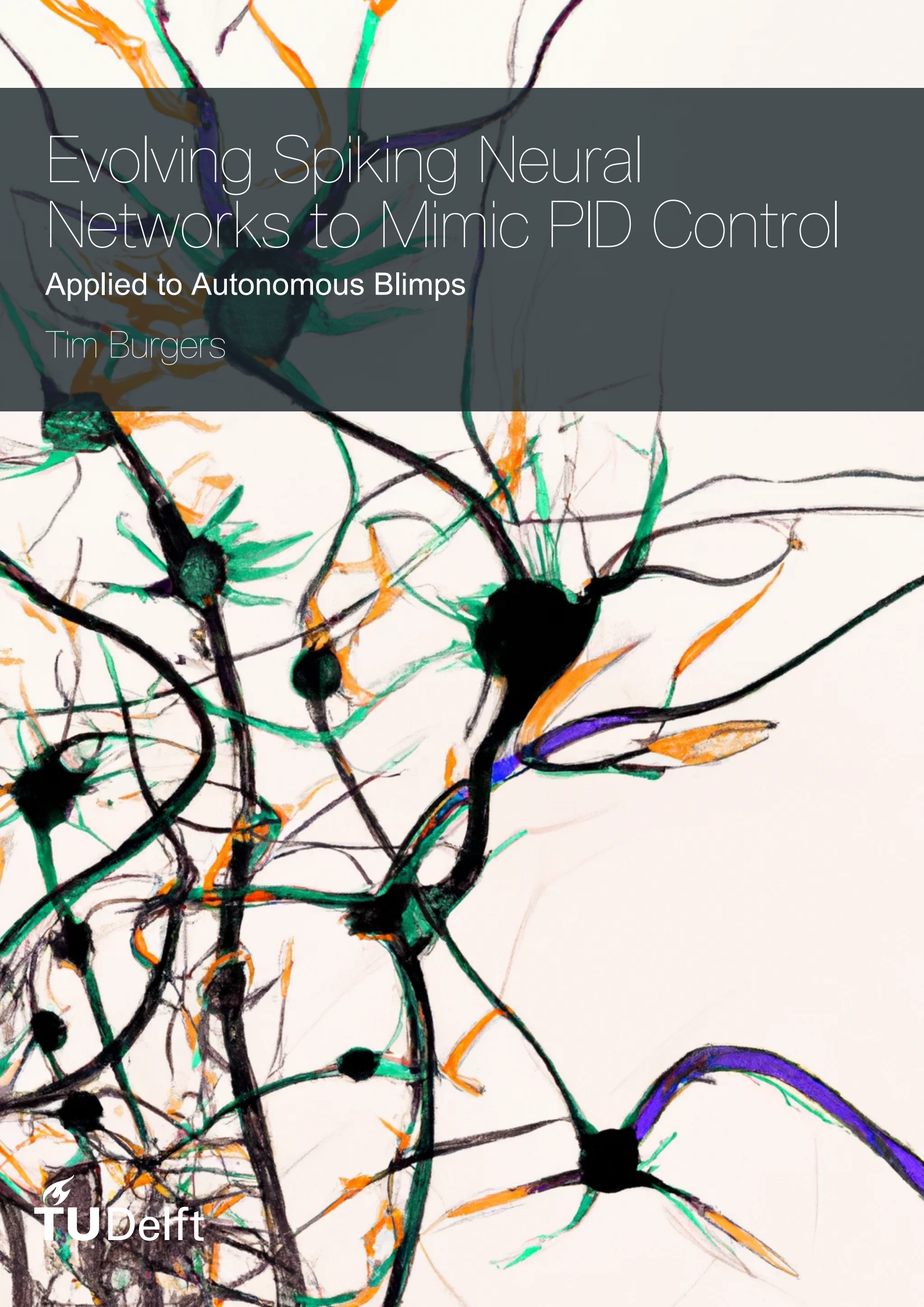
Evolving Spiking Neural Networks to Mimic PID Control

Applied to Autonomous Blimps

Tim Burgers



TU Delft

Evolving Spiking Neural Networks to Mimic PID Control

Applied to Autonomous Blimps

Thesis report

by

Tim Burgers

to obtain the degree of Master of Science
at the Delft University of Technology
to be defended publicly on October 6, 2023 at 10:30

Thesis committee:

Chair:	Prof. Dr. G.C.H.E de Croon
Daily supervisor:	Ir. S. Stroobants
Independent examiner	Dr. Ir. C. de Wagter
External examiner:	Dr. Ir. A. Bombelli
Place:	Faculty of Aerospace Engineering, Delft
Project Duration:	October, 2022 - October, 2023
Student number:	4556453

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

Faculty of Aerospace Engineering · Delft University of Technology

Preface

Working on this thesis has been a real journey. During the first year of my master's, I took a course on bio-inspired intelligence and learning for aerospace applications. It was my introduction to machine learning in academia and remains my favorite course from my master's program. In that course, I challenged myself by training an artificial neural network to solve a 2x2 Rubik's Cube using an evolutionary algorithm. Although my algorithm was only partially successful, it fueled my passion for this field of research. When I was looking for a thesis topic and Guido suggested I could focus on spiking neural networks, I was instantly intrigued. Looking back, I'm proud of a successful research process that allowed me to develop my work on the computer and test it in the real world. (although the hardware problems sometimes also delivered me some headaches and sleepless nights)

During this last year, a lot of people supported and helped me, whom I would like to thank here. First of all, I would like to express my gratitude to Stein and Guido for all their guidance throughout this year. Their insightful feedback and profound knowledge of this topic have significantly contributed to the quality of this thesis. Secondly, I would like to thank my mom and dad for their constant support. I am immensely grateful that, despite the unfortunate news last winter, my mother has made a complete recovery and can be there at my graduation ceremony. Finally, I would like to thank my fellow students and friends, Hajo, Hidde en Max, with whom I've shared office 2.40 at AE for the past year. It was a blast!

Tim Burgers
Delft, August 2023

Contents

List of Figures

List of Tables

List of Abbreviations & Symbols

List of Abbreviations

Abbreviation	Definition
ALIF	Adaptive Leaky-Integrate and Fire
ANN	Artificial Neural Network
EA	Evolutionary Algorithm
CMA-ES	Covariance Matrix Adaptation Evolutionary Strategy
CPU	Central Processing Unit
DoF	Degree of Freedom
IK	Inverse Kinematics
ISI	Inter Spike Interval
IWTA	Input-Weighted Threshold Adaptation
LIF	Leaky-Integrate and Fire
LTD	Long-term Depression
LTP	Long-term Potentiation
MAE	Mean Absolute Error
MAV	Micro Air Vehicle
MSE	Mean Square Error
NN	Neural Network
NEAT	NeuroEvolution of Augmenting Typologies
NEF	Neural Engineering Framework
PID	Proportional, Integral & Derivative
PSP	Postsynaptic Potential
PMCC	Pearson product-moment correlation coefficient
RL	Reinforcement Learning
R-LIF	Recurrent Spiking Neural Network with Leaky-Integrate and Fire Neurons
R-STDP	Reward modulated Spike-Timing Dependent Plasticity
SG	Surrogate Gradient
SNN	Spiking Neural Network
SOP	Synaptic Operation
STDP	Spike-Timing Dependent Plasticity

List of Latin & Greek Symbols

Symbol	Definition	Unit
A', B'	Matrices of the intrinsic dynamics of neural populations in NEF	[-]
F	Force	[N]
I	Postsynaptic current	[A]
K_D	Differential gain	[-]

Symbol	Definition	Unit
K_I	Integral gain	[-]
K_P	Proportional gain	[-]
$L(u)$	Loss function	[-]
R	Resistance	[Ohm]
T	Thrust	[N]
$W_{i,j}$	Weights of the presynaptic neuron i to the postsynaptic neuron j	[-]
a, b, c, d	Neuron parameter of the Izhickevich model	[-]
b	Bias	[-]
$c(t)$	Eligibility trace of the synapse	[-]
$d(t)$	Dopamine signal	[-]
g	Gravitational constant	[m/s^2]
e	Error between reference and actual state	[-]
h	Height	[m]
$h(t)$	Temporal linear filter	[-]
m	Mass	[kg]
p	Roll rate	[rad/s]
q	Pitch rate	[rad/s]
r	Yaw rate	[rad/s]
t^f	Spike timing	[s]
u	Membrane potential	[V]
$\hat{u}$	Target motor command	[-]
u	Membrane potential	[V]
x, y, z	Cartesian coordinates	[m]
Θ	Pitch angle	[rad]
ϑ	Threshold of the neuron	[V]
τ	Time constant / Decay parameter	[s]
$\rho(u, \hat{u})$	Pearson Correlation Coefficient	[-]
Φ	Roll angle	[rad]
Ψ	Yaw angle	[rad]
Ω	Rotor velocity	[rad/s]

1 Introduction

Flying robots are becoming increasingly prevalent in our society and are already employed for various purposes, ranging from agricultural applications (Menouar et al. 2017) to windmill inspections (Shafiee et al. 2021). These aerial vehicles can be used in places that would otherwise be very complex or even impossible for humans to reach. The growing interest in this robotics field leads to greater complexity in both hardware and algorithms used onboard the flying robots, often resulting in increased power demands. However, the power and weight budgets for flying robots are limited. This highlights the need for a power-efficient but computationally powerful controller for aerial vehicles.

A solution lies in analyzing how insects navigate through complex environments with precision despite their minimal weight and energy usage. This led to the development of simplified mathematical models of the brain's computational processes, called Artificial Neural Networks (ANN). Their ability to approximate non-linear functions makes them highly effective in controlling complex systems, such as quadrotors (Zhang et al. (2022) and Heryanto et al. (2017)). However, as the size of ANNs grow, so does their computational demand. Reducing the ANN's power consumption can be achieved by adopting the information transmission methods employed by biological brains. A brain uses sparse sequences of action potentials, called spikes, to communicate between neurons, whereas ANNs use continuous-valued activations. There are neural networks that use this spike-based approach to asynchronously transmit information, called a Spiking Neural Network (SNN) (Maass 1997). The availability of specialized hardware designed to execute SNNs, called *neuromorphic hardware*, enables the full potential increase in the power efficiency of SNNs by making use of the event-driven sparsity in the information transmission (Furber et al. 2014).

Research on the use of SNNs for robotic control tasks is still in the early phase of development, with one of the primary challenges being the lack of appropriate training algorithms. The SNN's temporal dynamics, sparse spiking activity, and non-differentiable spike signal make most existing ANN's training algorithms unsuitable (Taherkhani et al. 2020). Recent advancements have made it possible to apply error backpropagation to SNNs through surrogate-gradient algorithms (Neftci et al. 2019). Nevertheless, training SNNs using these gradient-based algorithms is still difficult due to the susceptibility to local minima, exploding/vanishing gradients, and sensitivity to initial conditions (Bing, Meschede, et al. 2019). Conversely, Evolutionary Algorithms (EA) are less prone to these limitations (Michalewicz 1994). At the cost of higher computational power and slower convergence, these training algorithms explore the solution space globally without gradient information and can easily be parallelized, which makes them suitable for the training of SNNs (Katoch et al. 2021).

Practical applications of SNNs in robotic control are limited due to the increased training complexity in comparison to non-spiking counterparts (Bing, Meschede, et al. (2019) and Bartolozzi et al. (2022)). Despite significant progress in recent years concerning SNNs in the control domain, training a fully neuromorphic implementation of a PID controller still remains a challenge. Some studies have manually set the connections and network parameters of the SNN to achieve integration/differentiation within the spiking network (Stagsted et al. (2020a), Stagsted et al. (2020b) and Stroobants, Dupeyroux, et al. (2022)). At the same time, others have added recurrent connections (Zaidel et al. 2021) or threshold adaption methods (Stroobants, De Wagter, et al. 2023) within the SNN to be able to mimic PID controllers. Both these studies showed limitations concerning the flexibility in the training of the network parameters and/or the accuracy of the derivative and integral controller.

Taking into account the SNN structures that enabled PID control in prior research, an SNN is evolved in this study to mimic a PID controller to effectively control the altitude of a real-world indoor blimp. The changing buoyancy and slow system dynamics make the blimp an interesting test vehicle for the SNN PID controller. To effectively control the altitude of the blimp, an integrator needs to be present to counteract the buoyancy-induced drift. Additionally, the slow responsiveness of the blimp makes it challenging for real-time control strategies, such as PID, to effectively control the blimp's altitude. In Gonzalez-Alvarez et al. (2022), an open-source indoor blimp was designed and used as a test vehicle for an evolved semi-neuromorphic altitude controller, showing adequate tracking of the reference signal. In addition to the SNN controller, a conventional non-spiking PD controller was used to remove large oscillations present. Moreover, the semi-neuromorphic controller was not trained to eliminate steady-state errors of the blimp.

Building upon this research, the main contributions of this thesis are as follows: 1) A fully neuromorphic altitude controller for a blimp was developed using an evolved SNN comprising only 160 neurons. This SNN controller effectively mitigates overshoot and oscillations while minimizing the steady-state error caused by the blimp's buoyancy. 2) The individual and combined effect of different SNN structures used in prior SNN PID research on the SNN controller's performance is analyzed. 3) Improvements were made to the hardware components of the open-source blimp, increasing the onboard computational power and the accuracy of the height measurements.

1.1. Research Objective

The main goal of this thesis is summarized in the research objective that is stated below

Developing an asynchronous neuromorphic feedback controller for accurate altitude tracking of a real-world blimp by using spiking neural networks, which are trained using a machine learning algorithm to mimic a conventional PID controller.

1.2. Structure of this Thesis

The thesis is divided into three main parts. Part I comprises the primary contributions developed in this thesis, presented in the form of a scientific article submitted to the 2024 International Conference on Robotics and Automation (ICRA). The article is split into four sections. The first section includes the introduction, highlighting the significance of this research, reviewing relevant prior research on the topic and providing a summary of the paper's contributions. Secondly, the methodology section covers the spiking neural network structure, setup of the real-world blimp experiment, evolutionary training algorithm and dataset generation methods. The third section presents the results and discussions of the performance of the different SNNs using test datasets as well as the real-world attitude-tracking performances of the controllers on the blimp. The fourth and final section of the paper contains a brief conclusion.

Part II contains the literature study of this thesis. For readers with limited prior knowledge in the control and/or machine learning domain, it is recommended to start by reading the literature study before the scientific article to acquire the necessary background knowledge on these topics. Chapter 3 presents the basics of Micro Aerial Vehicle (MAV) dynamics and PID control. Next, Chapter 4 covers the foundation of spiking neural networks. Including the model used to simulate a neuron and its connection, the methods used to encode and decode data, the training algorithms and the available neuromorphic processors. The final chapter of the literature study, Chapter 5, covers an analysis of the available SNN controllers that have been used to mimic PID controllers with and without the use of machine learning algorithms. When reading the literature study, there are two important notes to take in mind. Firstly, at the time of writing the literature study, the idea was to use a quadcopter instead of a blimp, which explains the focus on quadcopter dynamics in the literature study. Secondly, the initial research objective involved designing two different controllers: one SNN controller aimed at mimicking a PID controller without training the network parameters, and another SNN controller that is trained using a machine learning algorithm to control a quadcopter's position or angles without relying on PID control as the target. After the literature review, it has been decided to combine both methods to develop the SNN controller presented in this thesis.

The third and final part of the thesis contains the additional experimental results (Chapter 6) and a conclusion and recommendations for future work (Chapter 7). Chapter 6 presents the results that were not

included in the scientific paper due to the strict page limit set by ICRA 2024. In this chapter, all experimental results that have been recorded on the blimp are presented and discussed, together with an in-depth analysis of the step response of the developed SNN controller on the blimp.

