timesteps

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 m × 10 m × 7 m indoor space equipped for autonomous drone testing.

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 in this comparison uses a cycle count of 10, chosen on the basis of a preliminary analysis in which models were trained with cycle counts of up to 50. This analysis showed that while higher cycle counts could achieve similar or slightly improved rewards, they required substantially longer training times. A cycle count of 10 therefore provided the most favourable trade-off, achieving quite comparable reward while keeping training time manageable compared to the ANN baseline.

Each curve represents a single training run. While averaging across multiple seeds is common in RL to reduce variance [20], this was not feasible due to limited compute and long training times. All subsequent comparisons are therefore based on representative single runs.

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 lap-

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.

	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 of ANN and SNN policies over three trials.

top, with training time being proportional to the cycle count of the model.

To further compare flight performance, both models were trained with PPO until achieving a reward of $r \approx 50$ (ANN: 50.7, SNN: 50.82). Although they converged in a similar number of timesteps (ANN: 40.3×10^6, SNN: 41.1×10^6), training time differed greatly (ANN: 33 min, SNN: 10.4 h). Results over 1,000 simulated episodes are shown in Table 1.

Despite similar training rewards, the SNN achieved higher mean reward (64.93 vs. 58.23) and velocity on successful runs, with a lower crash rate (44.8% vs. 67.3%). Failures mainly resulted from extreme randomized initial conditions. The ANN tended to survive longer in failed runs but ultimately crashed more often, while the SNN either recovered or failed quickly.

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 starting from a defined hover position.

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).

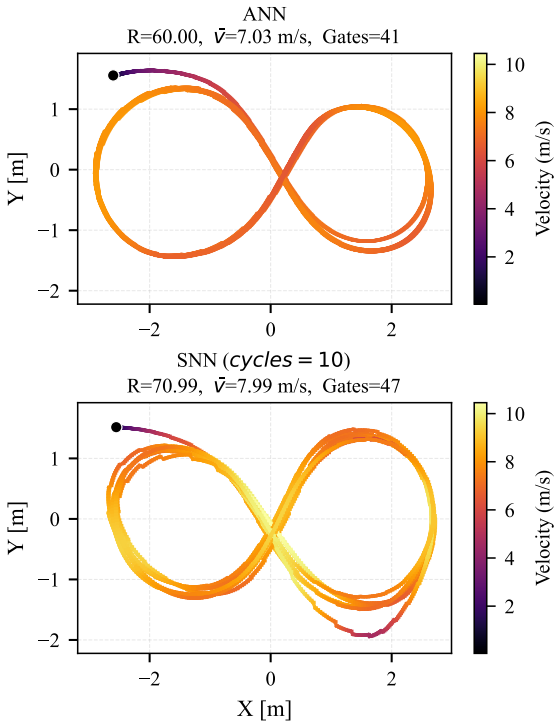

Figure 4: **Real flight** trajectory comparison between the best trials of the ANN and SNN with 10 cycles.

4.2 *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 5 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

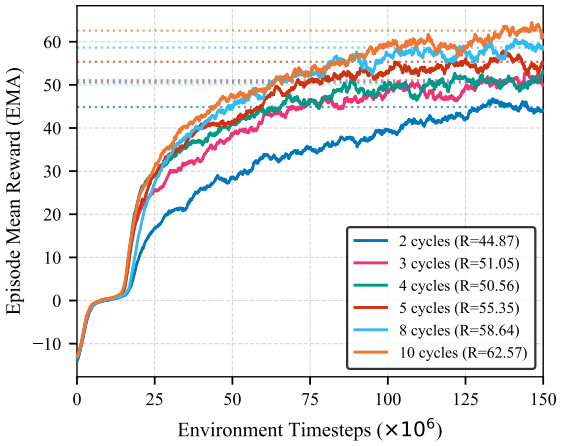

Figure 5: EMA of mean episode reward for six SNNs over 150 million timesteps.

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 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: lower observed update frequencies. 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,

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)

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 for **inference-constrained** tests from Table 3. Reward = Progress + Rate Penalty.

which passes 50.00 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.

