

Bio-Inspired Drone Control: A Reinforcement Learning-Trained Spiking Neural Networks for Agile Navigation in Dynamic Environment

Yin-Ching Lee

*Department of Computer Science
Boston University
Boston, USA
leewill@bu.edu*

Sebastiano Mengozzi

*DEI
University of Bologna
Bologna, Italy
sebastiano.mengozzi@unibo.it*

Luca Zanatta

*ITK
Norwegian University of Science and Technology
Trondheim, Norway
luca.zanatta@ntnu.no*

Andrea Bartolini

*DEI
University of Bologna
Bologna, Italy
a.bartolini@unibo.it*

Andrea Acquaviva

*DEI
University of Bologna
Bologna, Italy
andrea.acquaviva@unibo.it*

Francesco Barchi

*DEI
University of Bologna
Bologna, Italy
francesco.barchi@unibo.it*

Abstract—Controlling quadrotors autonomously in dynamic environments requires agile and robust flight policies to ensure rapid adaptation to environmental changes. Deep Reinforcement Learning (DRL) has been shown to be an effective method to train Artificial Neural Networks (ANNs) policies, outperforming optimal control algorithms in performance while being more resource-efficient. Spiking Neural Networks (SNNs), biologically inspired neural networks, present a promising approach by natively processing temporal data through discrete spikes. This property allows SNNs to incorporate the temporal dimension, even within a feed-forward architecture, unlike ANNs, which is crucial in dynamic environments. Moreover, SNNs can be efficiently executed on neuromorphic hardware accelerators, making them well-suited for deployment on resource-constrained computing platforms. In this work, we trained an agile flight SNN policy using the state-of-the-art Deep Reinforcement Learning (DRL) algorithm, Proximal Policy Optimization (PPO). The flight policy maps the system states to low-level control commands sent to the quadrotor. With simulation experiments, we demonstrate that, compared to ANN-based policies, SNN-based ones achieve a 2.5% improvement in success rate, a 40% increase in average flight speed, and a 28.6% reduction in the time required to reach the target. Our results suggest that neuromorphic computing approaches can be beneficial for dynamical state-based problems, providing valuable insights for designing lightweight and efficient controllers in time-sensitive applications.

Index Terms—Neurorobotics, Deep Reinforcement Learning, Spiking Neural Networks for Quadrotor Control.

I. INTRODUCTION

Unmanned Aerial Vehicles (UAVs), and especially quadrotors, have emerged as one of the most prominent robotic platforms due to their suitability for real-world applications

such as search and rescue, surveillance, and delivery. Their high agility [1] makes them particularly effective in dynamic and unconstrained environments. This capability allows autonomous systems to operate in unstructured settings where rapid real-time control is essential to adapt to environmental changes, minimize mission completion time, and avoid collisions.

However, achieving reliable agile flight in dynamic environments poses several challenges: (i) the autonomous quadrotor must be capable of perceiving its surroundings and detecting obstacles; (ii) real-time control is required to avoid collisions; and (iii) all algorithms must be executed on-board to ensure fast response times and avoid issues caused by communication delays or connection losses.

In this work, we explore the use of bio-inspired feed-forward neural networks - SNNs [2], [3] - to address the challenges of (ii) real-time control and (iii) on-board deployment. Due to their inherent event-based processing capabilities, SNNs offer the potential for highly efficient execution on custom neuromorphic hardware. In SNNs, information is encoded as discrete events, or spikes, which are temporally integrated by individual neurons. An output spike is emitted when a predefined threshold is crossed. This mechanism enables SNNs to model dynamical systems and effectively capture the temporal structure inherent in input event streams. When deployed on optimized accelerators, SNNs can significantly reduce memory footprint, energy consumption, and inference latency [4], [5], making them particularly well-suited for real-time applications on resource-constrained aerial platforms.

Previous approaches to control in dynamic environments have primarily relied on optimal control methods [6]–[8]. While effective in theory, these methods often lack robustness and require significant computational resources [9]–[11], making real-time execution on embedded systems impractical. Recent

The work has been partially funded by i) the Horizon Europe DECICE project (g.a. 101092582) and ii) the European Community and Italian National Fund project EdgeAI (g.a. 101097300).

979-8-3315-2037-3/25/\$31.00 ©2025 IEEE

work has demonstrated that ANNs can capture complex control strategies leveraging the DRL training paradigm [12]–[14]. DRL allows the formulation of high-level objectives through a reward function, enabling autonomous agents to learn control policies through trial-and-error interactions within simulated environments. Among DRL algorithms, PPO has emerged as a SOTA method due to its stability and sample efficiency [15]. DRL-trained controllers can match or exceed the performance of traditional optimal control methods while improving robustness to parameter uncertainty, thus narrowing the sim-to-real gap [9], [16], [17].

However, feed-forward ANNs are limited in their ability to process temporal information, as their outputs depend solely on inputs from the current state. While Recurrent neural networks (RNNs) can capture temporal dynamics, they introduce additional computational complexity and a larger number of trainable parameters, which pose challenges for real-time implementation [18], [19].