Part I

Scientific Article

Evolving Spiking Neural Networks to Mimic PID Control applied to Autonomous Blimps

Tim Burgers
Delft University of Technology
Delft, The Netherlands

Stein Stroobants
Delft University of Technology
Delft, The Netherlands
s.stroobants@tudelft.nl

Guido C.H.E. de Croon
Delft University of Technology
Delft, The Netherlands
g.c.h.e.decroon@tudelft.nl

***Abstract*—In recent years, Artificial Neural Networks (ANN) have become a standard in robotic control. However, a significant drawback of large-scale ANNs is their increased power consumption. This becomes a critical concern when designing autonomous aerial vehicles, given the stringent constraints on power and weight. Especially in the case of blimps, known for their extended endurance, power-efficient control methods are essential. Spiking neural networks (SNN) can provide a solution, facilitating energy-efficient and asynchronous event-driven processing. In this paper, we have evolved SNNs for accurate altitude control of a non-neutrally buoyant indoor blimp, relying solely on onboard sensing and processing power. The blimp's altitude tracking performance significantly improved compared to prior research, showing reduced oscillations and a minimal steady-state error. The parameters of the SNNs were optimized via an evolutionary algorithm, using a Proportional-Derivative-Integral (PID) controller as the target signal. We developed two complementary SNN controllers while examining various hidden layer structures. The first controller responds swiftly to control errors, mitigating overshooting and oscillations, while the second minimizes steady-state errors due to non-neutral buoyancy-induced drift. Despite the blimp's drivetrain limitations, our SNN controllers ensured stable altitude control, employing only 160 spiking neurons.**

I. Introduction

Throughout history, humans have been fascinated by how animals gracefully and precisely navigate complex environments. This fascination has inspired efforts to understand the brain's computational processes behind such behavior, leading to the development of Artificial Neural Networks (ANN). These ANNs represent the neural processes using simplified mathematical models. Their ability to approximate complex non-linear functions makes them highly effective in controlling complex systems, such as quadrotors [1, 2]. However, as the size of ANNs grow, both response latency and computational demands increase. The latter poses particular challenges for robotic applications with restricted onboard energy capacity, such as flying robots. A solution may be found in the information transmission methods employed by biological brains. ANNs rely on continuous-valued signals, whereas the biological brain employs sparse spatial-temporal "spike" signals—brief, rapid increases in neuron voltage—for data encoding and transmission.

This material is based upon work supported by the Air Force Office of Scientific Research under award number FA8655-20-1-7044.

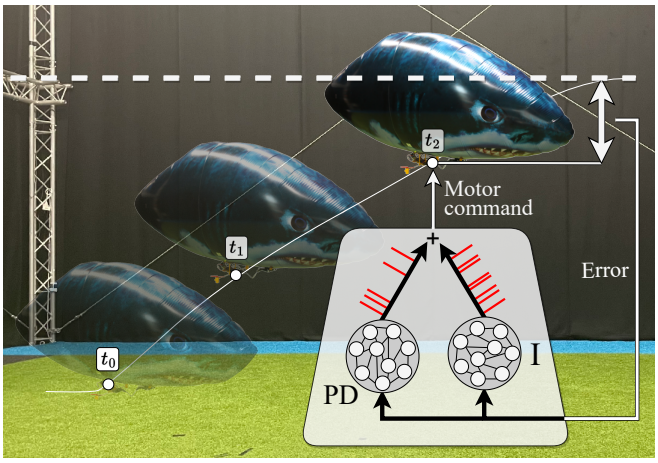


Fig. 1. The proposed SNN altitude controller for the blimp, where $[t_0,t_1,t_2]$ indicates time instances of the blimp's altitude while approaching a setpoint, marked by the dashed line.

There are neural networks that use this spike-based approach to transmitting information, called a Spiking Neural Network (SNN) [3]. These more biologically plausible neural networks show potential for energy-efficient and low-latency controllers [4]. This fact was demonstrated in a study by Vitale et al. [5], where an SNN controller in a fully neuromorphic control loop outperformed a conventional control loop on power consumption and control latency when tracking the roll angle of a bench-fixed 1 DoF birotor. However, the application of SNN controllers in robotics is still in its early stages of development. One of the biggest challenges is the availability of suitable training algorithms. The SNN's temporal dynamics, sparse spiking activity, and non-differentiable spike signal make most existing ANN training algorithms unsuitable [6]. Recent research has enabled the use of error backpropagation for SNNs by means of surrogate-gradient algorithms [7]. Nevertheless, training SNNs using these gradient-based algorithms is still difficult due to the susceptibility to local minima, exploding/vanishing gradient, and sensitivity to initial conditions [8].

Due to the increased training complexity of SNNs compared to non-spiking counterparts, practical applications of SNNs in the robotic control domain are still limited. Designing

a fully neuromorphic SNN controller to emulate low-level controllers, such as the Proportional-Integral-Derivative (PID) control, still remains a complex task. Recent research showed SNNs performing differentiation and integration within the network, by manually configuring neuron connections and weights [9, 10, 11]. In these studies, the controllers were implemented on Intel's Loihi neuromorphic processor, showcasing the promise of neuromorphic hardware by exhibiting very low latencies [12]. In Zaidel et al. [13], multiple populations with pre-determined network parameters were used to implement all three pathways of the PID controller to control a 6 Degrees of Freedom (DoF) robotic arm. The integral pathway was implemented using a fully recurrent population of neurons, while differentiation was achieved by using a slow and fast time constant for two populations. Although the integral controller succeeded in reducing the steady-state error, it was unable to completely eliminate it. In another study, spiking neurons were trained to replace the rate controller of a tiny quadrotor, the Crazyflie [14]. Integration in the SNN controller was achieved through discretized Input-Weighted Threshold Adaptation (IWTA), where the threshold depends on the previous layer's spiking activity. The training process had limitations because it relied partially on predetermined connections and grouped network parameters.

In this work, we investigate the different mechanisms, recurrency and IWTA, used in prior SNN PID research that enabled differentiation and/or integration. For each mechanism, we evolve an SNN to control the altitude of a real-world indoor blimp. The blimp is an interesting test platform for the SNN controller, allowing validation of all components of the PID controller. The changing buoyancy over time requires a good integrator to be present. Moreover, due to the high delays and slow system dynamics of a blimp, a high proportional gain is necessary in the reference PID controller. This reduces the blimp's rise time, requiring a strong derivative in the controller's output to prevent overshoot and oscillations. In Gonzalez et al. [15], an open-source indoor blimp was designed and used as a test vehicle for an evolved neuromorphic altitude controller, showing adequate tracking of the reference signal. However, even after including an additional non-spiking PD controller to the output of the SNN controller, there were still oscillations present of approximately $\pm$ 0.3m. Additionally, the SNN was only trained on a neutrally buoyant blimp. Slight changes in the buoyancy of the blimp would cause a steady state error, which the controller was unable to eliminate.

We build further on this research, presenting here the following contributions: 1) We developed a fully neuromorphic height controller for a blimp (visualized in Figure 1), using an evolved SNN of only 160 neurons that is able to minimize the overshoot and oscillations while also removing the steady-state error caused by the buoyancy of the blimp. 2) We analyze the individual and combined influence of recurrent connections and IWTA on the performance of the SNN controller 3) We made improvements to the hardware components of the open-source blimp, improving the onboard computational power and increasing the accuracy of the height measurements.

II. Methodology

The SNN controller consists of three layers of neurons, where all parameters are optimized using an evolutionary algorithm to mirror the output of a tuned PID controller. The Proportional-Derivative (PD) controller's rapid dynamics demand fast time constants, while the integral controller relies on slower dynamics and, thus, slow time constants. To facilitate the learning process, we split the controller into two separate parts based on the required time constants to model each component. After completing the training process, the evolved controllers are used to control the altitude of a helium-filled blimp. Detailed discussions on the SNN's structure, parameters, experiment setup, and the evolutionary training algorithm used in this study follow below.

A. Spiking Neural Network Controller

We use current-based Leaky-Integrate and Fire (LIF) neurons with a soft reset for the threshold (ϑ). The discretized equations that describe the dynamics of the three states of the neuron (synaptic current $i(t)$, membrane potential $v(t)$ and spike train $s(t)$) are described as follows:

$$i_i(t) = \tau_i^{\text{syn}} i_i(t-1) + \sum W_{ij} s_j(t) \tag{1}$$

$$v_i(t) = \tau_i^{\text{mem}} v_i(t-1) + i_i(t) - s_i(t-1)\vartheta_i \tag{2}$$

$$s_i(t) = H\left(v_i(t) - \vartheta_i\right) \tag{3}$$

where subscript i and j denote the post- and presynaptic neuron respectively. The discretized time constants, known as the decay parameters, of the synapses and the membrane potential are respectively referred to as τ^{syn} and τ^{mem}. The spiking behavior of a neuron is modeled using the Heaviside step function H, which outputs a spike when the membrane potential exceeds the threshold.

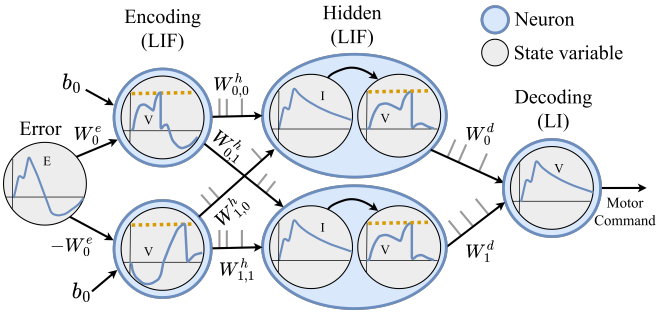


Fig. 2. The basic structure of the SNN controller. The encoding layer has an additional bias (b) added to the input current.

The basic structure of a spiking neural network controller is schematically depicted in Figure 2. The input weights, indicated by W^e, W^h and W^d, are linked to the encoding, hidden and decoding layers, respectively. The encoding layer is responsible for translating the floating-point input into a sequence of spikes, and conversely, the decoding layer performs the reverse operation.

The input to the network is the error of the controlled state. After applying the encoding weights and biases, the error signal is directly used as the synaptic current for the encoding neuron ($\tau^{syn} = 0$). To facilitate the training of the encoding layer, we paired neurons with shared bias and flipped weight sign. This results in symmetric encoding, ensuring similar spiking patterns for both positive and negative errors. The effect that the input weight and bias have on the spiking behavior of a LIF neuron is presented in Figure 3. The encoding layer incorporates a bias to achieve spike activity for the encoding of error values close to zero, which would otherwise be impossible.

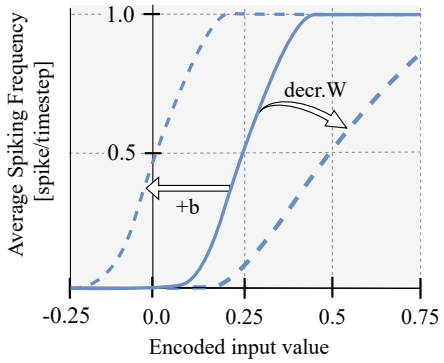


Fig. 3. Influence of the input weight and bias on the spiking frequency of a LIF neuron.

We study the effect of using different types of structures for the hidden layer of the SNN controller in this work. The most basic type of hidden layer (LIF) is depicted in Figure 2. In addition to the basic structure, we evaluated the influence of recurrency [13, 16] and Input-Weighted Threshold Adaptation (IWTA) [14], as both these network structures demonstrated their essential role in enabling integration within the SNN. We focused solely on threshold adaptation linked to the incoming spiking activity rather than the hidden neuron's activity itself. An overview of the different hidden layer structures is shown in Figure 4. In contrast to the original implementation in [14], the threshold for the IWTA-LIF neurons is modeled using a decay term (τ^{th}), where the threshold converges back to a base value (ϑ), after an increase/decrease (W^{th}) caused by an incoming spike. This method of implementing IWTA adds new dynamics to the threshold and increases the solution space.

The spiking neural network controller is decoded using a single leaky-integrator neuron, that calculates the exponential moving average of the spikes in the hidden layer.

B. Real-World Experiment

To validate the performance of the SNN controller on a real-world application, we implemented an SNN altitude controller for an open-source micro-blimp developed in [15]. The combination of the buoyancy-caused drift and slow system dynamics make the blimp a useful test vehicle for this research. The input of the SNN height controller is the difference between a reference altitude and the onboard lidar

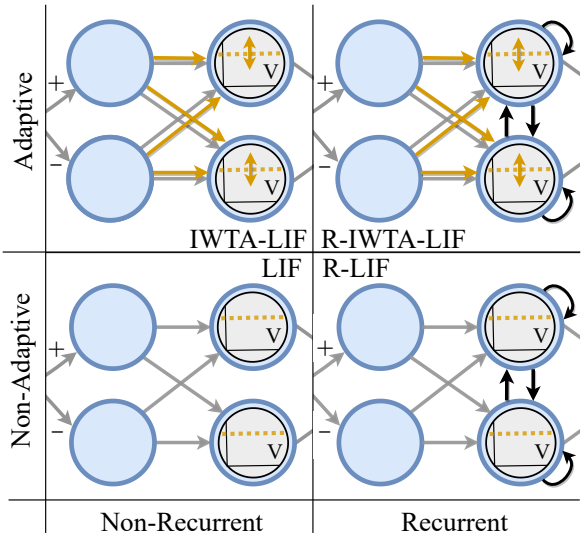


Fig. 4. Visualization of the different hidden layer structures. The two left and right circles represent the encoding and the hidden neurons respectively.

measurement, which makes it a fully on-board closed-loop control system. The output of the SNN is the motor command, $u \in [-3.3, 3.3]$, where $u < 0$ indicates downward and $u > 0$ upwards movement. The blimp's control system consists of two coreless direct current (DC) motors attached to a 180-degree rotating shaft, which enable the rotors to push the blimp both up- and downwards. A visual representation of the blimp and its hardware components is provided in Figure 5. Two improvements have been made to the blimp's hardware setup. Firstly, to improve the altitude tracking ability, we implemented a LiDAR sensor, the TFmini S LiDAR-module, which significantly increased the accuracy from $\pm$ 20cm to $\pm$ 1cm. Secondly, the Raspberry Pi Zero 2 W, running on Ubuntu 20.04, replaced its predecessor to increase the processing power and prevent software compatibility issues. The total weight of all hardware components attached to the blimp is 140g. In order to ensure smooth communication between all system components, we have used the Robot Operating System (ROS1) framework. The control loop runs at a rate of 10 Hz.

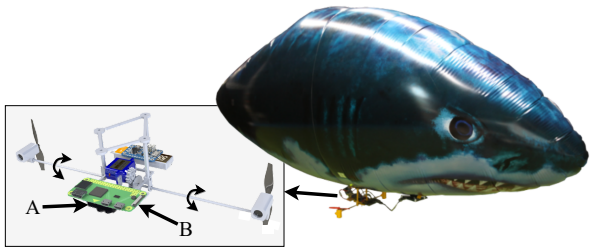


Fig. 5. The open-source micro-blimp used for the real-world experiments [15]. Two adjustments made on the blimp are: A) TFmini S LiDAR-module B) Raspberry Pi Zero 2 W.

C. Evolutionary Training Algorithm

The training process for the SNN controller employs an evolutionary algorithm, consisting of two recurring steps: population creation and evaluation, with each cycle representing one generation of the evolution. In the first step, a set number of individuals forms the population, each representing a unique controller with slightly varying parameters. The second step ranks these individuals based on performance via a cost function, and this information guides the creation of the new population. Further details for each step are discussed below.

Every training loop was executed on the DelftBlue supercomputer, running on 20 cores for 50,000 generations with 50 individuals [17]. To prevent overfitting, we randomly sampled one of the 100 training datasets to evaluate each generation. Additionally, both the size and the frequency of the step inputs used in each dataset were varied to enhance the diversity of the training data. We conducted a total of 30 training loops for each combination of controller type (2) and hidden layer structure (4), resulting in a total of 240 training sessions. Each controller is limited to 80 neurons to showcase the potential of small-scale networks for neuromorphic control.

1) Population Creation: For the creation of a new population, we have used a Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) [18]. CMA-ES is a distribution-based optimization algorithm that iteratively adjusts the mean and covariance matrix of a multi-variate Gaussian distribution, from which all network parameters of each individual are sampled.

The CMA-ES is implemented using the *Evotorch* Python library [19]. Every network parameter that needs to be evolved is initialized using the mean and standard deviation of a Gaussian distribution. Strict-bounded parameters, such as the decay constants in the neuron model, are constrained by rejecting and resampling. The initial mean is set by sampling from a uniform distribution within parameter bounds, while the standard deviation is set to 1/10th of the parameter's range. An overview of all trained parameters including the bounds is provided in Table I, where W^r represents the recurrent weights.