Under similar observed frequency, results in Table 5 are consistent. The 8-cycle model gives the highest reward (70.01) with strong velocity and gates, while the 5-cycle model again leads in velocity (8.36 m/s) and gates (50.00) with competitive reward (69.81). Its observed frequency was higher than expected at 337.20 Hz, likely due to flight controller scheduling, which may have improved responsiveness.

Lower-cycle models at 333 Hz show reduced rewards and fewer gates, and the 10-cycle model benefits less from added cycles because responsiveness declines with higher computational load. Representative trajectories are shown in Figure 6.

The trajectories **(a)** and **(d)** in Figure 6 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-

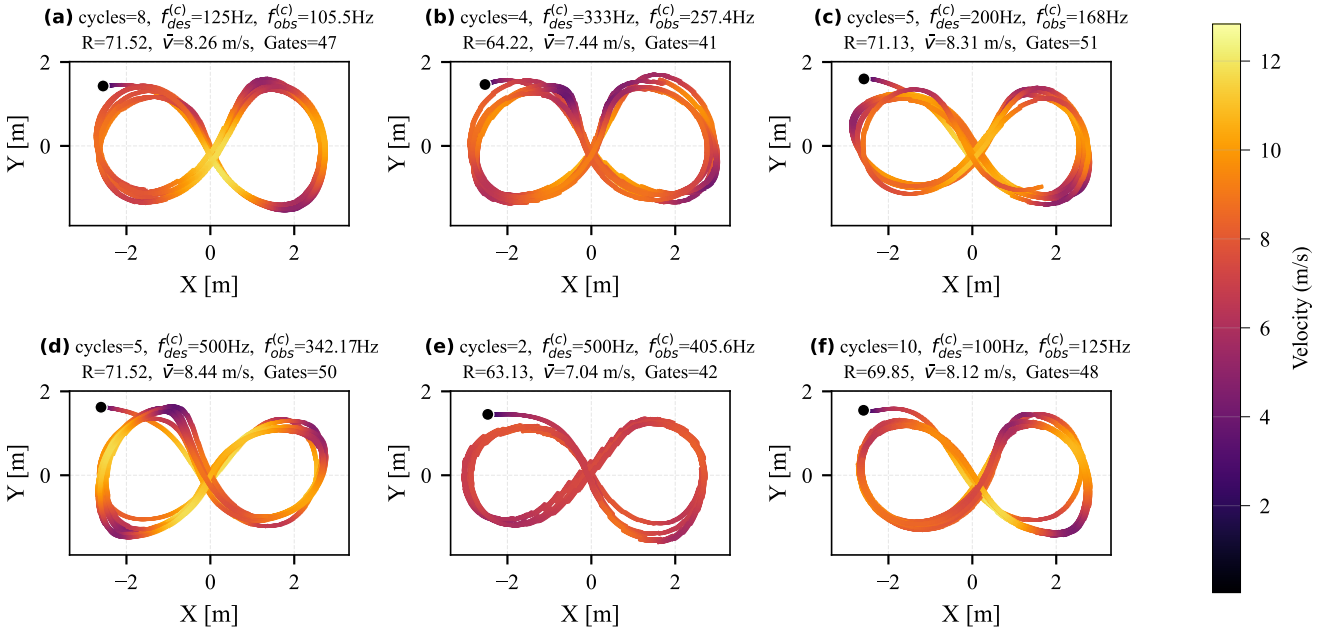

Figure 6: **Real flight** trajectories for selected trials. (a) and (d) highest reward, (b) and (e) second lowest and lowest reward, (c) highest gates passed, (f) highest cycle model. Videos: `https://drive.google.com/drive/folders/1uGINGe71wu0Hrh0_ZBDAHMIO-s6B5qd3?usp=drive_link`

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 tests across different cycle counts for three trials.

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 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.

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.

An extensive analysis of different cycle counts demonstrated a clear trade-off between temporal resolution and observed update frequency. 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] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction, 2nd ed*. Reinforcement learning: An introduction, 2nd ed. The MIT Press, Cambridge, MA, US, 2018. Pages: xxii, 526.

[8] 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].

[9] 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.

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

[19] 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.

[20] 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].