On the other hand, SNNs emerged as effective models to efficiently capture temporal dynamics. For this reason, we propose using SNNs as feed-forward neural controllers that inherently exploit temporal dynamics due to their inner structure. We train SNNs using the DRL framework SpikeGym [20]. The resulting feed-forward SNN controller outperforms its ANN-based counterpart, achieving 2.5% higher success rate in traversing a highly-dynamical environment, while reducing completion time by 28.6%. Moreover, the trained controller exhibits reduced computational complexity, making it suitable for deployment on embedded platforms such as nano-drones.

To evaluate our method, we replicate the high-speed quadrotor flight task from [17], which involves navigating through a sequence of swinging gates. This task is well-suited for testing a controller’s ability to operate in rapidly changing environments with limited information in a state-based setting. It introduces partial observability, non-linear dynamics, and strict real-time constraints, making it a compelling benchmark for assessing both control performance and adaptability.

The remainder of this paper is structured as follows: Section II presents the theoretical background. Section III reviews related work. Section IV details the methods used in our implementation. Section V presents the experimental results. Finally, Section VI concludes the paper by examining its limitations and outlining directions for future work.

II. PRELIMINARIES

An SNN comprises spiking neurons that process and transmit information through discrete spikes. Among various neuron models, the Leaky Integrate-and-Fire (LIF) model is one of the most widely used, governed by the differential equation: $\tau \frac{dv(t)}{dt} = -v(t) + Ri(t)$. Here, τ is the time constant, $v(t)$ represents the membrane potential, R is the membrane resistance, and $i(t)$ is the input current. For discrete-time implementations, the equation can be reformulated recurrently: $v(t+1) = v(t)e^{-\frac{dt}{\tau}} + RI(t)$ where dt is the simulation time step. The input current for neuron i in layer n is: $i_i^n(t) = b_i^n + \sum_{j=1}^{l(n-1)} w_{ij}^n z_j^{n-1}(t-1)$. Here, b_i^n is the bias

(constant current), w_{ij}^n are the synaptic weights, $z_j^{n-1}(t-1)$ represents pre-synaptic spikes, and $l(n-1)$ is the number of neurons in the previous layer. A spike is emitted if the membrane potential $v(t)$ reaches the threshold v_{thr} :

$$z(t+1) = \begin{cases} 1 & \text{if } v(t) \geq v_{thr} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

After a spike, the neuron resets its potential depending on the design: 1) Hard Reset: $v(t+1) = v_{rest}$ if $z(t) = 1$ [21] or 2) Soft Reset: $v(t+1) = v(t) - v_{thr}$ if $z(t) = 1$ [22].

a) *Encoding Strategies*: Converting continuous-domain inputs into spike-domain inputs is critical in SNNs. Common encoding methods include: i) Rate-Coded Conversion (RCC): Information is encoded in the firing rate. Given a sample $X \in \mathbb{R}^{m \times n}$, a normalized value ($0 \leq X_{ij} \leq 1$) determines the spike probability: $P(R_{ijk} = 1) = X_{ij}$, $P(R_{ijk} = 0) = 1 - X_{ij}$. Although RCC is straightforward, it requires multiple time steps for convergence, introducing latency that makes it unsuitable for tasks requiring rapid responses. ii) Delta Conversion (DC): Spikes are generated based on changes in input values, making DC particularly effective for time-series data. However, it does not produce spikes for constant inputs, which may lead to system failures in dynamic environments. iii) Latency-Coded Conversion (LCC): Information is encoded in spike timing, where higher input values result in earlier spikes. This can be implemented by supplying input values as a current to a layer of LIF neurons, which then acts as a spike encoding layer, producing an output spike train. LCC reduces firing rates, enhancing energy efficiency and inference speed, making it well-suited for real-time control systems.

b) *Training Strategies*: Training SNNs is challenging due to the non-differentiability of spike functions. Approaches can be categorized as follows: i) Biologically Accurate Methods: Techniques like Spike-Timing-Dependent Plasticity (STDP) train shallow networks but are limited in expressiveness. They have been applied to tasks like semi-supervised MNIST classification [23] and robotic control [24], [25]. ii) ANN-to-SNN Conversion: ANNs are trained and then converted into SNNs using frameworks like NengoDL [26]. While effective for tasks like autonomous driving [27], this approach may not fully exploit the unique capabilities of SNNs. iii) Gradient-Based Training: Approximate methods, such as e-prop [22], STBP [21], and Slayer [28], enable backpropagation through spikes. These methods have shown success in diverse applications, including image classification [28], anomaly detection [29], and reinforcement learning [30].