TABLE I
OVERVIEW OF ALL PARAMETERS AND BOUNDS USED TO EVOLVE THE SNN, WITH ADDITIONAL PARAMETERS IN THE HIDDEN LAYER FOR *RECURRENCY AND †IWTA

	Param.	Size	Bounds
Enc.	W^e	N	[-2, 2]
	b	N	[-1, 1]
	τ^{mem}	N	[0,1]
	ϑ	N	[0,10]
Dec.	W^d	N	[-1,1]
	τ^{mem}	N	[0,1]

	Param.	Size	Bound
Hidden	W^h	2N	[-2, 2]
	τ^{syn}	N	[0, 1]
	τ^{mem}	N	[0, 1]
	ϑ	N	[0, 10]
	$^*W^r$	NxN	[-1, 1]
	$^\dagger\tau^{th}$	N	[0,1]
	$^\dagger W^{th}$	NxN	[-1, 1]

2) Population Evaluation: The performance of each individual SNN controller is determined by comparing the output of the SNN controller (u) to the output of a tuned PD or integral controller ($\hat{u}$). The SNN is tasked to learn the mapping between the input signal (e) to the output of the target controller ($\hat{u}$). The discretized equation that describes the PID response is provided below:

$$\hat{u_k} = \underbrace{K_p e_k + K_d \frac{e_k - e_{k-1}}{T}}_{\text{PD controller}} + \underbrace{K_i (\sum_{k=0}^{k} T e_k)}_{\text{Integral controller}} \tag{4}$$

where K_p, K_i, and K_d refer to the proportional, integral and derivative gains respectively and T is denoted by the sampling period. Both the input error signal and the target controller response are recorded in a dataset.

To quantify the performance of the SNN compared to the target controller, the Mean Absolute Error (MAE) was used as the main term in the cost function. The MAE was used instead of the mean square error (MSE) to prevent over-penalization of the error that is created during the transient response to a step input because this led to more oscillations in the steady state. Additionally, the cost function was augmented with the Pearson Correlation Coefficient (PCC) [20], denoted by ρ, to incentivize that the sign of the output of the SNN and PD/I controllers is equal. The PCC measures the linear correlation between two signals, ranging from -1 to 1, where the latter indicates a fully linear relation. Since the fitness function is a minimization function, the PCC is included by adding $1 - \rho$ to the MAE. This results in the following cost function that was used in the population evaluation step of the evolutionary training process:

$$L(u, \hat{u}) = \text{MAE}(u, \hat{u}) + (1 - \rho(u, \hat{u})) \tag{5}$$

D. Dataset Generation

The methods used to gather the test and training data for each controller are discussed below.

1) PD controller: To successfully train the PD SNN controller, we need a broad spectrum of error signals. Therefore, we used a semi-randomly tuned PID controller on the neutrally buoyant real-world blimp. The error signal of these recordings is used as the SNN input data of the dataset. The target signal for the training algorithm is generated by passing the recorded error signal through a PD controller with the tuned gains for the blimp's altitude controller.

2) Integral controller: The Integral SNN has to learn to integrate the error within the SNN itself. For this training process, the decay parameter of the decoding neuron is purposely bound to ensure a quick decay (eg. [0-0.3]) in order to prevent the algorithm from converging to a slow decay. A slow decoding decay parameter would imply that the integration only happens in the decoding neuron, instead of in the hidden layer.

If the buoyancy remains constant throughout the training process of the integral controller, the network might learn to add a bias to counteract the buoyancy. Therefore we must adjust the buoyancy during training to ensure that the network learns to integrate information over time. Instead of recording multiple datasets with varying buoyancy, we decided to train the Integral controller on a model where we could also change

the "buoyancy" within a single dataset to facilitate the learning process.

The blimp was modeled by the double integrator control problem and controlled using a PID controller. The integral gain was set to match that of the tuned real-world blimp, while the PD gains were adjusted to align the model's dynamics approximately with those of the tuned PID-Blimp system. In the double integrator system, the output of the controller, $u(t)$ is directly proportional to the second derivative of the state plus an additional bias: $\ddot{x}(t) = u(t) + b$. The bias simulates the level of buoyancy of the blimp. Every time a step input is received, the bias is randomly sampled ($U(-4, 4)$). Each dataset consists of 5 step inputs maintained for 50s.

III. Results

The first subsection displays the training outcome of both SNN controllers using a test dataset, followed by the assessment of their performance on an actual blimp. The results also contain a performance analysis of various neural mechanisms within the hidden layer.

A. Training of the SNN Controllers

1) PD SNN Controller: The result of the PD training process is provided in Figure 6, showing a single step response from the test dataset. The solid red line represents the SNN's target, while the dashed red line depicts the proportional controller. The P controller is included to visualize the additional effect of the derivative. Initially, all spiking PD controllers show clear influences of the derivative, as they match the target signal. However, after the initial damping of the proportional output, the controllers start to diverge. To prevent overshoot and oscillations, the derivative controller should counteract the proportional controller when the state is approaching the setpoint, which happens around 4 seconds in the Figure. The only controller that counteracts the P controller sufficiently is the LIF SNN. Based on this analysis and the lowest loss value across the entire test dataset, shown in Table II, we opted to use the LIF SNN for the blimp's altitude controller. The larger solution space for the recurrent and IWTA neuron structures makes the search space more complex and leads to local minima.

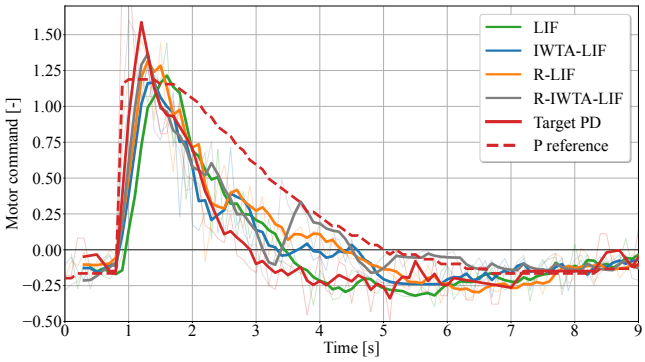


Fig. 6. The moving average of the step responses of all evolved SNN controllers compared to a PD target signal using a test dataset. The LIF shows derivative action by damping the command.

TABLE II
Loss for SNN PD controllers (u) on complete test dataset (500s) using tuned PD as target ($\hat{u}$)

	LIF	IWTA-LIF	R-LIF	R-IWTA-LIF
$L(u, \hat{u})$	0.68	0.74	0.74	0.73

2) Integral SNN Controller: The result of the integral training process is provided in Figure 7, showing the response of the different hidden layer mechanisms to a test dataset. The LIF SNN failed to learn to integrate, hence, it is excluded from the figure. All three spiking controllers following the test signal, without a clear standout performer. To see how well the controllers perform in the real-world, they are all tested on the indoor blimp.

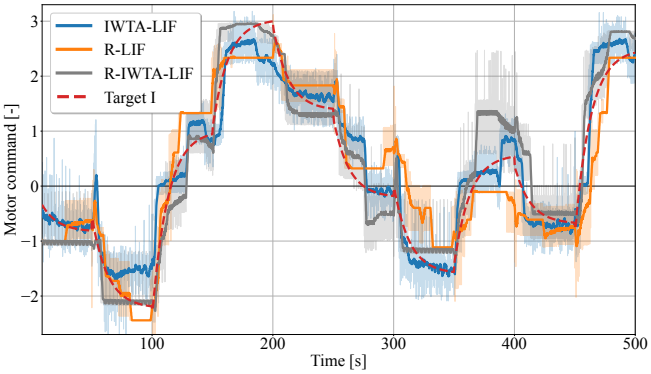


Fig. 7. The moving average of the step responses of three evolved SNN controllers compared to a integral target signal using a test dataset, with a changing bias every step input.

B. Performance of SNN controlling Real-World Blimp

To analyze the performance of each controller separately, we first test the PD SNN controller using a neutrally buoyant blimp. Afterward, we added some weight to the blimp to make it negatively buoyant. The negatively buoyant blimp is used first to evaluate the performance of the SNN I controllers, followed by the evaluation of the fully spiking controller.

1) PD control of neutrally buoyant blimp: We assess the performance of the PD SNN controller with LIF hidden structure by comparing it to a tuned conventional PD controller. The tracking accuracy is tested using five different step sizes Δh=[1,0.5,0,-0.5,-1], maintained for 50s each and the results are shown in Figure 8. Both the conventional PD and the SNN show small oscillations, $\pm$6 cm, around the setpoint. These oscillations are caused by the discretized mapping of the motor command to the actual voltage sent to the motors using PWM. This causes a deadzone to be present in the motor command signal, which is the region of the motor commands, u =[-0.1,0.1], that does not result in the actuation of the rotors.

2) Integral control of negatively buoyant blimp: To isolate the effect of the spiking integrator, we used the non-spiking PD controller in combination with the spiking integrator to control the negatively buoyant blimp. The result of two-step responses

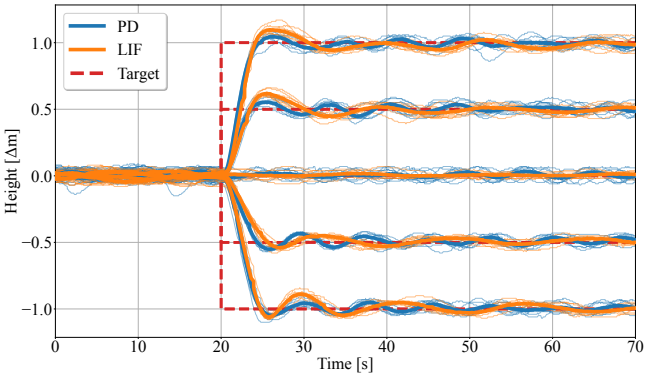


Fig. 8. Real-world step responses of the altitude of the neutrally buoyant blimp using a conventional PD control and an SNN controller with no recurrency or IWTA in the hidden layer. Each setpoint was tested eight times for every controller and the average is shown by the thick line.

is shown in Figure 9, where the setpoint was changed after 70 seconds. When analyzing the steady-state error, we take the average over the last 10 seconds of each step. The steady-state error of the IWTA-LIF (±2 cm) is significantly smaller than the ones for the R-LIF and R-IWTA-LIF (±5 cm). Despite the small oscillations caused by the LIF SNN, we decided to use this spiking integrator for the full spiking controller due to the minimal steady-state error.

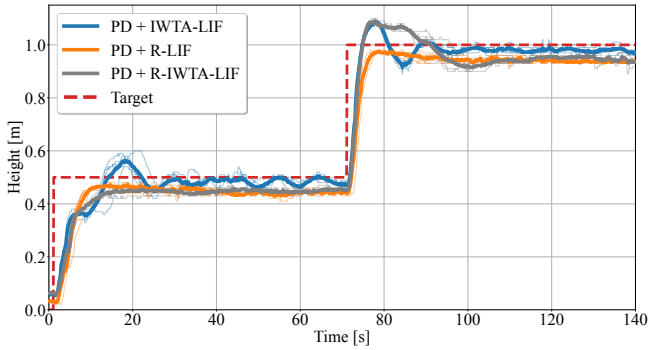


Fig. 9. Real-world step responses of the altitude of the negatively buoyant blimp using a conventional PD controller and different spiking integrators. The average over five runs is shown for each controller.

3) Full spiking control of negatively buoyant blimp: The combination of both the spiking PD (LIF) and integral (IWTA-LIF) controller is shown in Figure 10. We analyzed the step response for the blimp using different setpoints h=[0.5,1,1.5]m, maintained for 70s. The combined SNN controller demonstrates effective altitude control while minimizing the steady-state error to ±3cm. However, the SNN controller shows relatively large initial oscillations when receiving a downward step input, compared to the upward steps. The oscillations result from the delay introduced when the rotors must make a 180-degree turn during direction changes. Given the blimp's negative buoyancy, continuous upward thrust is essential for stability. In cases of upward step inputs, the rotors constantly push the blimp upwards. Conversely, when a lower setpoint is used, the blimp is initially pushed downwards by the rotor before pushing back up to attenuate the movement. This difference in overshoot is also visible for the PID controller, with an average overshoot of 14cm for upward steps and 20cm for downward steps.

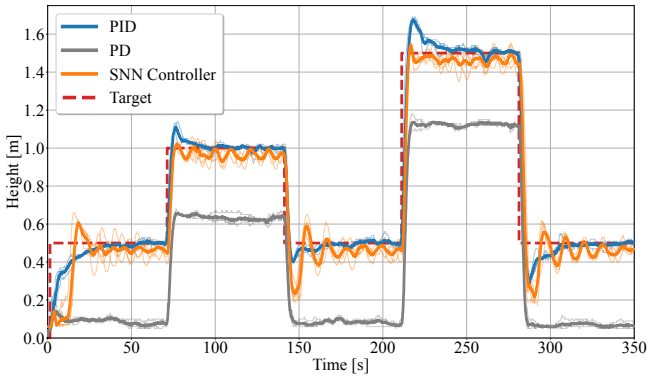


Fig. 10. Real-world multi-step response of the altitude of the negatively buoyant blimp using a non-spiking PID & PD controller and the SNN controller, which is the combination of the SNN PD (LIF) and the SNN integral (IWTA-LIF) controller. The average over five runs is shown for each controller.

IV. Conclusion

In this work, we evolved a spiking neural network (SNN) that successfully controls the altitude of a non-neutrally buoyant indoor blimp. The SNN parameters were optimized through an evolutionary algorithm, facilitating extensive exploration of the solution space. This exploratory training approach allowed for an in-depth analysis of various hidden-layer configurations, recurrency and Input Weighted Threshold Adaptation (IWTA), for the Leaky-Integrate and Fire (LIF) neuron model. As a result, we developed two complementary SNN controllers which, when combined, achieved accurate tracking of the reference state. The first controller exhibited rapid response to control errors while effectively mitigating overshoot and large oscillations, after being trained on a tuned PD controller for the blimp. In parallel, the second controller was designed to minimize steady-state errors arising from the blimp's non-neutral buoyancy-induced drift. This controller learned to perform integration of the error using IWTA within the hidden layer of the network.

Despite the limitation within the blimp's current drivetrain configuration, the developed SNN controllers have showcased their ability to maintain stable altitude control, employing just 160 spiking neurons. All processing and sensing is performed onboard the blimp, with the SNN running on the Raspberry Pi's CPU. Future research will focus on the completion of the neuromorphic control loop, integrating event-based sensors with neuromorphic processors. This integration aims to fully demonstrate the potential of neuromorphic computing in robotic control.

REFERENCES

[1] D. Zhang, A. Loquercio, X. Wu, A. Kumar, J. Malik, and M. W. Mueller, "A Zero-Shot Adaptive Quadcopter Controller," 9 2022. [Online]. Available: http://arxiv.org/abs/2209.09232

[2] M. Heryanto, H. Suprijono, B. Yudho, and B. Kusumoputro, "Attitude and altitude control of a quadcopter using neural network based direct inverse control scheme," *Advanced Science Letters*, vol. 23, pp. 4060–4064, 05 2017.

[3] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[4] Z. Bing, C. Meschede, F. Röhrbein, K. Huang, and A. C. Knoll, "A survey of robotics control based on learning-inspired spiking neural networks," *Frontiers in Neurorobotics*, vol. 12, 2019.

[5] A. Vitale, A. Renner, C. Nauer, D. Scaramuzza, and Y. Sandamirskaya, "Event-driven vision and control for uavs on a neuromorphic chip," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 103–109.

[6] A. Taherkhani, A. Belatreche, Y. Li, G. Cosma, L. P. Maguire, and T. M. McGinnity, "A review of learning in biologically plausible spiking neural networks," *Neural Networks*, vol. 122, pp. 253–272, 2020.

[7] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-based optimization to spiking neural networks," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 51–63, 11 2019.

[8] Z. Bing, C. Meschede, F. Röhrbein, K. Huang, and A. C. Knoll, "A survey of robotics control based on learning-inspired spiking neural networks," *Frontiers in Neurorobotics*, vol. 12, 2019.

[9] R. Stagsted, A. Vitale, A. Renner, and L. B. Larsen, "Event-based PID controller fully realized in neuromorphic hardware: A one DoF study," in *IEEE International Conference on Intelligent Robots and Systems*, vol. 2, 7 2020, pp. 10 939–10 944.

[10] R. Stagsted, A. Vitale, A. Renner, and L. Larsen, "Towards neuromorphic control: A spiking neural network based PID controller for UAV," in *Robotics: Science and Systems XVI*, vol. 1, 1 2020.

[11] S. Stroobants, J. Dupeyroux, and G. De Croon, "Design and implementation of a parsimonious neuromorphic pid for onboard altitude control for mavs using neuromorphic processors," in *Proceedings of the International Conference on Neuromorphic Systems 2022*, 2022, pp. 1–7.