III. RELATED WORK

Thanks to their inherently dynamic nature, SNNs are well-suited approaches to learn control policies for robotics applications, which often involve interaction with unconstrained and dynamic environments. Several studies have investigated the use of SNNs in a DRL framework, including [24], [30]–[34], where researchers successfully solved various OpenAI Gym Atari games [35] using either Spike-Timing-Dependent Plasticity (STDP) or ANN-to-SNN conversion techniques. Additionally,

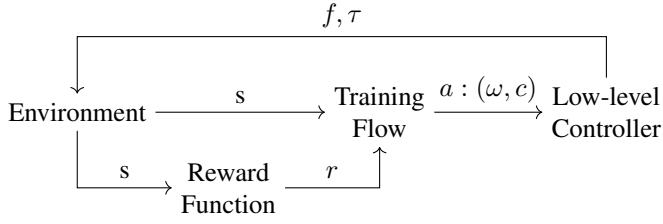


Fig. 1: Reinforcement learning Framework for Training. The framework illustrates the interaction between the environment, agent, reward function, and the low-level controller in reinforcement learning process.

other works [20], [22], [36], [37] have demonstrated the effectiveness of gradient-based training methods for similar tasks.

In this study, we compare the performance of feed-forward SNNs and ANNs on an agile flight task in which a quadrotor must navigate through a sequence of fast-moving gates. Agile drone flight—particularly drone racing—has become a widely adopted benchmark for evaluating quadrotor control strategies [38]. While [16] focused on scenarios with static gates, other studies such as [6], [17] addressed more complex settings involving dynamic environments with moving obstacles, using learning-based approaches.

SNNs have also shown strong potential in real-world applications. For example, [39] implemented a neuromorphic PID controller using an SNN and tested it on a physical blimp. Due to the network’s compact size, training was conducted via evolutionary algorithms rather than gradient-based methods. Similarly, [4] proposed a fully neuromorphic end-to-end vision-control pipeline, where the vision module was trained in a self-supervised manner and the control module was optimized with a genetic algorithm. This system was evaluated in the same real-world setting used for data collection. Finally, the authors of [40] applied an SNN trained through imitation learning for real-time attitude estimation and control on a physical quadrotor, demonstrating the feasibility of low-latency, energy-efficient neuromorphic controllers in real-world scenarios.

However, in all of these works, SNNs were not trained using DRL, and the environments were static and free of obstacles. In contrast, our work presents a fully neuromorphic, state-based control pipeline in which an SNN is trained using the PPO algorithm in a dynamic, obstacle-rich environment.

IV. METHODS

A. Framework Overview

The proposed framework, outlined in Figure 1, integrates a physics simulation environment, an SNN-based agent trained with the PPO algorithm, and a low-level controller interfacing the SNN agent and the environment. To train the SNNs and the ANNs we use SpikeGym [20], which is built on skrl [41]. The environment, implemented in the Isaac Gym simulator [42], reproduces the scenario of a quadrotor navigating through five consecutive fast-moving gates defined in [6], [17]. At

Training Flow

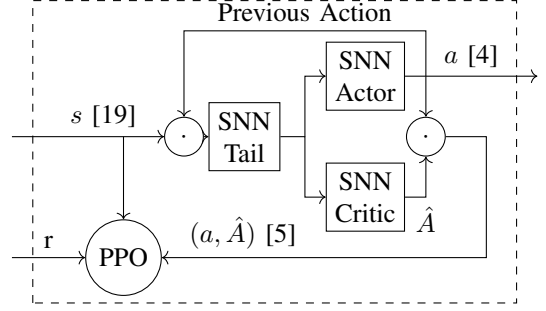


Fig. 2: Schematic structure of the training flow. The schematic shows the information flowing between components and, in square brackets, their dimensions. The symbol dot represents the concatenation operation.

every simulation step, it returns the state of the system $s \in \mathbb{R}^{19} := (d_{tgt}, q, v, \omega, d_{gate}, v_{gate})$, where d_{tgt} is the distance between the quadrotor and the final hovering target, q , v , and ω are respectively the attitude quaternion, the velocity, and the angular velocity of the quadrotor expressed in body-frame coordinates, while d_{gate} is the distance between the quadrotor’s position and the position of the next gate’s center and v_{gate} is the velocity of the gate’s center expressed in world-coordinate frame. During the interaction with the environment, a reward signal r generated by a custom reward function is provided to the SNN policy. As shown in Figure 2, the SNN processes the input state and generates common thrust and body rates (CTBR) through the actor head, which is then translated by the low-level flight controller into input forces for the physical simulator. Concurrently the critic head generates the estimated advantage value \hat{A} used by the training algorithm. This closed-loop design enables the SNN agent to learn agile flight policies through successive interactions.

B. Spiking Neural Network Architecture

We conducted a grid search to identify the SNN architecture that yielded the highest reward signals. Our results indicate that SNNs with 2 hidden layers, each containing 1,024 neurons, consistently achieved the highest reward. Consequently, this architecture was selected for our study. LIF neurons are tailored to enable efficient temporal processing in this task. The hyperparameters were selected from [20] where the authors explored the behavior of SNNs in several robotics tasks. We used a decay rate of 0.2, allowing membrane potentials to gradually diminish and enabling the network to capture relevant temporal dependencies while discarding outdated signals. A fixed firing threshold of 0.5 provides consistent spike generation. To facilitate effective training, we adopted a sigmoid surrogate gradient function, which combines the temporal dynamics of spiking neurons with the smooth gradient propagation of continuous activations.

C. Reward Function Design

In this task, the quadrotor must efficiently navigate through a sequence of swinging gates while avoiding collisions, which requires optimizing both precision and agility. To achieve such behavior, we propose the following reward formulation:

$$r_t = r_{gate,t} + r_{pos,t} + r_{pos,t} \cdot (r_{up,t} + r_{spin,t}) - \beta \cdot r_{acc,t} \quad (2)$$

The $r_{gate,t}$ component is a one-time reward for passing each gate. If the agent passes through the gates from the front, the one-time reward is multiplied by 1.5 to encourage front-passing and avoiding unnecessary detours.

The $r_{pos,t}$ component is introduced to assess the position of the quadrotor:

$$r_{pos,t} = \frac{0.05}{1 + d_{tgt}^2} + \frac{2.5}{1 + d_{gate}^2} - \frac{2.0}{1 + d_{ring}^2} \quad (3)$$

where d_{tgt} measures the distance between the quadrotor and the hovering target, d_{gate} measures the distance between the quadrotor and the next gate center, and d_{ring} measures the distance between the quadrotor and the closest point on the edge of the gate. We combined d_{tgt} and d_{gate} to incentivize fast navigation through the gates. Since the reward is sparse, faster traversal allows the agent to spend more of the remaining episode hovering near the target, thereby minimizing d_{tgt} and maximizing overall reward. This design incentivizes the agent to learn efficient gate traversal strategies. The distance d_{ring} measures proximity to the gate boundary and is computed by projecting the quadrotor's position onto the gate's YZ-plane and calculating the Euclidean distance to the gate edge. Penalizing proximity to the gate edges encourages the quadrotor to avoid risky trajectories close to obstacles, thus reducing collision likelihood.

The terms $r_{up,t}$ and $r_{spin,t}$ are introduced to encourage stability by rewarding alignment between the agent's body-frame and world-frame z-axes, and by promoting minimal rotation around the body-frame z-axis. Lastly, the term r_{acc} represents the penalty for sudden changes in linear and angular accelerations. Sudden changes can destabilize the agent. Penalizing them encourages smoother trajectories.

D. Simulation Environment

1) *Task Scenario*: Before the episode begins, each gate is initialized with a starting angle uniformly sampled between $-\pi$ and π . The offset between consecutive gates, ranging from 0.5 m to 9 m, is randomly selected. Each gate is influenced by gravity and swings within the Y-Z plane. The quadrotor is modeled as a rigid body. The quadrotor is initialized at a fixed position with a randomly sampled initial forward velocity (along the X-axis) ranging from 0 m/s to 10 m/s. These values are randomly sampled during training to ensure a diverse dataset, which enhances the robustness of the model by exposing it to a wide range of dynamics. These parameter ranges are chosen in accordance with [6], [17]. The objective is to navigate through this dynamic environment by successfully passing through all five gates.

2) *Simulation Hyperparameters*: Since our environment configuration is randomly sampled, we set the total training period to 500,000 time steps, with a time step duration of 16.7 ms and an overall episode length of 60 s, to encourage the PPO algorithm to explore various policies under different scenarios.

V. EXPERIMENT

In this section, we present the results obtained from the scenarios described in Section IV-D to investigate the research question of whether SNNs can serve as an effective solution for dynamic, state-based problems compared to ANNs.

A. Experiment Setting

Our framework is built on top of SpikeGym, the SNN DRL training framework introduced in [20], which itself is based on the skrl reinforcement learning library [41]. As a baseline, we selected the ANN architecture proposed in [17], which consists of four layers with ELU activation functions, but it was implemented in rl_games [43]. This ANN outperformed previous work leveraging learning-based and optimal control algorithms [6]. To ensure a fair comparison with existing SOTA approaches, we retrained the same ANN using the skrl framework within our environment, applying the reward function described in Section IV-C.

1) *Evaluation Initialization Configurations*: During evaluation, the gate offsets were set to values within the range of [0.5, 9] m, while the quadrotor's initial forward velocity (along the X-axis) was varied within [0, 10] m/s. These combinations were used to construct diverse evaluation scenarios. Smaller offsets demand higher agility from the agent to successfully navigate the gates while avoiding collisions. We tested each configuration for 40 episodes.

2) *Task Success Criteria*: In our experimental settings, an episode is successful only if the quadrotor passes through all five gates. Collisions, flying too close to the ground or too far from the hovering target, or exceeding safe roll and pitch angles relative to the ground result in episode termination.

3) *Comparative Metrics*: To compare the performance of the SNN-based agent against the SOTA method, we present heat maps of success rates across the described evaluation scenarios, as shown in Figure 3. Additionally, Figure 4 reports the time taken by the quadrotor to pass through the final gate, the average total rewards during evaluation, and the average success rates for both SNNs and ANNs. In Figure 5, we show the XY and XZ projections of the quadrotor's trajectories, along with its instantaneous speed profile. We used these trajectory visualizations to analyze flight smoothness, positional precision, and the agent's speed management along the trajectories.

B. Main Results

1) *Reward*: We modified the original reward function described in [17] by adjusting the weights and introducing additional terms. These modifications encourage front-facing gate passage while penalizing proximity to gate edges. We trained both ANNs and SNNs using this revised reward



Fig. 3: Heat maps comparing success rates between (a) SOTA and (b) our SNN-based method across various agent initialization and environment configuration. The experiments evaluate performance over 40 episodes with varying initial velocities (0-10 m/s) along the x-axis of the agent and gate separation distances (0.5-9 m).