[12] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[13] Y. Zaidel, A. Shalumov, A. Volinski, L. Supic, and E. Ezra Tsur, "Neuromorphic NEF-Based Inverse Kinematics and PID Control," *Frontiers in Neurorobotics*, vol. 15, 2 2021.

[14] S. Stroobants, C. De Wagter, and G. De Croon, "Neuromorphic control using input-weighted threshold adaptation," in *Proceedings of the 2023 International Conference on Neuromorphic Systems*, 2023, pp. 1–8.

[15] M. Gonzalez-Alvarez, J. Dupeyroux, F. Corradi, and G. C. De Croon, "Evolved neuromorphic radar-based altitude controller for an autonomous open-source blimp," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 85–90, 2022.

[16] H. Qiu, M. Garratt, D. Howard, and S. Anavatti, "Towards crossing the reality gap with evolved plastic neurocontrollers," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, 2020, pp. 130–138.

[17] Delft High Performance Computing Centre (DHPC), *DelftBlue Supercomputer (Phase 1)*, 2022, https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1.

[18] N. Hansen, "The cma evolution strategy: a comparing review," *Towards a new evolutionary computation: Advances in the estimation of distribution algorithms*, pp. 75–102.

[19] N. E. Toklu, T. Atkinson, V. Micka, P. Liskowski, and R. K. Srivastava, "EvoTorch: Scalable Evolutionary Computation in Python," pp. 1–25, 2023. [Online]. Available: http://arxiv.org/abs/2302.12600

[20] I. Cohen, Y. Huang, J. Chen, J. Benesty, J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient," *Noise reduction in speech processing*, pp. 1–4, 2009.

Part II

Literature Study

*This part has been assessed for the course AE4020 Literature Study.

3

Conventional MAV Control

As stated in this thesis's research objective, the goal is to develop low-level controllers that are able to perform stable position or angle control on a drone. To successfully develop a controller for any system, it is useful to get some insights into the dynamics of the systems that are to be controlled.

This chapter will provide the reader with knowledge of the fundamental dynamics of a quadrotor, together with some of its inherent challenges that arise when attempting MAV control in Section 3.1. After that, in Section 3.2, a brief overview of the different types of conventional controllers is presented together with an introduction to PID control.

3.1. Quadrotor Dynamics

When designing a controller, it is essential to have some insights into the dynamics of the system or process that needs to be controlled. For this thesis, the controlled system will be a drone with four rotors, i.e., a quadrotor. The four rotors of the drone are the actuators of the system, each exerting a force (F) that coincides with the rotor's rotation axis and a torque (τ) in the opposite spinning direction of the rotor. A free body diagram of the drone is provided in Figure 3.1. Both opposing rotors (1,3 & 2,4) are spinning in the same direction since this enables decoupled yaw controllability due to the canceling of the torques.

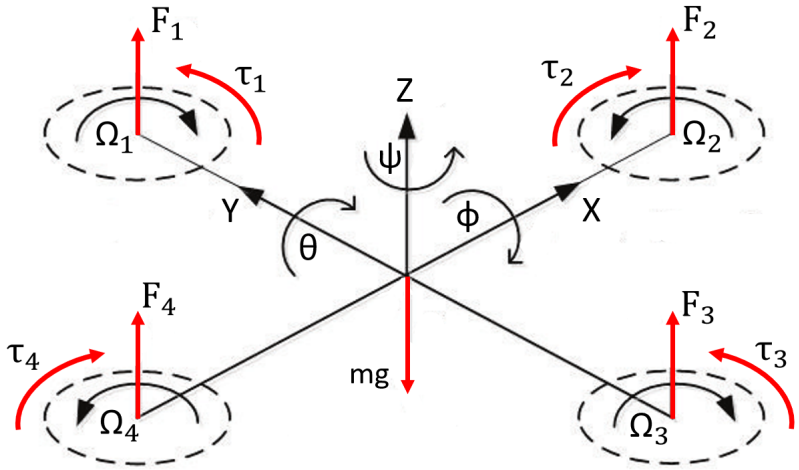


Figure 3.1: Free body diagram of a quadrotor. The forces and torques acting on the drone are shown in red. Adjusted from Islam et al. (2017)

Some characteristics describe the challenges that arise when trying to control the highly nonlinear dynamics of a drone. First, the quadrotor is an underactuated system since the drone has 6 DoF (translation and rotation) but only four independent control inputs. This implies that it is impossible to fully control all 6

DoF simultaneously. Secondly, drones are inherently unstable systems (Zulu et al. 2014). A quadrotor will deviate from its hover condition unless stabilizing control input is used. So, it is crucial that when a drone is in flight, it always receives a control input to prevent it from crashing.

Other challenges of designing a drone controller are the parametric and the nonparametric uncertainties that influence the quadrotor dynamics (Emran et al. 2018). Parametric uncertainties are uncertainties in the system parameters, such as deviations in the weight of the drone. Other uncertainties, such as external disturbances (e.g. wind gusts), are labeled as nonparametric uncertainties. For a practical controller of a drone, it must be designed to cope with the (non)parametric uncertainties while still showing accurate reference tracking behavior.

When creating a controller for a real drone, it is common to first design and test it in simulation. When a controller designed in simulation is used on an actual drone, it has to bridge the reality gap to control the drone successfully. A more realistic and often complex simulation model results in a smaller reality gap. So it is crucial to take into account the complexity of the modeled quadrotor dynamics when designing a controller. No decision has yet been made on the dynamic model that will be used to simulate the drone in this thesis.

3.2. Basic PID Control

Different types of controllers can overcome the challenges of controlling the highly nonlinear dynamics of drones, as described in the previous section. There are three different types of controllers: linear, (model-based) nonlinear and learning-based (Mooney et al. 2014). This research will focus on using spiking neural networks (learning-based) in combination with PID control (linear). As stated in the research objective, a spiking neural network will be used to mimic a PID controller. This linear controller is chosen, instead of a different linear (e.g. Linear Quadratic Regulator) or nonlinear controller, due to its simplicity and efficiency (Åström et al. 1995). Model-based nonlinear controllers (e.g. Model Predictive Control and Backstepping) and other linear controllers are not relevant to this MSc thesis and will thus not be further discussed in this section. An extensive overview of different types of controllers used for drone control can be found in Zulu et al. (2014). This section will continue with the basics of PID control.

One of the most used types of controllers used is the Proportional, Integral and Derivative controller, known as the PID controller. The closed-loop control feedback is used to maintain the actual output of a system as close to a set target as possible. This simple yet efficient controller can effectively control a system's transient and steady-state response. Due to the simplicity, applicability, ease of use and straightforward functionality of the PID controller, it is still the industry standard, especially for the lowest-level controllers (Okasha et al. 2022). The mathematical notation of the PID controller is shown in Equation 3.1.

$$u(t) = K_{\mathsf{p}} e(t) + K_{\mathsf{i}} \int_0^t e(\tau)\mathsf{d}\tau + K_{\mathsf{d}} \frac{\mathsf{d}e(t)}{\mathsf{d}t} \tag{3.1}$$

In which $e(t)$ is the error signal, defined as the difference between the reference signal of the controlled state x_{ref} and the actual state $\hat{x}$, i.e. $e(t) = x_{ref}(t) - \hat{x}(t)$. The control signal is denoted by $u(t)$, and the proportional, integral and derivative gains are represented by K_p, K_I and K_D, respectively. The three gains assigned to each component of the PID controller are used to tune the system's response based on predefined requirements for that system. Examples of types of requirements are stability, robustness and set-point tracking performances such as rise-time, overshoot and settling time (Ang et al. 2005). The effects of the control gains on the set-point tracking performance are summarized in Table 3.1.

Closed-Loop Response	Rise Time	Overshoot	Settling Time	SteadyState Error	Stability
Increasing K_P	Decrease	Increase	Small Increase	Decrease	Degrade
Increasing K_I	Small Decrease	Increase	Increase	Large Decrease	Degrade
Increasing K_D	Small Decrease	Decrease	Decrease	Minor Change	Improve

Table 3.1: Effects of independent P, I and D Tuning (Ang et al. 2005)

4

Spiking Neural Networks

Spiking neural networks (SNN) are considered to be the third generation of neural networks (Maass 1997). In the first generation of ANN, the neuron was modeled as a perceptron. Later on, the non-linear activation function was added to the model of the neuron, which led to the origin of the second generation of ANNs. The SNN opens the door for asynchronous, event-based signal processing by transferring information in the form of spikes, which closely mimics the behavior of biological neurons found in animals' brains. One of the main advantages of the SNN over the previous generations of ANNs is the inherently low energy usage caused by the more sparse and asynchronous communication style of using spikes (Wu et al. (2022), Yin et al. (2020), Panda et al. (2020), Merolla et al. (2014)). The SNNs employ an event-based technique of communication between neurons instead of the synchronous method used by non-spiking ANNs, which involves transferring signals between neurons at each processing timestep.

This chapter will discuss the most relevant properties that help to understand the principles and characteristics of Spiking Neural Networks. As mentioned earlier in the introduction of this chapter, the SNNs are based on biological neurons. Therefore, a brief overview of the biological neuron will be presented in Section 4.1 to provide some background information. In Section 4.2, the most relevant artificial models for spiking neurons are characterized. Section 4.3 will analyze the various available techniques of encoding and decoding information into a spike pattern, discussing the benefits and drawbacks of each approach. Not all machine learning algorithms developed for ANNs can be directly applied to Spiking Neural Networks, which is why Section 4.4 will cover the relevant learning methods applicable for SNNs. Finally, a brief review of the available neuromorphic processors will be provided in Section 4.5, with a focus on relevant features for this MSc thesis

4.1. Biological Neuron

To understand the layout of the artificial neural models described in Section 4.2, it is useful to get a brief overview of the working principles of the biological neurons.

In Figure 4.1, the most relevant parts of the biological neuron and its connection to neighboring neurons are highlighted. One single biological neuron consists of 3 different parts, which are the *Dendrite*, *Soma* and *Axon*. The Dendrite and the Axon are, respectively, the parts of the neuron where the input spikes are received, and the resulting output spikes are sent to. The inner part of the neuron is called the Soma, whose main function is to track the neuron's membrane potential and to send a spike, also called an *action potential*, if the membrane potential exceeds a specific limit. The part where the Axon of a neuron (presynaptic neuron) is connected to the Dendrite of a successive neuron (the postsynaptic neuron) is called the synapse. At the synapse, the action potential of the presynaptic neuron is transferred to the postsynaptic neuron (Gerstner et al. 2014).

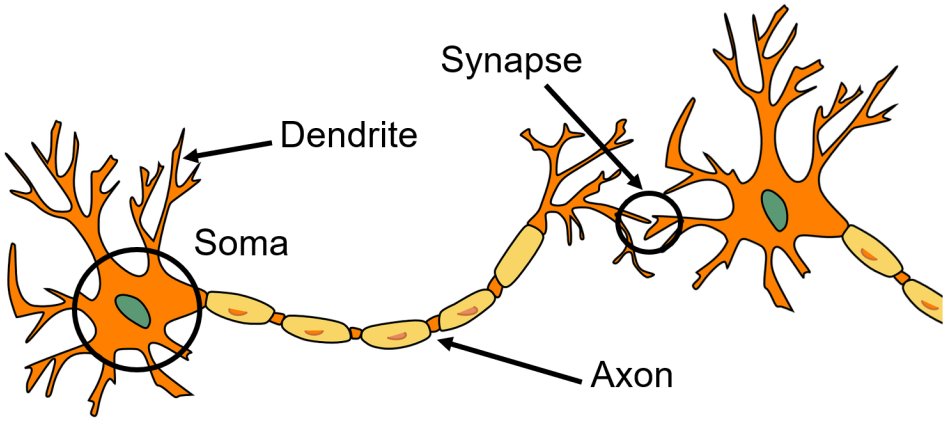


Figure 4.1: The schematics of biological neurons. Image adapted from Pixabay[1]

4.2. Artificial Spiking Neural Models

Different mathematical models can be used to describe the dynamics of a single neuron in an SNN. Some models are designed to approximate the dynamics of biological neurons as closely as possible, resulting in a complex model that requires a large amount of computational power. The Hodgkin-Huxley can be seen as such a complex biologically plausible model (Hodgkin et al. 1952). On the other hand, there are more simple neural models, such as the *LIF*, the Leaky-Integrate and Fire (Stein 1965) and the Izhekevic model (Izhikevich 2003). The simplicity of these models is the reason that they are computationally efficient to use. However, one of the most efficient neuron models, the LIF, lacks biological plausibility. An overview of the most used neural models with their corresponding computational cost and biological plausibility is presented in Figure 4.2

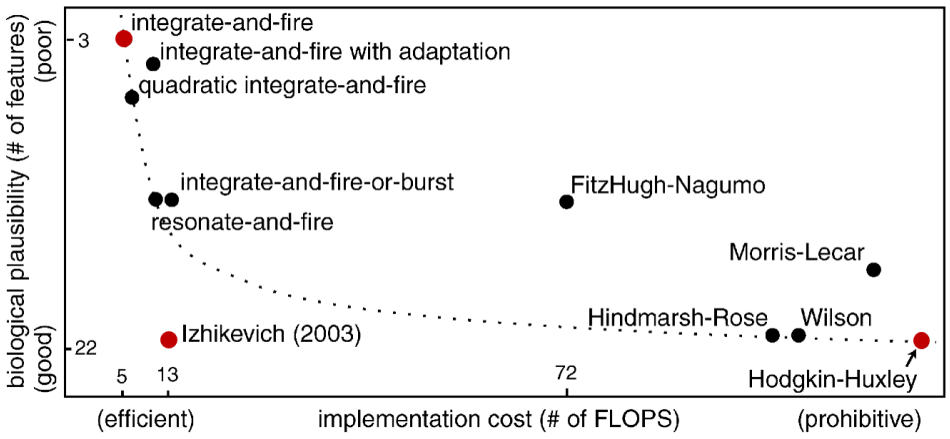


Figure 4.2: Trade-off between the biological plausibility and the complexity of the most used neural models (Izhikevich 2004). The different neural models that have been discussed in-text are highlighted in red.

For this MSc thesis, the main focus will be on the usage of computationally efficient neural models, like LIF (or close relatives such as the Adaptive LIF model) and the Izhikevich model, since power efficiency is a more critical requirement for drone controllers than biological plausibility. An extensive review of

[1] https://pixabay.com/vectors/neuron-nerve-cell-axon-dendrite-296581/

all different types of spiking neural models that are used can be found in Izhikevich (2004) and Bing, Meschede, et al. (2019).

4.2.1. Leaky Integrate and Fire Model

One of the most widely used artificial spiking neural models is the leaky Integrate and Fire model, first introduced by Stein (1965). This model only uses the spike timing to carry the neural information instead of the specific shape of the spikes. The formula for a sequence of spikes, also known as spike train, is shown in Equation 4.1

$$S(t) = \sum_f \delta(t - t^f) \tag{4.1}$$

In the equation above, $\delta()$ is the Dirac function, and t_f is the time at which the neuron emits a spike. The dynamics of the LIF model is described by three components: the membrane potential u, the postsynaptic potential/current I and the firing threshold ϑ. The three different components are covered in the following paragraphs.

Membrane Potential The membrane potential (u) is the only state of a LIF neuron. The differential equation for the membrane potential of the neurons is shown in Equation 4.2, and it can be separated into two parts: leak voltage and input voltage. These two elements show that $u(t)$ acts as a leaky integrator of $I(t)$, the postsynaptic current.

$$\dot{u}(t) = -\frac{1}{\tau_{mem}} \left[u(t) - u_{\text{rest}}\right] + \frac{R}{\tau_{mem}} I(t) \tag{4.2}$$

where τ_{mem} is the membrane time constant and R is the input resistance.