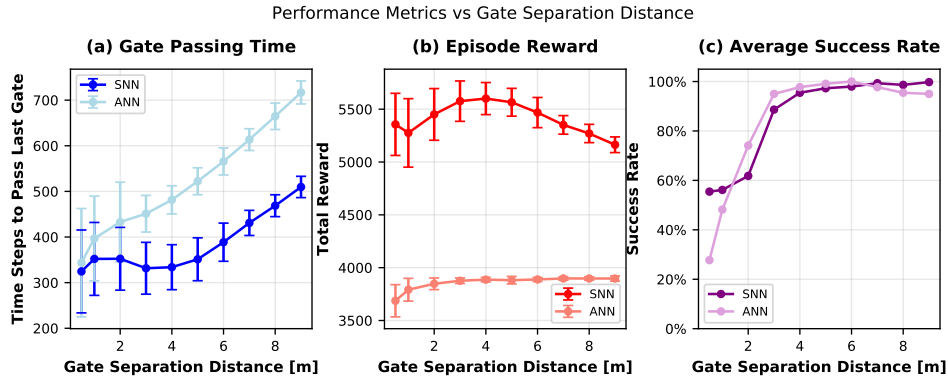


Fig. 4: Comparison between SNN (our method) and ANN (SOTA) across three performance metrics: a) completion time of gate passage, b) average total reward, and c) average success rate, plotted against gate separation distances. Each data point represents the average over 440 trials (40 episodes for each of the 11 initial velocity settings ranging from 0-10 m/s). Error bars indicate the maximum and minimum values across all trials.

function, which serves as a standardized baseline for comparing SNNs and ANNs. As shown in Figure 4, our SNN-based method achieves higher average episode rewards across all offset settings. This superior performance can be attributed to two key factors: (i) a higher success rate in the gate passage task, and (ii) faster completion times for passing through all gates.

2) *Success Rate*: Figure 3 presents heat maps comparing the average success rates of the ANN-based and SNN-based controllers under different scenarios. The experimental results show that decreasing the gate separation distance increases the likelihood of collisions, thereby making task completion more challenging. The SOTA exhibits a significant performance

degradation when gate separation distances is below 1 m . In contrast, our controller maintains higher success rates at low gate separations, achieving an average success rate of 55.5% compared to the SOTA's 27.7% when offset = 0.5 m . SNNs achieve a total average success rate of 85% across all conditions while the SOTA achieves 83%. While the margin is modest, the SNN shows more robust behavior, especially at higher initial velocities (7-10 m/s) and lower offset scenarios, where solving the task becomes increasingly difficult. As shown in the heat maps, the SNN handles these challenging scenarios with greater performance consistency, suggesting better adaptability to rapid state changes. We attribute this advantage to the SNN's temporal processing characteristics, which enable it to respond

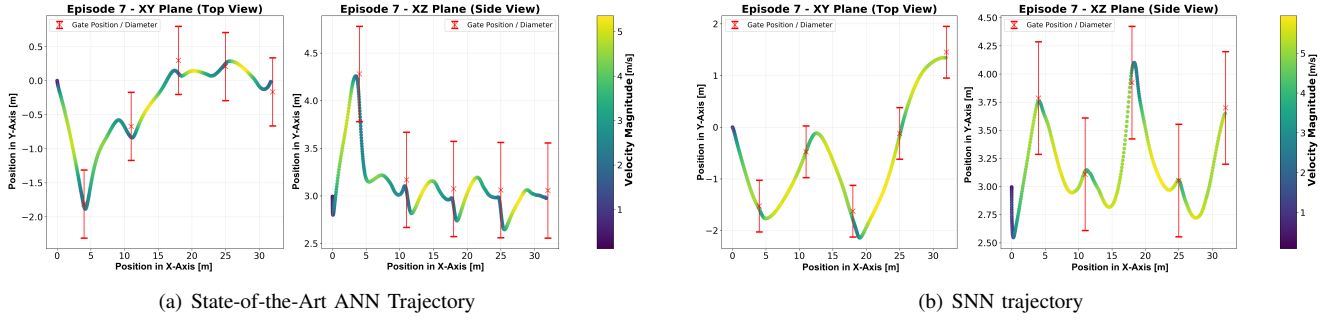


Fig. 5: Comparison of agent trajectories between (a) SOTA and (b) SNN-based methods shown in XY (top view, left) and XZ (side view, right) planes. The trajectories are colored according to velocity magnitude, captured from a single episode with 0 m/s initial velocity and 7 m gate separation. Red dots represents the final position before gate passage, with error bars showing gate radius. Blue bars represents the hovering positions.