Firing Threshold It is important to note that Equation 4.2 only describes the subthreshold dynamics of the membrane potential. When the membrane potential of a neuron exceeds the threshold, ϑ, the neuron spikes and the membrane potential resets to the resting potential u_{rest}. The time at which the spike is emitted is denoted by $t^f \leq t$ with f = 1,2,3 .... After the spike is emitted, the neuron can be kept at its resting potential for a predetermined amount of time, called the *refractory time*, t_{ref}. The mathematical notation of the threshold dynamics is shown in Equation 4.3. In the LIF model, the threshold is a constant, in contrast to the ALIF (covered in Section 4.2.2) where the threshold is variable over time.

$$\begin{gathered} t^f : u(t^f) = \vartheta \\ t^f \leq t \leq t^f + t_{ref} \Rightarrow u(t) = u_{rest} \end{gathered} \tag{4.3}$$

Postsynaptic Current The postsynaptic current ($I(t)$) is described using Equation 4.4, where $W_{i,j}$ is the weight between the pre- and postsynaptic neuron, $\epsilon(t - t_i^f)$ is the shape of the presynaptic spikes, and the subscripts i & j denote the pre-and postsynaptic neuron respectively.

$$I(t) = \sum_i W_{i,j} \sum_f \epsilon(t - t_i^f) \tag{4.4}$$

For most LIF models, the presynaptic shape of the spikes is simplified to a Dirac pulse, $\epsilon(s) = \delta(s)$, with $s = t - t_i^f$ (Tang 2022). However, a more realistic shape of a presynaptic spike is an exponentially decaying function, such that the presynaptic spikes' impact on the postsynaptic potential has a limited duration. In this case, the postsynaptic current acts as a leaky integrator on the presynaptic spike train. This behaviour can be realized with eg. $\epsilon(s) = \frac{1}{\tau_{syn}} exp(-s/\tau_{syn})$, where τ_{syn} is the synaptic time constant. The effect of using different presynaptic spike shapes on the PSP is shown in Figure 4.3. The varying height of the peaks is the result of different synaptic weights.

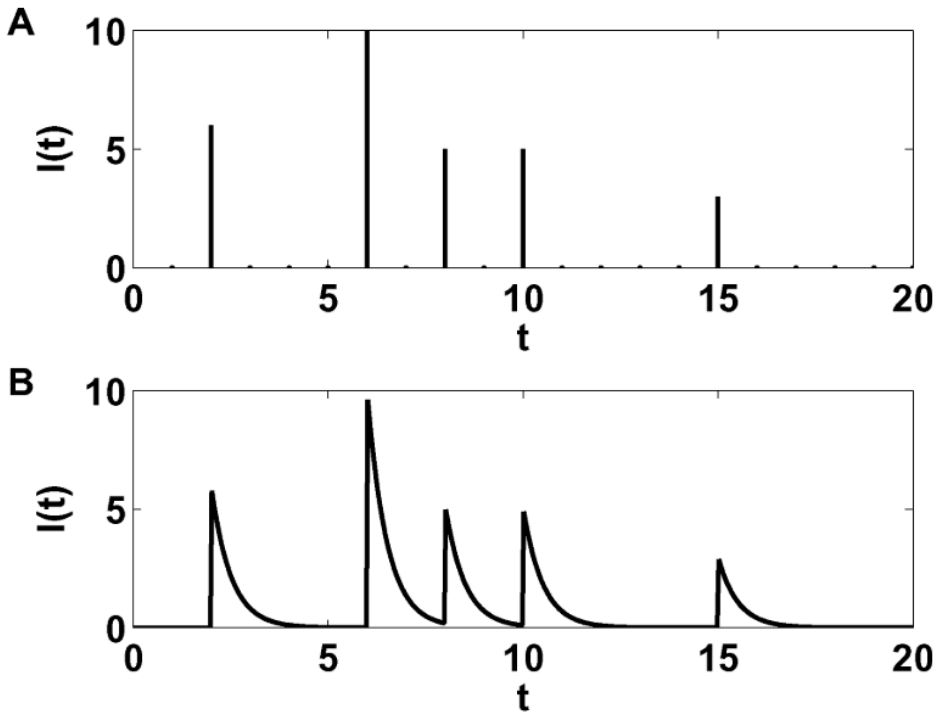


Figure 4.3: Time traces of the PSP for different types of presynaptic spike shapes, with ϵ equals A) a Dirac pulse and B) a decaying exponential function with τ_{syn} = 0.5. The firing time t^f is shown for f = 2,6,8,10 & 15 (Shen et al. 2008)

4.2.2. Adaptive Leaky-Integrate and Fire Model

The Adaptive Leaky-Integrate and Fire (ALIF) neuron model is very similar to the LIF neuron model. Nonetheless, when a constant current is injected into the LIF neuron model, it results in a constant output frequency of spikes. More realistic neuron models, such as the ALIF and Izhikevich model, have implemented a way that allows for *frequency adaptation* to happen, which means that the neuron habituates to its input current. There are two different implementations for the ALIF, which will be discussed in the following two paragraphs.

Adaptive Threshold A way to interpret this frequency adaptation is to regard the threshold as a variable leaky integrator, which is increased when an output spike is emitted (Bellec et al. 2018). So when no spikes occur for a while, the threshold will be reset to its previously set value. This results in the following differential equation for the threshold dynamics, where ϑ_0 is the resting threshold, β is the constant which increases the threshold with each output spike ($S_j(t)$) that is emitted

$$\dot{\vartheta}(t) = -(\vartheta(t) - \vartheta_0) + \beta S_j(t) \tag{4.5}$$

Adaptive Current Although the method described above is a more intuitive way of implementing the effect of frequency adaptation for ALIF, the most used mathematical models for ALIF use an adaptation current w_k instead of a variable threshold (Gerstner et al. 2014). This adaptation current is included in the dynamics of the membrane potential of the neuron, which leads to an additional term in Equation 4.2.

$$\dot{u}(t) = -\frac{1}{\tau_{mem}}\left[u(t) - u_{\text{rest}}\right] + \frac{R}{\tau_{mem}}\left[I(t) - w_k(t)\right] \tag{4.6}$$

The differential equation of the adaptation current is shown below, where a_k and b_k are design parameters, and τ_k is the time constant of the adaptation current.

$$\dot{w_k}(t) = -\frac{a_k}{\tau_k}\left[u(t) - u_{\text{rest}}\right] - \frac{1}{\tau_k} w_k(t) + b_k S_j(t) \tag{4.7}$$

So instead of increasing the threshold, this implementation of the ALIF temporally decreases the total current ($I(t) - w_k(t)$) when the neuron emits a spike, which causes the frequency adaptation to happen.

4.2.3. Izhikevich Model

The Izhikevich neuron model has a biological plausibility that is comparable to the Hodgkin-Huxley model but a computational complexity comparable to the LIF neuron model, as can be seen in Figure 4.2. In contrast to the LIF models that only have one state variable, this neural model has two state variables, the membrane potential (v) and the recovery variable (u). The subthreshold dynamics of this neuron model is fully described by the two ordinary differential equations (ODEs), as shown in Equation 4.8 (Izhikevich 2003), in which the postsynaptic current is described by I. Due to the addition of the recovery variable, spike frequency adaptation, as discussed in the introduction of Section 4.2.2, also occurs in this neural model. The threshold dynamics of the Izhikevich model is shown in Equation 4.9, which shows the reset of the membrane potential and recovery variable when the predefined threshold of +30 mV is reached.

$$\begin{aligned} \frac{dv}{dt} &= 0.04v^2 + 5v + 140 - u + I \\ \frac{du}{dt} &= a(bv - u) \end{aligned} \tag{4.8}$$

$$\text{if } v \geq 30mv, \text{ then } \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \tag{4.9}$$

There are four parameters, a, b, c and d, present in Equation 4.8 and 4.9 which can be used to tune the response of the neuron. The time scale of the recovery variable u is described by the parameter a. A slower recovery is the result of smaller values of a. The recovery variable's sensitivity to subthreshold fluctuations in the membrane potential is described by the parameter b. Greater values of b result in a stronger coupling between v and u which causes low-threshold spiking dynamics and possible subthreshold oscillations. The parameter c characterizes the reset value of the membrane potential when a spike is emitted. The after-spike reset of the recovery variable is described by parameter d. A visual representation of the influence of the parameters of the model on the spiking behavior is shown in Figure 4.4.

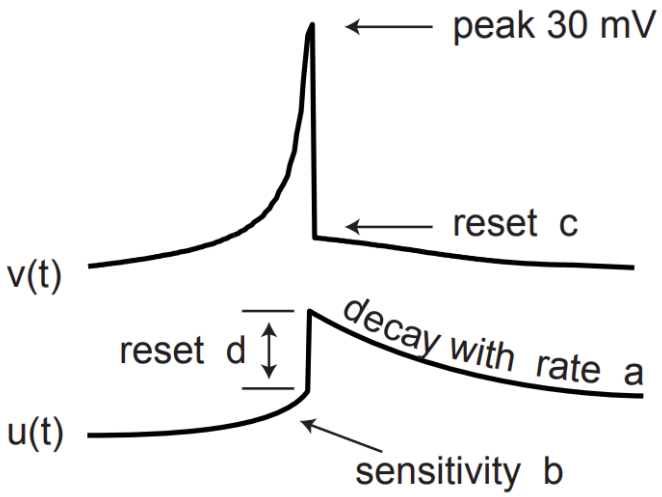


Figure 4.4: Visual representation of the effect of the parameter of the Izhikevich neuron model on membrane potential and the recovery variable (Ning et al. 2012)

4.3. Encoding & Decoding Methods

In a conventional synchronous controller, the in- and outputs are coded as floating points. This is because most of the current hardware and sensors use floating points. However, the goal of this MSc thesis is to implement spiking neural networks as controllers. Since an SNN uses spikes to communicate with each other, it is necessary to find a way to convert floating points to a sequence of spikes. In this section, different ways to encode and decode information into spikes will be discussed. There are multiple design choices available when selecting a well-suited coding method for a certain application:

- Using one neuron vs multiple neurons
- Using one timestep vs multiple timesteps
- One neuron can spike only once vs multiple times

These are three variables that can be tuned when encoding information: number of neurons, timesteps and number of spikes. Using these three distinctions, most of the coding methods can be assigned to a combination of these three attributes. The three most used types of coding schemes will be shortly discussed in this work: position, temporal and rate-based coding (Dupeyroux et al. 2022).

4.3.1. Position Coding

Position coding can be seen as the most basic type of coding scheme. For this scheme there are multiple neurons required, at least two, per feature that needs to be coded, for this explanation, only one feature will be used. The input space of the feature is divided over the number of neurons that are present. This implies that each neuron in the coding layer has a predefined exclusive range of input values which will cause the neuron to spike. So in the positional-coding layer, there will be only one neuron that spikes in that group of neurons, which corresponds to the predefined range of values.

4.3.2. Temporal Coding

Where positional coding only uses one timestep to encode or decode data, temporal coding uses multiple timesteps to code a single value. When using this coding scheme, the exact spike timing is a variable that is used to code data. One of the most used temporal coding schemes is the Time-to-first spike coding scheme (Johansson et al. 2004). It basically means that a higher stimulus, which most often corresponds with larger input values, causes the neuron to fire early. So using the spike time as a variable, different input values can be encoded. This scheme can be used with a single neuron or with multiple neurons. When multiple neurons are used with this temporal scheme, a combination of temporal and positional coding applies, such as the Gaussian Receptive Field model (Dupeyroux et al. 2022).

4.3.3. Rate-Based Coding

The last type of coding scheme, Rate-Based coding, can be seen as a type of temporal coding scheme in the sense that time is used as a variable to encode/decode data. However, temporal coding uses often only one spike per neuron and then the exact spike timing is used as a variable to encode the data. Whereas, Rate-Based coding creates multiple spikes per neuron and uses the inter-spike interval, ISI, to encode and decode data. The main firing rate of a neuron codes the size of the coded data, e.g. a larger spiking frequency corresponds to a large input value. The main firing rate can be decoded using a sliding integration window (Webb et al. 2011).

4.3.4. Encoding & Decoding Performance Analysis

When comparing the three types of coding schemes, there are some interesting performance aspects that are inherent to that type of coding method. The positional coding scheme adds the least amount of delay to the system, since it does not use time as a variable for the coding and thus the delay is limited to a single timestep. Whereas, rate-based decoding requires integration over a certain window of the spike train. The size of this integration window directly affects the amount of delay. The temporal coding schemes have a relatively low delay compared to most rate-based coding schemes (Eliasmith 2003) However, temporal coding schemes (such as TTFS) are not well-suited when dealing with dynamic inputs (Bonilla et al. 2022), making it an unqualified coding scheme for the SNN controllers that have to be designed for this MSc thesis .

When comparing the resolution of the coding scheme, which can be seen as the ability to encode or decode a distinct number of values, there are also some inherent differences between the different coding

schemes. When analyzing the resolution of a scheme, the number of neurons that are used to code the value is constant for all methods. This means that the positional coding scheme always leads to a lower resolution compared to the temporal and rate-based schemes due to the lack of *time* as a variable. The resolution of temporal and rate-based schemes depends on the exact type of algorithm used and the selected time window for both methods.

An analysis of the performance of different types of combinations of encoding and decoding methods has been performed in Schuman, Rizzo, et al. (2022), specifically focused on control tasks[2]. There are two relevant key observations for this research that the authors of this review paper conclude:

- Rate decoding requires encoding approaches that tend to produce more spikes across fewer input neurons
- Encoding approaches that generate more spikes tend to outperform those that generate fewer spikes

These observations are relevant for this MSc thesis since there have been prior studies that have implemented an SNN PID controller using either a positional (Stagsted et al. 2020b, Stroobants, Dupeyroux, et al. (2022)) or a rate coding scheme (Zaidel et al. (2021)). Since rate encoding generates more output spikes compared to positional encoding, following both observations, it should show that the rate-coded SNN PID outperforms the positional-coded SNN PID implementation. An analysis of both implemented SNN PID controllers is provided in Section 5.1.

4.4. Machine Learning Algorithms for SNNs

Similarly to ANNs, Spiking Neural Networks can be trained by adjusting the weights of the network. Biologically speaking, the weight of a neuron corresponds to the strength of the connection between different neurons, which is called *synaptic efficacy*. The ability to adjust the synaptic efficacy is known as *synaptic plasticity*, which enables the learning ability of a biological network of neurons (Lines et al. 2017).

There are multiple types of methods available to adjust the synaptic efficacy of an SNN: Unsupervised, Supervised, Reinforcement and Evolutionary Algorithms. As stated in the research objective in Chapter 1, a part of the research for this MSc thesis will focus on training an SNN to control a drone. This chapter provides a basic level of understanding of the most relevant learning methods that are available for the training of an SNN. A full in-depth analysis of prior research on the applications of the relevant learning method, focusing on control tasks, will be provided in Chapter 5.

4.4.1. Unsupervised Learning

Unsupervised learning methods are nowadays most often used to train ANNs in pattern recognition tasks. When using unsupervised learning, it is not able to correct a neural network while training when it is not learning as desired. One of the most used unsupervised learning rule for SNNs is STDP, Spike-Timing dependent plasticity. This biologically plausible learning method is based on a Hebbian learning rule, which is best described as: ”those who fire together, wire together” (Hebb 1949). Mathematically, the Hebb learning rule is expressed as

$$\Delta w_{i,j} \propto v_i v_j, \tag{4.10}$$

where the pre- and postsynaptic neurons are referred to as v_i and v_j respectively. So, when using an STDP learning rule, it means that the synaptic weight between two neurons is increased (Long-Term Potentiation) when first a presynaptic neuron spikes and shortly after the postsynaptic neuron also fires. When it is the other way around, the postsynaptic neuron fires before the presynaptic neuron, which results in a decrease in the synaptic weights (Long-Term Depression) between the neurons. An overview of the weight change as a function of spike timing of the pre-and postsynaptic neuron is shown in Figure 4.5. In which A_+ and A_- are constants and τ_+ and τ_- are the time constants, controlling the decay of the change in synaptic efficacy.

[2]Two different OpenAI Gym environments are used with a continuous action space

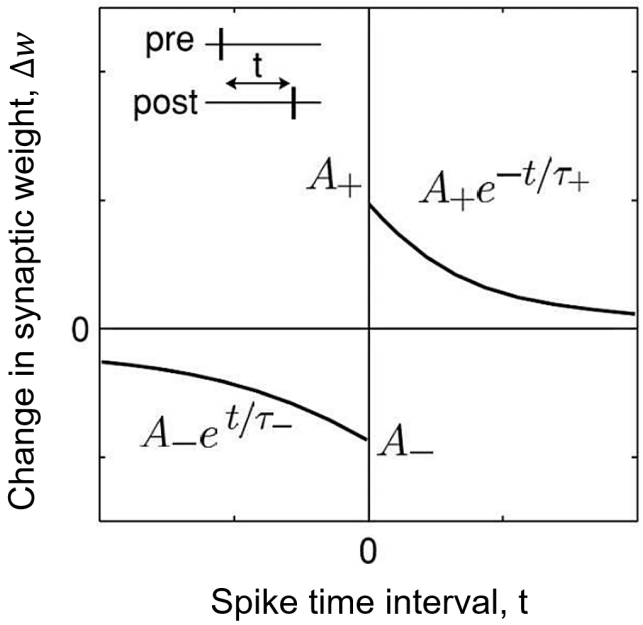


Figure 4.5: The STDP learning rule. Describing the weight change of a synapse as a function of the exact spike timing of the pre-and postsynaptic neuron. Adjusted from Taherkhani et al. (2020)

4.4.2. Supervised Learning

Supervised learning methods require a labeled dataset or external signal containing the desired output of the neural network. In the case that an SNN needs to behave similarly to a PID controller, the control output of a PID controller can then ideally be used as a training signal. Based on the difference (error) between the desired output (the training signal) and the actual output of the SNN, the weights of the network can be adjusted to minimize the control error.

There are different approaches for changing the weights of an SNN. One of the most used methods for updating the weight of non-spiking neural networks is *error backpropagation* in combination with a gradient descent algorithm. However, there is an intrinsic problem when calculating the gradient of the membrane potential w.r.t. the weights of the network caused by the discontinuous signal of the membrane potential triggered by the threshold dynamics of the neuron model. In recent years, there have been several approaches that found a way to deal with this problem, such that the principle of error backpropagation can also be applied to spiking neural networks.

The first method that was developed to allow the backpropagation for SNN is called *SpikeProp* (Bohte et al. 2002). To deal with the non-differentiable issue, the threshold dynamics for the membrane potentials of post-synaptic neurons were assumed to be piecewise linear and thus differentiable. This method uses the exact spike timing in its loss function in combination with a neuron model that allows each neuron to only fire once to encode data temporally. This poses a limitation on the encoding that can be used, which excludes the use of rate encoding.

SpikeProp is an example of a method in which the non-differentiable problem is solved by adapting the spiking neuron model dynamics to ensure well-behaved gradients. However, there is also a different method that does not directly adjusts the dynamics of the neuron but instead only uses a surrogate gradient (SG) function to replace the non-differentiable functions during backpropagation. This means that SG is not used during the forward pass of the neural network (Bohte 2011). This surrogate gradient is a continuously differentiable function. An overview of the different types of SG that are used can be found in Neftci et al. (2019).

There exist many more different methods that enable supervised learning methods for SNN than were covered in this subsection. For an extensive overview of all types of methods, refer to Wang et al. (2022) or Tavanaei et al. (2019)

4.4.3. Reinforcement Learning

Classical reinforcement learning (RL) is a type of machine learning that involves the training of an agent to make decisions in an environment while trying to maximize the total amount of received reward. There exist some examples for which an SNN is trained on a simple control task using some of the classical reinforcement algorithms that are also used for the training of ANN, such as temporal difference or model-based learning rules (Bing, Meschede, et al. 2019). However, there is one type of RL algorithm that is specifically designed for training SNNs and which is often used for the training of control tasks (Spüler et al. (2015), Bing, Meschede, et al. (2019), Ramezanlou et al. (2020), Lu et al. (2021), Pérez Fernández et al. (2021) & Juarez-Lora et al. (2022)). This learning method is called *Reward-modulated Spike-Timing Dependent Plasticity* (R-STDP) (Izhikevich 2007).

The R-STDP learning rule is a combination of the STDP rule (described in Section 4.4.1) and the classical RL method. Similarly to classical RL algorithms, there exists an external reward signal that is designed to be positive when desired behavior occurs and possibly negative to punish unwanted behavior. A biological equivalent of a reward signal in a mammal brain is *dopamine*, a type of neurotransmitter, which is why the reward signal for R-STDP is often denoted as $d(t)$. The internal state of the synapse, when using the R-STDP learning rule, is described by 2 variables: w, the weight of the synapse and c, the eligibility trace of the synapse. The equations describing the change in synaptic weight and the dynamics of the eligibility trace is shown in Equation 4.11 & 4.12 respectively.

$$\Delta w = c(t) \cdot d(t) \tag{4.11}$$

$$\dot{c}(t) = -c(t)/\tau_c + STDP(\Delta t)\delta(t - t_{pre/post}) \tag{4.12}$$

The eligibility trace is a way to deal with the credit assignment problem. This implies that the eligibility trace allows us to look into the past at the moment a reward is received, to pinpoint which behavior has led to the received reward. Based on whether the reward is positive or negative, LTP or LTD is applied to the synapse respectively. A visualization of R-STDP for a single synapse is shown in Figure 4.6.

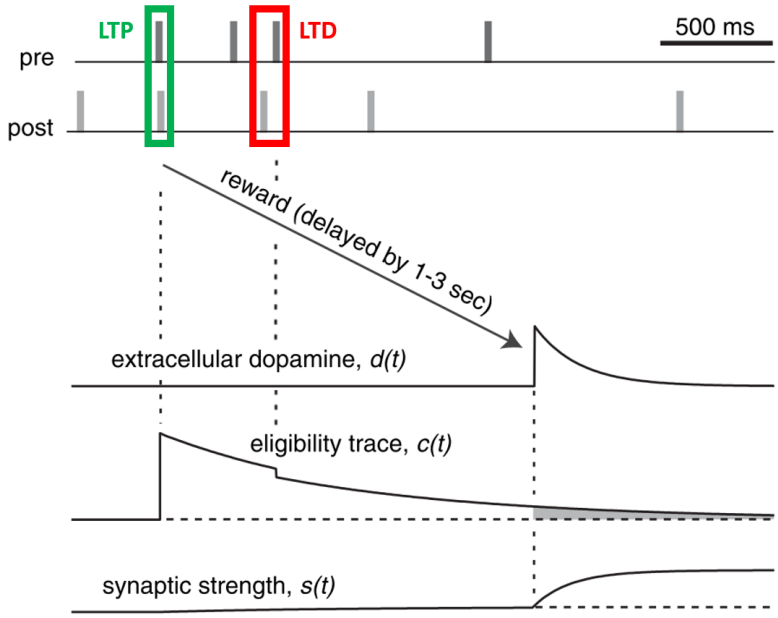


Figure 4.6: Visualisation of the R-STDP learning rule. Adjusted from Izhikevich (2007).

It is important to note that there are different types of reward (dopamine) functions available, with a varying level of sparsity (Bing, Meschede, et al. 2019). Sparse reward functions (eg. where only certain events trigger a reward/punishment) encourage exploration of the solution space and can lead to more generalized solutions since they have to find an optimal solution with less information on what is preferred or not. However, a sparse reward function can lead to slow convergence and suffers more from the credit assignment problem. Increasing the density of the reward function can decrease these negative effects, but it can potentially get stuck more easily in a local optimum since the denser reward function can force the solution more to a certain part of the solution space (Clawson, Ferrari, et al. 2016). Overall, it is important to note that the design and selection of the reward function plays a very important role in the performance of the learning algorithm. The same argumentation holds for the effect of choosing the fitness function for EA learning performance (further discussed in Section 4.4.4) as for the effect of the reward function on learning performance for RL.

4.4.4. Evolutionary Algorithms

Evolutionary algorithms (EAs) are a type of learning algorithm that can be used to optimize the structure and/or parameters of ANNs as well as for SNNs. These algorithms are inspired by the process of natural selection and use concepts such as crossover and mutation to explore the solution space and evolve the network's structure and parameters. A large number of slightly different behaving agents are created and based on a predetermined fitness function, the best scoring agents can be used to create the next generation. This process continues until it reaches a set limit for the number of generations or it reaches a set value for the fitness function.

There are many different types of EAs, which all behave similarly as described above. One type of evolutionary algorithm for (spiking) neural networks is the NeuroEvolution of Augmenting Topologies (NEAT) algorithm. This algorithm and a variation on it called MONEAT, is used to successfully develop an SNN controller for a free-flying drone in simulation by Qiu et al. (2020b). NEAT is a method that uses a genetic algorithm to evolve both the *weights* as well as the *topology* of the network simultaneously. It starts with a simple network structure with some random connections and gradually increases its complexity over time. While using the probabilistic process for mutation and crossover, the EA can adjust network weights and add or remove connections, neurons or layers.

A key feature of the NEAT algorithm is that it uses an identification system to isolate similarly structured individuals in one group and use the EA on each group separately (Qiu et al. 2019). This guarantees a certain level of diversity in the structures of the best-performing individuals and thus exploring more regions of the solution space which increases the chance of finding a global optimal solution. The NEAT algorithms, or different types of structure-adapting EA, are particularly useful for problems where the optimal network structure (eg. feedforward or recurrent) is not known beforehand. A drawback of using EA, in general, is the large computational cost it requires to converge to an optimal solution (Howard et al. 2014). However, for the application in this MSc thesis, it is not important since all training is performed offline so the computational cost is not a limiting variable.

There exist a variety of different types of EA other than the NEAT that is described above. Refer to Gavrilov et al. (2016) and Bing, Meschede, et al. (2019) for an overview of the other types of EA compatible with SNNs.

4.5. Neuromorphic Processors

To make use of the full potential of the SNN, it is necessary to use neuromorphic hardware in stead of the conventional processors that make use of the von Neumann Architecture. In a von Neumann Architecture there is one Central Processing Unit (CPU) and a separate external memory module. Such an architecture experiences the well-known von Neumann bottleneck, in which the speed of data transfer between the CPU and external memory limits system performance (Bouvier et al. 2019). This bottleneck makes the von Neumann Architecture unsuitable for neuromorphic computing, because high parallelism is required to optimally support SNN. Ideal neuromorphic hardware would use a more parallelized architecture where each neuron has its own local memory and processing unit. However, since this approach is not scalable

for larger networks (Shrestha et al. 2022), most neuromorphic processors group a number of neurons together in one core. This implies that each neuron still has its own local data, but they share the same data path to other cores to reduce the amount of connections between each separate neuron. It is possible to fully utilize the SNN's low energy usage with this strategy for neuromorphic computation. A more in depth analysis on the neuromorphic architecture is provided in Shrestha et al. (2022).

There are several neuromorphic processors available and multiple review/survey papers have been written which compared the most used ones (Shrestha et al. (2022), Bouvier et al. (2019),Schuman, Potok, et al. (2017) and Gallego et al. (2022)). The three most referenced neuromorphic processors will be briefly covered. An overview of the relevant characteristics of each processor is shown in Table 4.1. It is important to note that all three neuromorphic processors are digitally implemented. Although there are some neuromorphic analog processors on the market (Moradi et al. (2018), Neckar et al. (2019)), the number of neurons that can be used is still limited due to scalability issues that arise when using an analog implementation (Joubert et al. 2012).

The SpiNNaker processor (Furber et al. 2014) has a large configurational flexibility, since multiple different neural models can be used and it allows on-chip learning. One of the drawbacks of this chip is the high energy usage per synaptic operation (SOP). On the other hand, the TrueNorth chip (Merolla et al. 2014) is extremely efficient but is limited to the usage of the LIF neural model and does not support on-chip learning. The more recently developed Loihi chip from Intel (Davies et al. 2018) is a balanced combination with the availability of On-chip learning, similar to the SpiNNaker and the low energy usage of the TrueNorth. The Energy per synaptic operation of the Loihi is even slightly less than the TrueNorth (Shrestha et al. 2022).

For this MSc thesis, the implementations of the SNN PID and Learned SNN will not be influenced by the neuromorphic processor selection. The reason for this is that the SNN controllers will first be designed and tested using conventional digital hardware (e.g. CPU/GPU) to create a proof of concept before being restricted by the additional limits that the neuromorphic pose on the controller. However, if the SNN controllers will be used on a physical drone, following the reasoning in the previous paragraph, the best option would be the Loihi processor due to the high power efficiency, large number of available neurons and the ability to use on-chip learning.

Neuromorphic processors	**SpiNNaker**	**TrueNorth**	**Loihi**
Supplier	U. Manchester	IBM	Intel
Implementation	Digital	Digital	Digital
Neurons/Chip	16k	1024k	131k
Synapses/Chip	16M	268M	2M
Enegry/SOP (pJ)	10k	26	23.6
Neural Model	Configurable	LIF	LIF
On-Chip Learning	Yes	No	Yes
Reference	Furber et al. (2014)	Merolla et al. (2014)	Davies et al. (2018)

Table 4.1: Comparison of Large Scale Neuromorphic Processors. Adjusted from Shrestha et al. (2022)

5

Spiking Neural Controllers

The goal of this thesis is to develop two types of spiking neural feedback controllers that can control a MAV. One SNN will be developed to mimic the behavior of a tuned PID controller, from now on referred to as the *SNN PID controller*, and the other SNN will be trained using a type of machine learning algorithm, which will be referred to as the *Learned SNN controller*. An overview of the different types of implementations of the SNN controller discussed in this chapter is shown in Figure 5.1. This chapter will provide an in-depth overview and analysis of the available research considering the use of PID and Learned SNN controllers, applied to a variety of control tasks.

The layout of this chapter is structured as follows. Section 5.1 will focus on different types of structures of SNN that have been used in other research, which enable the spiking neural network to successfully approximate the proportional, differential and integral pathway of a conventional PID controller. To develop the Learned SNN controller, an overview and analysis of the different types of SNN machine learning algorithms applied to a type of control task is provided in Section 5.2. Each section will conclude with a discussion on which approach is best suited for the future implementation of each SNN controller.

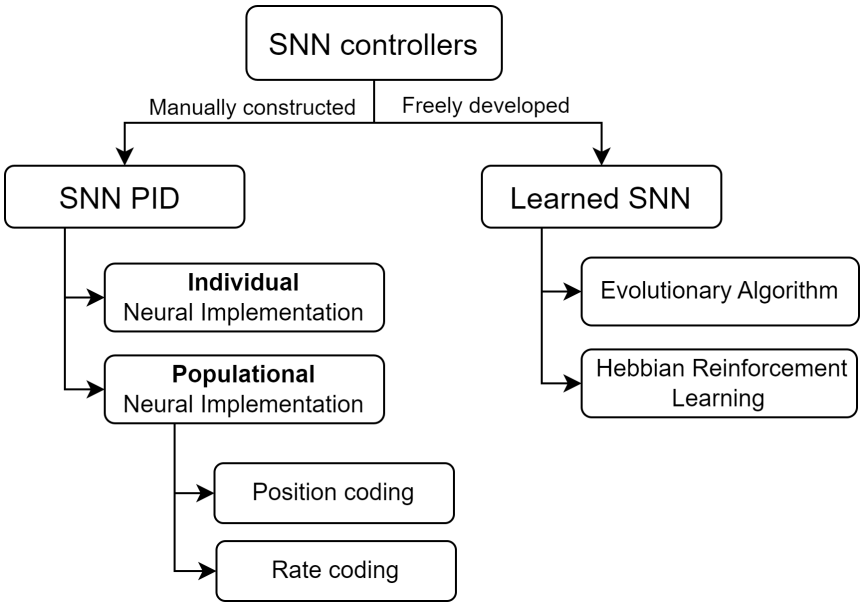


Figure 5.1: Overview of the different implementations for the SNN PID controller and the different machine learning algorithms for the Learned SNN controller

5.1. SNN PID Controllers

There are multiple types of SNN controllers that mimic the behavior of a PID controller. A categorization of two different implementations has been made and each will be covered in the following subsections,

together with some pros and cons of these implementations. Only the methods will be covered that have implemented a fully neuromorphic PID controller.

There is also an implementation of a PID controller that is able to deal with spike encoded data shown in Jimenez-Fernandez et al. (2012). However, this method uses blocks that are able to integrate and differentiate spike streams by internally converting the spikes into real numbers, performing the integration/differentiation and encoding this result back into a spike train. This, of course, is not a real neuromorphic implementation of a PID controller and thus will not be further covered in this section.

5.1.1. Individual Neural Implementation

This type of SNN only exists out of three spiking neurons in one single layer and is based on the work from Webb et al. (2011). Every neuron is used to recreate one term of the proportional, integral and derivative controller. The input to the neurons, which is an error signal, is scaled such that it lies within a biological range (e.g. a range of positive input currents) and inserted directly as input current to the neuron. Rate coding, by using an average spike count over a sliding window, is applied for the output decoding of each of the three neurons. To be able to use one spiking neuron for differentiation and integration, Webb has made use of the ability of *frequency adaption* (covered in Section 4.2.2), which is a characteristic of the Izhikevich neuron model. This means that the firing rate of the neuron can increase or decrease (depending on the sign of parameter d) when a constant input current is injected.