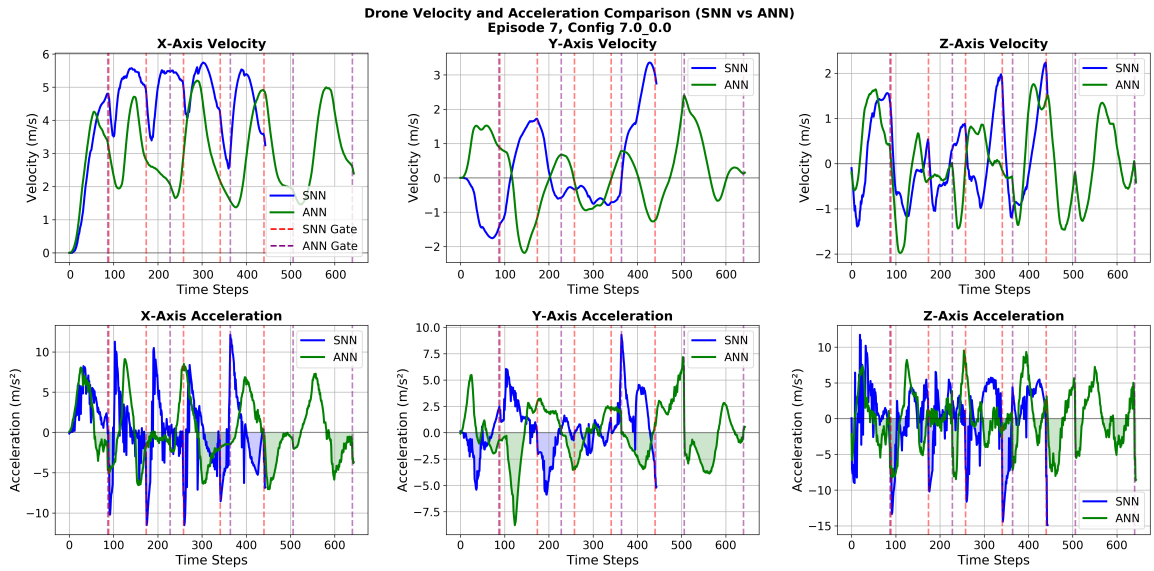


Fig. 6: Comparison of quadrotor dynamics between SNN (blue) and ANN (green) controllers during a single evaluation episode with 7 m gate separation and 0 m/s quadrotor initial velocity. The plots display velocity and acceleration components along the X, Y, and Z axes. Gate passage events are marked by vertical dotted lines, differentiated for SNN (red) and ANN (purple).

more effectively to dynamic conditions.

3) *Completion Time*: Beyond success rates, we evaluated the controllers' efficiency in completing the gate-passing task. Figure 4 compares the mean completion times (measured as time steps to pass the last gate) across different offsets. Our method demonstrates lower completion times across all configurations. At smaller offsets (0.5-2 m), both controllers show relatively similar completion times due to the shorter physical distance to traverse. However, as gate separation increases, the SNN exhibits significantly faster completion times, with a difference of approximately 200 time steps at 9 m separation.

This performance advantage can be attributed to the SNN's spike-timing-based processing, enabling it to better adapt control commands in response to rapid changes in velocity and dynamic gate movements. The speed profiles in Figure

5 highlight a key distinction between the two controllers: the SNN consistently maintains higher speeds throughout the trajectory, with an average speed of 4.03 m/s, compared to the ANN's slower trajectory with an average speed of 2.88 m/s — representing a 40% increase in speed for the SNN. However, the reason for the lower completion time isn't solely due to higher speed. Upon further analysis of the quadrotor's dynamics, the SNN's advantage actually lies in its responsiveness to environmental changes—despite having limited information (i.e., only the next gate is known at any time). This responsiveness is reflected in the acceleration metrics shown in Figure 6. SNN-based controllers exhibit substantially higher negative acceleration—7.5%, 8.6%, and 23.4% higher in the X, Y, and Z axes, respectively—resulting in a total acceleration magnitude 21.7% greater than ANNs. SNN's negative jerk magnitude is overall 57.0% greater than

that of ANNs. While higher jerk implies reduced smoothness, in this context it shows rapid control adjustments rather than instability. The proposed method's higher acceleration enables the quadrotor to brake, change direction, and accelerate toward the next gate almost immediately after passing each one. This behavior is shown in Figure 5, which results in a higher overall speed profile and, consequently, lower task completion times. Despite both networks are feed-forward, the LIF model equips the SNN with the temporal context. This property enriches the SNN controller, making it capable of extracting information from the history of the input signal and adapt control commands in real-time to meet environmental constraints. In contrast, ANNs process observations without explicit memory of past states, limiting their capacity to learn nuanced acceleration strategies. This difference contributes to the SNN's stronger dynamic behavior and more aggressive control, while the ANN exhibits a more conservative approach.

4) *Energy and Memory Analysis*: SNNs are widely recognized for their energy efficiency, enabled by their event-driven computation paradigm in which neurons fire only when their membrane potential surpasses a defined threshold. This results in sparse activation, with the majority of neurons remaining inactive at any given moment—substantially reducing power consumption compared to conventional ANNs, where all neurons engage in every forward pass. Although our study was conducted in simulation, we believe analyzing power consumption and memory usage provides valuable insights for future deployment on embedded systems.