The 4 parameters of each of the Izhikevich neurons (a,b,c & d) are optimized using backpropagation with surrogate gradients function. The output weights of the neurons are not optimized and can be seen as the gains of the PID terms. The input (blue) and desired output (dashed green) training signal that was used during the supervised learning phase are shown in Figure 5.2. The desired output for the P and D neurons where the signal itself and its derivative. For the desired output of the integral neuron, a leaky integral was calculated. The loss function that was used was the Pearson product-moment correlation coefficient (PMCC) between the actual output and the desired output (Cohen et al. 2009). A PMCC close to one is desired, which means that there is a linear dependence between the two signals and a value of 0 means there is no linear dependence.

After the neurons were trained, a new test input sequence was used to show the performance of each trained neuron. The right plots show the spike frequency response of the P, I and D neurons to the test input sequence. It can be seen in the figure that the proportional and derivative neurons are able to follow the desired output quite well. However, when analyzing the PMCC that are presented in Table 5.1, it clearly shows that the PMCC of the P and I neurons are relatively high. This means that the discrepancy between the desired and actual output of the integral neuron can be solved by tuning the gain i.e. the weights of the neuron. Overall, it can be said that these three neurons show promising results, especially the ability of (leaky) integration and differentiation using a single spiking neuron is interesting.

Although the results are promising, there is a critical limitation for this approach of the PID implementation to work well. The problem arises when the error signal that is to be encoded is allowed to contain a negative range of numbers, which is almost always the case for most used controllers. The range of input error values is said to be scaled to a biologically plausible input current. This means that a very negative error is encoded to an input current around zero. An input current of zero does not result in spikes being

Table 5.1: The Izhikevich neural model parameters for each neuron together with the PMCC of both the training and test dataset Webb et al. (2011)

Function	a	b	c	d	PMCC with training	PMCC with test
Proportional	0.100	0.222	-61.6	0.0	0.976	0.958
Derivative	0.0105	0.656	-55.0	1.92	0.832	0.688
Integral	0.0158	0.139	-70.0	-1.06	0.920	0.920

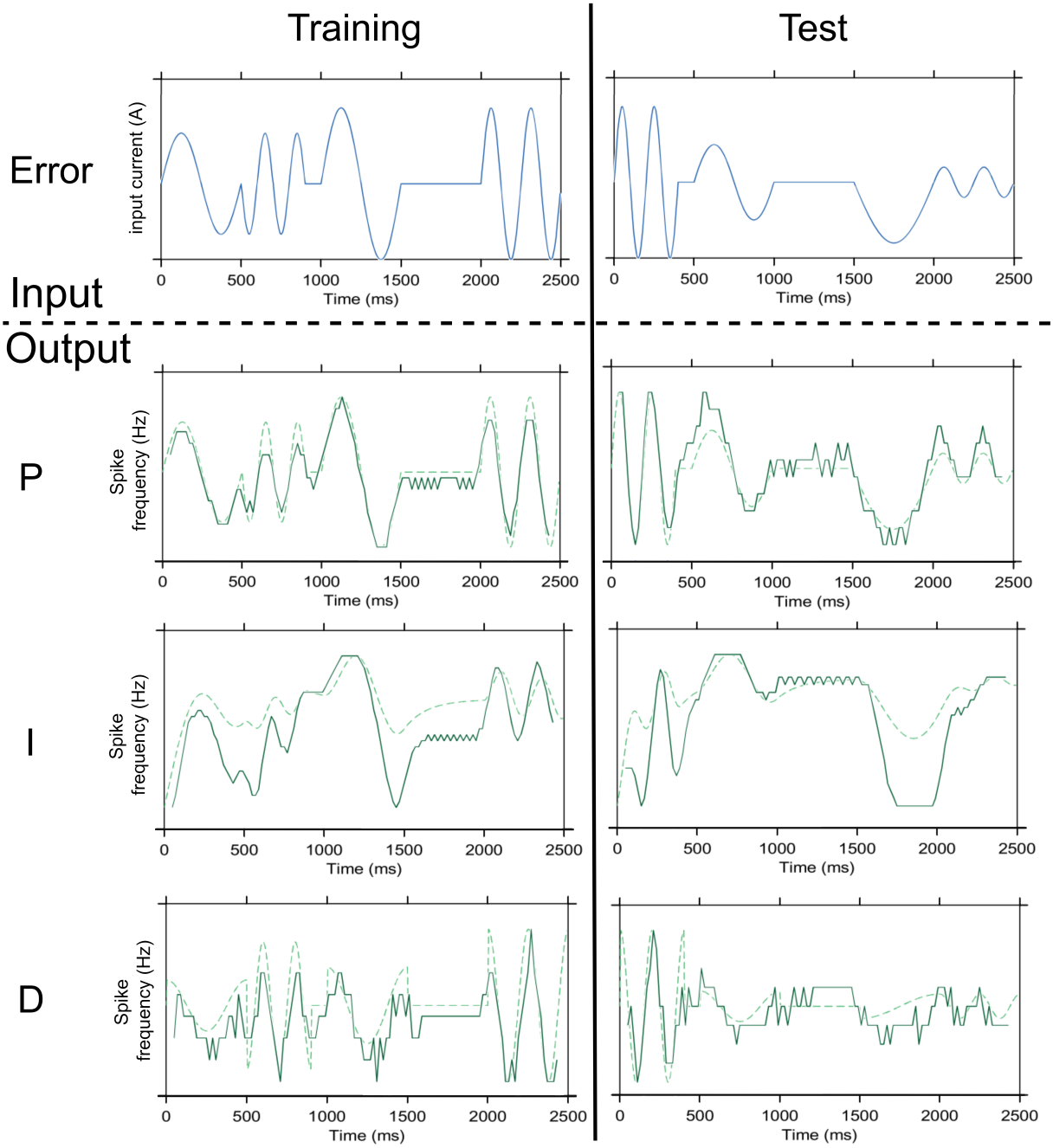


Figure 5.2: An overview of the results of the training (left) and test (right) dataset using the Individual Neural Implementation. The top plots show the input sequence that is used and the six lower plots show both the desired (dashed green) and the actual (continuous green) spiking frequency of the output neurons, each representing one term of the PID controller. Adjusted from Webb et al. (2011)

emitted. This means that the derivative and integral neuron will fail to perform useful outputs when it is dealing with a range of negative error values.

Nevertheless, for this research, it is not only useful to analyze *if* an SNN is able to learn to differentiate/integrate but also *how* it is able to do so, by closely analyzing the effect of frequency adaption. To start off, it is interesting to analyze the effect of the d parameter on the neuron model response. The d parameter is the amount that the recovery variable increases (d>0) or decreases (d<0) when a spike is emitted. A positive d means that the recovery variable is increased every time a spike is emitted. A higher

recovery value means that the change of the membrane potential over time ($\frac{dv}{dt}$) is decreased. With a positive d, the neuron is basically subtracting previous inputs (u>0) from the current input (I) as shown in Equation 4.8, which will be the approximation of the differentiation of the input. On the contrary, a negative d parameter leads to the addition of the previous inputs to the current inputs and acts then the neuron acts as a leaky integrator. The window over which the integration/differentiation takes place is scaled by the a parameter of the Izhikevich neuron. A larger a means that the window is smaller because the parameter a regulates the decay rate of the recovery variable.

These key insights into how the izhikevch neurons have learned to differentiate and integrate for a positive valued error signal will help in providing possible implementations for the SNN PID controller. It should be noted that the method of using only 3 neurons to mimic a PID controller has not been applied to any control tasks. The paper of Webb et al. (2011) only provided a proof of concept.

5.1.2. Populational Neural Implementation

This category of implemented SNN PID controllers consists of SNNs that use populations of neurons to mimic each term of the PID controller instead of using a single neuron as described in Section 5.1.1. The following subsections will focus on different methods for a neural population to be able to integrate or differentiate since these are the most difficult operations to realize in a neural PID controller.

Integral Implementations

Different methods have been used to approximate the integral term of the PID controller using a population of spiking neurons. These methods can be subdivided into two categories, based on the type of encoding that is used in the SNN: position or rate coding. There are also papers that have claimed to create a neuromorphic PI controller, where the ”integration” is performed by applying an exponentially decaying spike trace on the position encoded error signal (Zhao et al. 2020). Since this is not actually a neuromorphic implementation of integration, this research will not be further discussed.

The SNNs that used positional coding, used a one-hot type of encoding and decoding, meaning that there is only one neuron active (emitting a spike) in a population during one single time step. Both Stagsted et al. (2020b) & Stroobants, Dupeyroux, et al. (2022) used populations of neurons to implement a method to add two hot-encoded position values. Stagsted et al. (2020b) used a NxN population (where N is the resolution used for the encoding of the inputs) for the addition operation and the output population also had a resolution of N neurons. A similar method of integration for an SNN PI controller was implemented in Glatz et al. (2019). While Stroobants, Dupeyroux, et al. (2022) used a more efficient method for the addition operation by only requiring a 2N population to output the addition to a population with a 2N-1 resolution. In both works, the inputs of the addition module are the position encoded error and the output of the addition module itself, leading to the integration of the error. A disadvantage of this type of integration is that that the integrated signal quickly saturates when the time step is decreased. In Stagsted et al. (2020a), an integration method is described that fixes this problem. This method makes use of a two-step integration method with the first step being a single spiking neuron, since a neuron behaves as a (leaky) integrator as well, and the second step being a population of neurons (Kreiser et al. 2018). The output of this integration module is still a one-hot position encoded value for the integrated error.

An advantage of the position encoded method for integration is that it prevents integral wind-up to happen, since the integrated error over time is naturally bound by the highest and lowest representable number in the output population. An inherent disadvantage of using position encoding is the limited resolution of the encoded value (e.g. the integrated error in this case).

Another way of integration is found in Zaidel et al. (2021), which is based on a rate-based coding method and is implemented using the Neural Engineering Framework (NEF) (Eliasmith 2003). The principle of NEF uses a population of rate coded spiking neurons to encode a variable or state, x(t). An integrator can be approximated using a network of a recurrently connected population of spiking neurons as shown in Figure 5.3. The intrinsic dynamics of the neural population (e.g. A'&B') of LIF neurons is related to the standard control theoretic characterization as $A' = \tau A + I$ and $B' = \tau B$ (Eliasmith 2003), where τ is the

synaptic time constant. An integrator in the standard control form, $\dot{x} = Ax(t) + Bu(t)$ is the result of using $A = 0$ and $B = 1$ (e.g. $\dot{x} = u(t)$), which results in $A' = 1$ and $B' = \tau$. In NEF, the standard control notation of the integrator is denoted by Equation 5.1 (Zaidel et al. 2021).

$$x(t) = h(t) * [A'x(t) + B'u(t)] \tag{5.1}$$

where $h(t)$ is a temporal linear filter that is applied to the arrived spikes, $A' = 1$ and $B' = \tau$. The dynamics of this integrator can best be described by analyzing the spiking activity in the neural population of Figure 5.3. When returning to the basics, an integrator has two functions: 1) save the state to the next timestep and 2) add or remove an input variable. The A' of 1 means that the encoded state value x(t) is directly transferred back into the population and it thus fulfills the first function. The second function of the integrator is implemented by using the $B' = \tau$, which adds the input to the state (multiplied with τ). A similar implementation of the integral calculation for a PI controller for a flapping drone in simulation was used in Clawson, Stewart, et al. (2018).

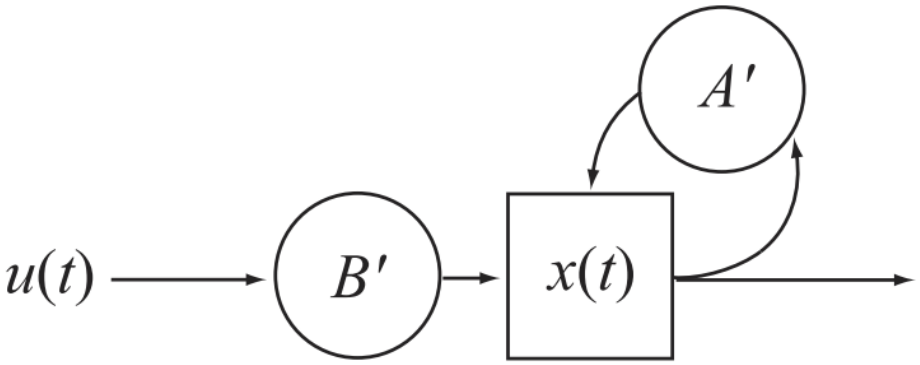


Figure 5.3: Block diagram of a recurrently connected population of neurons, $x(t)$, which encodes the state vector, A' is the neural dynamics weight matrix and B' is the neural input weight matrix. The neural population is denoted by the square box and the weights are illustrated by the circles.(Eliasmith 2003)

There are some important remarks to make about the use of this type of integration. The encoded state is actually an approximation of the state ($\hat{x}(t)$) and not the exact state ($x(t)$). This means that the product of A'$\hat{x}(t)$ is not equal to A'$x(t)$. The effect of the error of the approximation of x on the integrator can be analyzed by assuming that the state is perfectly encoded ($\hat{x}$ = x) but the imperfection of the approximation is the result of an offset in A'. For the perfect integrator, A' is equal to one. When A' is slightly lower than 1, it means that there is a leak in the integrator. An A' that is larger than one results in an unstable integrator, which adds an extra fraction of x(t) to the x(t) of the next time step. In practice, this type of integrator is thus either leaky or unstable.

Derivative Implementations

After analyzing the available implementations of the derivative term in a population of spiking neurons, different methods have been identified and will be discussed in this section. Similarly to the integral implementation discussed in the previous section, different methods for neuromorphic differentiation have been found using two different types of coding: position and rate coding. Additionally, there are studies that have implemented a neuromorphic PID controller (Stroobants, Dupeyroux, et al. (2022) & Vitale et al. (2021)) that calculate the derivative without the use of spiking neurons and input the encoded derivative value directly in a spiking neural network. Since this is not a neuromorphic implementation of differentiating, these will not be further covered in this subsection.

Stagsted et al. (2020b) and Stagsted et al. (2020a) used a one-hot encoded type of position encoding for the derivative implementation. Similarly as explained in the previous subsection for the integral implementation, these papers used a population of neurons to create a one-hot encoded module for the subtraction of two position encoded signals. This is designed by carefully selecting which neurons are connected and setting their weights and threshold accordingly to create a subtraction module of two position encoded signals.

Instead of using position coding, the D term in the SNN PID in the research of Zaidel et al. (2021) was implemented using two rate coded populations of neurons. The population that encoded the error was connected to another population of neurons through 2 different synapse connections: one with a short and one with a longer synaptic time constant τ. The information passed through the synapses with the short time constant should encode the more recent value of the variable, while the long time constant encodes more of the past values of the variable. By subtracting the input from the short τ with the long τ, an approximation of the rate of change of the error is created. A balance should be found in selecting both time constants, such that the data is transferred through these synapses can be used to correctly approximate the derivative. This research lacks to provide evidence that the population of neurons correctly approximates the derivative of the input signal. It has shown the combined effect of the PD and SNN PID controllers on a target reaching task, but lacks isolated proof of the ability of differentiation. This method of differentiation should be further analyzed, since no other research has been found concerning this topic, to prove that this is a reliable method of differentiation.

5.1.3. Evaluation of SNN PID Controllers

As discussed in the subsections above and shown in Figure 5.1, the different implementations of a fully neuromorphic PID controller can be split into roughly three categories: individual, position coded populational and rate coded populational. This subsection provides an analysis of the efficiency and performance of the different types of SNN PID Controllers during control tasks.

Individual Neural Implementation

It is important to consider the complexity of the system that is controlled when analyzing the performance of the different SNN controllers. The individual neural implementation of the SNN PID controller from Webb et al. (2011), which was implemented on the SpiNNaker architecture, was only used to correctly apply the P, I and D terms on a sinusoidal input current that would encode an error signal with only a positive range of numbers. It was not used to control a system, solely a proof of concept was provided and no other research was found that applied this type of SNN controller to a control task. In contrast to the populational neural implementations, which were used for control tasks with varying levels of complexity.