For reference, the experiment in [4] running four SNNs totaling 28,800 neurons on Intel's Loihi [44] chips reported power consumption of 0.95 W, with 0.94 W attributed to idle power and only an additional 7–12 mW used for inference processing. In contrast, our proposed model comprises a single SNN with two hidden layers of 1,024 neurons each—totaling 2,048 neurons, which is significantly fewer than the aforementioned setup. Spike propagation involves only additions, as opposed to multiplications, and neuron activations demonstrate a sparsity of approximately 15–20% [40]. Given our architecture—an input layer of 23 neurons, two hidden layers of 1,024 neurons, and an output layer of 4 neurons—the theoretical number of additions per timestep is 1,076,224. However, due to the sparsity, the actual number of operations reduces to approximately 215,000 additions.

We also estimate the memory footprint of our model. The network learns a weight for each synaptic connection, a time constant (τ) for each layer, and a bias for each neuron. Since the threshold and decay rate are fixed, they do not contribute to the parameter count. We used *float32* representation for our model parameters. In total, the SNN has approximately 1,078,280 parameters, corresponding to an estimated memory footprint of around 4.3 MB. Considering *int8* quantization for embedded systems deployment, the estimated memory occupation reduces to 1.08 MB.

VI. CONCLUSIONS

This work investigates the potential of SNNs as a solution for dynamic, state-based control problems, in particular traversing fast moving gates, leveraging their inherent temporal processing capabilities. Our experimental results show that feed-forward SNN-based controllers achieve higher success rates than SOTA feed-forward ANNs, particularly in scenarios with low gate separation, where agility is critical. The advantage lies in the SNN's responsiveness to dynamic changes, as the spike-timing-based processing enables rapid decision-making immediately after passing each gate, resulting in a 2.5% improvement in success rate, a 40% increase in average flight speed, and a 28.6% reduction in completion time compared to ANNs. Moreover, the SNNs has a low estimation of memory footprint, making it suitable for embedded systems deployment. During our investigation, we noticed a divergence in optimal reward function design between SNNs and ANNs. This observation suggests that neural architectures may require specialized reward structures to achieve optimal performance. This finding opens an important area for future research: understanding the relationship between network architecture and reward function design. Our future work will focus on improving the interpretability of these differences, aiming to provide principled guidelines for reward function design across different neural architectures in DRL applications.

REFERENCES

- [1] A. Loquercio, E. Kaufmann *et al.*, "Learning high-speed flight in the wild," *Science Robotics*, vol. 6, no. 59, p. eabg5810, 2021.
- [2] S. Ghosh-Dastidar and H. Adeli, "Spiking neural networks," *International journal of neural systems*, vol. 19, no. 04, pp. 295–308, 2009.
- [3] A. Tavanaei, M. Ghodrati *et al.*, "Deep learning in spiking neural networks," *Neural networks*, vol. 111, pp. 47–63, 2019.
- [4] F. Paredes-Vallés, J. J. Hagenars *et al.*, "Fully neuromorphic vision and control for autonomous drone flight," *Science Robotics*, vol. 9, no. 90, p. eadi0591, 2024.
- [5] G. Orchard, E. P. Frady *et al.*, "Efficient neuromorphic signal processing with loihi 2," in *2021 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE, 2021, pp. 254–259.
- [6] Y. Song *et al.*, "Policy search for model predictive control with application to agile drone flight," *IEEE Transactions on Robotics*, vol. 38, no. 4, pp. 2114–2130, 2022.
- [7] P. Foehn, A. Romero *et al.*, "Time-optimal planning for quadrotor waypoint flight," *Science robotics*, vol. 6, no. 56, p. eabh1221, 2021.
- [8] M. Neunert, C. De Crousaz *et al.*, "Fast nonlinear model predictive control for unified trajectory optimization and tracking," in *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2016, pp. 1398–1404.
- [9] E. Kaufmann, L. Bauersfeld *et al.*, "A benchmark comparison of learned control policies for agile quadrotor flight," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 10 504–10 510.
- [10] K. Huang *et al.*, "Datt: Deep adaptive trajectory tracking for quadrotor control," *arXiv preprint arXiv:2310.09053*, 2023.
- [11] J. Hwangbo *et al.*, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.
- [12] D. Silver, A. Huang *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.
- [13] V. Mnih, K. Kavukcuoglu *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [14] N. Rudin, D. Hoeller *et al.*, "Learning to walk in minutes using massively parallel deep reinforcement learning," in *Conference on Robot Learning*. PMLR, 2022, pp. 91–100.

- [15] J. Schulman, F. Wolski *et al.*, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
- [16] E. Kaufmann *et al.*, “Champion-level drone racing using deep reinforcement learning,” *Nature*, vol. 620, no. 7976, pp. 982–987, 2023.
- [17] S. Mengozzi, L. Zanatta *et al.*, “Towards nano-drones agile flight using deep reinforcement learning,” in *2024 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*, 2024, pp. 1–6.
- [18] U. Thakker, G. Dasika *et al.*, “Measuring scheduling efficiency of rnns for nlp applications,” *arXiv preprint arXiv:1904.03302*, 2019.
- [19] U. Thakker, J. Beu *et al.*, “Run-time efficient rnn compression for inference on edge devices,” in *2019 2nd Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMCC2)*. IEEE, 2019, pp. 26–30.
- [20] L. Zanatta, F. Barchi *et al.*, “Exploring spiking neural networks for deep reinforcement learning in robotic tasks,” *Scientific Reports*, vol. 14, no. 1, p. 30648, 2024.
- [21] Y. Wu, L. Deng *et al.*, “Spatio-temporal backpropagation for training high-performance spiking neural networks,” *Frontiers in neuroscience*, vol. 12, p. 331, 2018.
- [22] G. Bellec, F. Scherr *et al.*, “A solution to the learning dilemma for recurrent networks of spiking neurons,” *Nature communications*, vol. 11, no. 1, pp. 1–15, 2020.
- [23] P. U. Diehl and M. Cook, “Unsupervised learning of digit recognition using spike-timing-dependent plasticity,” *Frontiers in computational neuroscience*, vol. 9, p. 99, 2015.
- [24] S. A. Lobov, A. N. Mikhaylov *et al.*, “Spatial properties of stdp in a self-learning spiking neural network enable controlling a mobile robot,” *Frontiers in neuroscience*, vol. 14, p. 88, 2020.
- [25] Z. Bing, C. Meschede *et al.*, “End to end learning of spiking neural network based on r-stdp for a lane keeping vehicle,” in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 4725–4732.
- [26] D. Rasmussen, “NengoDL: Combining deep learning and neuromorphic modelling methods,” *arXiv*, vol. 1805.11144, pp. 1–22, 2018. [Online]. Available: <http://arxiv.org/abs/1805.11144>
- [27] A. Shalumov, R. Halaly *et al.*, “Lidar-driven spiking neural network for collision avoidance in autonomous driving,” *Bioinspiration & Biomimetics*, vol. 16, no. 6, p. 066016, 2021.
- [28] S. B. Shrestha and G. Orchard, “Slayer: Spike layer error reassignment in time,” *Advances in neural information processing systems*, vol. 31, 2018.
- [29] L. Zanatta, F. Barchi *et al.*, “Damage detection in structural health monitoring with spiking neural networks,” in *2021 IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4. 0&IoT)*. IEEE, 2021, pp. 105–110.
- [30] D. Chen, P. Peng *et al.*, “Deep reinforcement learning with spiking q-learning,” *arXiv preprint arXiv:2201.09754*, 2022.
- [31] A. Mahadevuni and P. Li, “Navigating mobile robots to target in near shortest time using reinforcement learning with spiking neural networks,” in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 2243–2250.
- [32] M. S. Shim and P. Li, “Biologically inspired reinforcement learning for mobile robot collision avoidance,” in *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 3098–3105.
- [33] D. Patel, H. Hazan *et al.*, “Improved robustness of reinforcement learning policies upon conversion to spiking neuronal network platforms applied to atari breakout game,” *Neural Networks*, vol. 120, pp. 108–115, 2019.
- [34] W. Tan, D. Patel *et al.*, “Strategy and benchmark for converting deep q-networks to event-driven spiking neural networks,” *arXiv preprint arXiv:2009.14456*, 2020.
- [35] G. Brockman, V. Cheung *et al.*, “Openai gym,” 2016.
- [36] G. Tang, N. Kumar *et al.*, “Deep reinforcement learning with population-coded spiking neural network for continuous control,” in *Conference on Robot Learning*. PMLR, 2021, pp. 2016–2029.
- [37] L. Zanatta, A. Di Mauro *et al.*, “Directly-trained spiking neural networks for deep reinforcement learning: Energy efficient implementation of event-based obstacle avoidance on a neuromorphic accelerator,” *Neuro-computing*, vol. 562, p. 126885, 2023.
- [38] D. Hanover, A. Loquercio *et al.*, “Autonomous drone racing: A survey,” *IEEE Transactions on Robotics*, 2024.
- [39] T. Burgers, S. Stroobants *et al.*, “Evolving spiking neural networks to mimic pid control for autonomous blimps,” *arXiv preprint arXiv:2309.12937*, 2023.
- [40] S. Stroobants, C. De Wagter *et al.*, “Neuromorphic attitude estimation and control,” *IEEE Robotics and Automation Letters*, 2025.
- [41] A. Serrano-Muñoz, D. Chrysostomou *et al.*, “skrl: Modular and flexible library for reinforcement learning,” *Journal of Machine Learning Research*, vol. 24, no. 254, pp. 1–9, 2023. [Online]. Available: <http://jmlr.org/papers/v24/23-0112.html>
- [42] V. Makoviychuk, L. Wawrzyniak *et al.*, “Isaac gym: High performance gpu-based physics simulation for robot learning,” *arXiv preprint arXiv:2108.10470*, 2021.
- [43] D. Makoviichuk and V. Makoviychuk, “rl-games: A high-performance framework for reinforcement learning,” https://github.com/Denys88/rl_games, May 2021.
- [44] M. Davies, N. Srinivasa *et al.*, “Loihi: A neuromorphic manycore processor with on-chip learning,” *IEEE Micro*, 2018.