Position Coded - Populational Neural Implementation

The postion coded populational controllers were used on multiple types of control tasks. Each of the position coded populational controllers discussed here was implemented on the LoiHi. The SNN PID from Stagsted et al. (2020b) was used on a PushRobot and one a 1 DoF of a birotor mounted to a structure that only allowed the birotor to roll[1]. The exact same birotor system was used in Stagsted et al. (2020a). While the performance of the roll control in both studies was comparable[2], the SNN PID controller in Stagsted et al. (2020a) was realized using fewer neurons ($\pm$ 40k instead of $\pm$ 65k). The main reason for the increased efficiency of the SNN is the different methods used for integration, as discussed in Section 5.1.2. The position coded populational controller in Stroobants, Dupeyroux, et al. (2022) was designed to control the height of a flying drone, while the angles were stabilized using a non-spiking PID. This SNN PID controller only consisting of 138 neurons showed to be able to achieve stable and robust control, with a larger settling time and overshoot compared to the non-spiking PID controller. There are multiple factors that help explain this robust control behavior while using significantly fewer neurons compared to the SNN implemented by Stagsted: a lower resolution of the input/output variable, the use

[1] For the birotor, a cascade SNN PID controller was used to control both the angle (outer) and the rate (inner)

[2] Compared with a manual tuned PID controller, the SNN PID controller had relatively larger overshoot, low rise time and longer settling time

of a quadratic distribution of the control output instead of a linear distribution and the derivative is not calculated but directly encoded into the SNN.

Rate Coded - Populational Neural Implementation
The rate coded populational PID controller in Zaidel et al. (2021) is used to control a robotic arm with 6 DoF in simulation and in real life implemented on the LoiHi using the NEF framework. The use of the NEF framework allows the controller to work on numerous neuromorphic hardware. In this paper, there was also an SNN implemented to calculate the inverse kinematics (IK) of the robotic arm, meaning that the target position of the end-effector (in cartesian coordinates) was transformed into target angles of each of the 5 joints. These target angles were used as reference input in five separate SNN PIDs.

When evaluating the efficiency of the rate coded SNN during the control task, it is useful to analyze how many neurons are used in order to make statements about the efficiency of the SNN. A population size of 250 spiking neurons is used in this work, which makes a total of 1000 neurons for a single SNN PID controller. Decreasing the number of neurons per population negatively influences the ability of the population of neurons the encode the correct value. The difference between the value that a population of neurons encodes ($\hat{x}$) and the actual value that should be encoded (x) is called the representation error. This error is caused by the finite number of neurons in the population and the relation between the number of neurons per population and the representation error is found to be $E \propto \frac{1}{n^2}$ (Eliasmith 2003).

Increasing the number of neurons per population is not the only parameter that would decrease the representation error of the population. The distribution of the neuron tuning curve (intercepts and maximum firing rate) also have a large impact on the representation error of the encoded variable. A visualization of two different types of distributions is shown in Figure 5.4. A uniform distribution of the tuning curves is shown in Figure 5.4.A, while Figure 5.4.B shows a triangular distribution where the intercepts (the value at which the neuron starts/stops spiking) are more centered around the midpoint of the input range. This non-uniform distribution leads to a decrease in the representation error of the population of neurons (DeWolf et al. 2020). The reason for this is that there are a relatively large number of neurons in the uniform distribution that almost always spike or almost never spike, such as depicted by the thick black line in the right image of Figure 5.4. This line shows a neuron that is inactive for 87% of the input range, which means that it does not encode useful information for the range where the spiking frequency is zero.

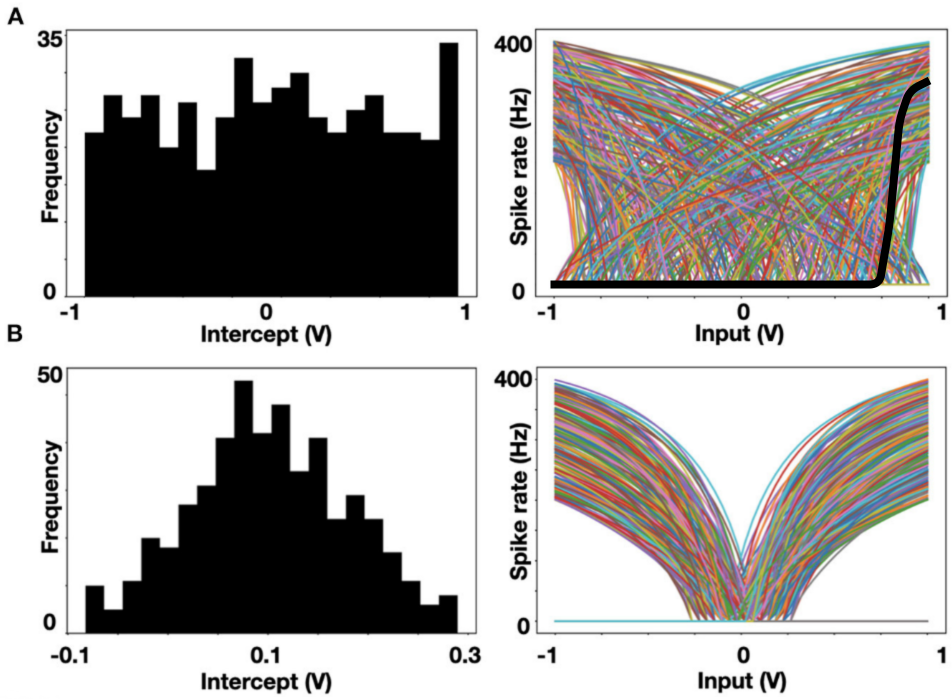


Figure 5.4: Difference in the distribution of the neuron tuning curves in a population of spiking neurons. A) A uniform distribution B) A triangular distribution. The black line in the right image of A indicates a neuron with a intersect around 0.75V. Adjusted from (Zaidel et al. 2021)

Performance Comparison of SNN PID Controllers

After analyzing the three different types of SNN PID controllers in the previous subsection, a trade-off can be performed to find the best approach for implementing the SNN PID controller. To start off, although the individual neural implementation showed some impressive results by implementing an SNN PID controller with only 3 neurons, the limitation that prevents the input error to be negative makes it an unsuitable candidate for the SNN PID drone controller. The insight this minimal implementation has provided, into the effect that frequency adaption can have in enabling integration/differentiation, can potentially be transferred into a successful PID controller. It could be possible to add additional neurons that can deal with the integration of differentiation for the negative inputs, but this has not been performed in any research so far. The additional neurons can potentially make this implementation a suitable candidate for the SNN PID controller for the SNN PID controller. However, since it has not been implemented yet and thus there is no information on its performance, it will be left out of the following performance comparison.

This leaves both the populational neural implementations for the performance comparison. It is difficult to directly compare the performances of the different SNN PIDs because neither the characteristics of the SNN (e.g. number of neurons) nor the system that is controlled is equal. To be able to make a useful trade-off, an analysis of the performance and efficiency of the methods for implementing mathematical operations (e.g. integration or differentiation) will be performed. This will provide some insight into the overall performance and efficiency of the SNN PID controller. Additionally, the ability for online adaption possibilities and the robustness of the controller will also be added to the trade-off.

The main difference between both implementations is the type of coding they use. The type of coding used in the SNN directly impacts the way how calculations are performed within the network, as discussed in the previous subsection. Position coded networks use a sparsely connected population to resemble the addition/subtraction modules that form the basic element of operations. On the other hand, the equivalent of a rate coded population is a densely connected population of neurons with recurrent connections. The efficiency of such a basic element is determined by the computation power that is required to output a result with a certain level of accuracy. The level of accuracy, in this case, would be equal to the representation

error of the population. The computation power of a population of neurons is largely dependent on the number of neurons used and the total spiking activity of the population. The relation between the representation error and the number of neurons for rate coded populations was found to be $E \propto \frac{1}{n^2}$ (Eliasmith 2003). In a one-hot positional encoded population of neurons, the representation error due to the limited number of neurons is equal to $E \propto \frac{1}{n}$ (i.e. the bin size of the position encoding). So, larger populations of rate encoded neurons are generally more efficient than the equivalently sized position coded populations.

When analyzing the robustness of the SNN PID, it can be useful to evaluate the control performance when part of the population of neurons is silenced. One research had developed a rate coded SNN controller that mimicked the behavior of an LQR controller for a target reaching task of a spring-mass-damper system (Slijkhuis et al. 2022). While the SNN was executing the target reaching task, part of the total number of neurons in the population was silenced. The SNN controller showed to be still successful in this task when even half of the population was killed. Even though no research was found that made a similar analysis, it would make sense that a position coded population with half of its neurons being silenced would have more problems in adapting successfully. The reason for this is that a neuron in a position encoded population fulfills one specific task in the population, so silencing it means that this particular task can not be executed. While a neuron in a rate coded population shares more ”responsibilities” in the execution of a specific task, making them more robust to silencing.

Almost all studies that have implemented an SNN PID controller, make a statement in the conclusion that online adaption can be the next step for future research (Webb et al. (2011), Stagsted et al. (2020b), Stagsted et al. (2020a), Zaidel et al. (2021)). Both types of controllers have the ability to use an online learning algorithm on the connection weights that resemble the gains of the controller. However, when further analyzing the ability of online adaption, it should be noted that SNNs that do not use position coding have more weights and thresholds that can be adjusted in this process. Meaning that in the position coded structure, almost all weights and thresholds are manually selected to perform a certain operation, like addition or to enforce one hot encoding. In a rate coded structure, there are more variables, like weights and thresholds, that can be adjusted online to increase the solution space of the controller.

To conclude, after trading off all arguments provided above, the main focus will be on the implementation of a rate-based populational SNN PID controller for this thesis. This type of SNN PID controller has shown to be a potential candidate for a successful drone controller due to its performance, efficiency, robustness and ability to enable online adaption. An additional potential solution could be the individual neural implementation with additional neurons to deal with the negative input variables. However, this has not been implemented and applied to control tasks yet, so it is hard to estimate its performance.

5.2. Learned SNN Controllers

Next to the SNNs that are used to mimic a PID controller, it is also possible to train an SNN using a more ”free” approach where e.g. the topology of the SNN is not necessarily fixed during the training phase. In the SNN PID, the spiking networks are either manually designed to mimic a PID controller (e.g. the position encoded populations) or using some form of supervised training to make the SNN behave like a PID controller (e.g. the Individual and Rate coded populational neural implementations).

In this section, an analysis will be performed on the available research that has used a more ”freely developed” type of machine learning algorithm to train an SNN to perform a control task: Evolutionary and Hebbian Reinforcement Learning algorithms. Similarly to the previous section, the control task covered in this section will not be limited to drone control, but also includes tasks like the control of a driving robot or the movement of a robotic arm.

5.2.1. Evolutionary Algorithm

Evolutionary algorithms are powerful learning algorithms that can deal with the non-continuous behavior of SNNs. In Section 4.4.4, an introduction was provided on the basics of EA. This section will focus on prior research on SNN controllers that have been trained using EA for different types of control tasks. More specifically, the distinction between the different implementations of the EA and the achieved performances

of these learned SNN controllers will be analyzed.

Before exploring the SNN controllers that have been developed to control (a part of) a MAV, let us first start with non-flying control tasks. In Pérez et al. (2018) is an SNN controller trained to control a non-linear model of the elbow joint of an arm. This research makes use of a fixed topology of 5 neurons, 3 input and 2 output neurons, where not only the connection weights are optimized, but also the parameter of the Izhikevich neural model. An *differential* EA is used, which is a greedy algorithm. This implies fast convergence, but it has poor exploration so it can get easily stuck in a local optimum. This type of differential EA functions well for relatively simple control tasks, where there are fewer local optimums. An EA that explores more of the solution space at cost of computational cost, is the NEAT algorithm (introduced in Section 4.4.4), which is used in various studies considering spiking controllers (Qiu et al. (2019), Qiu et al. (2020a) & Vandesompele et al. (2017)). This algorithm was used to optimize both the topology and connection weights of an SNN that had to control the balancing cart-pole system (Qiu et al. 2019). The controller was able to successfully balance the pole on the cart, even when only the cart position and pole angle were inserted into the SNN instead of also inserting the velocity components of the states. It should be noted that the output neuron, that drove the force exerted on the cart, was non-spiking with a sigmoidal activation function. The outputs of the hidden neurons were rate decoded such that the input to the output neuron is a weighted sum of firing rates.

A similar type of EA, which can optimize both the topology and the synaptic weights, was also used to train an SNN for drone control in simulation. The SNN controller in Howard et al. (2014) is able to control the roll, pitch and thrust of a drone to bring the drone as close to the target position as possible, under a variety of wind conditions [3]. The EA used here had two interesting features: self-adaptive mutation rates and incremental evolution. Both proved to increase the performance of the controller. The mutation rate indicates the probability of a certain event happening during the creation of new individuals (e.g. the addition of a neuron). The special thing about these mutation rates was that they also were part of the genome. There are different mutation rates for the different types of mutative actions that are possible such as changing the weights and adding/removing a connection or neuron. The second interesting feature of this EA is the implementation of incremental evolution. During the evolution, the wind conditions became step by step more difficult, starting from 1 m/s steady wind up to 5m/s gusting wind. In the end, due to the incremental evolution, the controller was able to perform target tracking tasks while experiencing the 5m/s wind gusts. On the contrary, the SNN controllers that were directly trained in these wind conditions failed in learning this task. In this work, Howard et al. conclude that to transfer the controller to a physical drone, its ability to make online adaptions is important.

This is exactly the main focus of the research in Qiu et al. (2020a), where an SNN controller is first trained on a simple linear hexacopter model and after training this model is exploited on a more complex nonlinear model (Santoso et al. 2018), but with an online adaptation rule enabled. The idea is that the step from the simple to the complex model simulates the reality gap. The online weight adaptation rule implemented was a rate-based Hebbian learning rule (Izhikevich and Desai 2003), where the parameters of this learning rule were also separately optimized using the NEAT algorithm. It should be noted that control in this research is limited to height control only. Nevertheless, an SNN controller trained on the simple model with an optimized Hebbian learning rule exploited on the more complex non-linear model showed to outperform a PID controller that was directly tuned on the more complex model.

Finally, an SNN controller had been trained to control the full 6 DoF of a flying hexacopter in simulation in Qiu et al. (2020b). The simulated model is the same complex model as in Qiu et al. (2020a), but no online adaption is implemented in this neural model. An overview of the cascade controller that is used to control the drone is shown in Figure 5.5. For each DoF, the tuned PID controllers are one-by-one replaced by an SNN and subsequently are going through an evolutionary cycle using a variation of the NEAT algorithm before replacing the next PID for an SNN. The fully implemented SNN controller outperformed the full PID controller with a smaller rise time and almost no overshoot. It should be noted that similarly to the SNN in Qiu et al. 2019, the output neuron of each controller is non-spiking and has a sigmoidal activation function. Additionally, there are two interesting observations when analyzing the control diagram in Figure 5.5. First

[3] yaw was decoupled and stabilized by a proportional controller

of all, the controller developed here only controls the angles and not the rates. The second observation is that not only the error of the controlled state is used as an input, but also the velocity of that state. These two design choices make it easier for the controller to be successful, since only the high-level controller is neuromorphically implemented (i.e. the angle controllers) and each controller has more knowledge about the state (i.e. the additional velocity input simplifies the derivative computation).

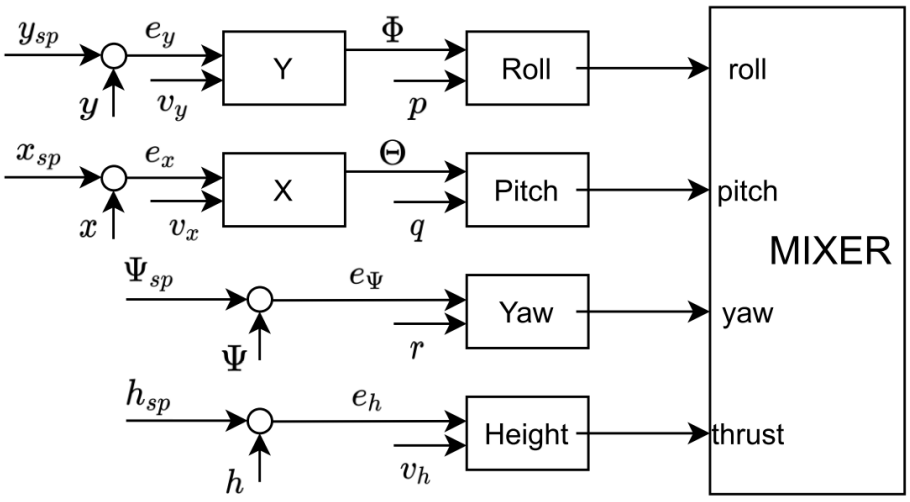


Figure 5.5: The control diagram of the hexacopter. Each of the 6 controllers has two inputs: 1) the error between the setpoint and the actual state (i.e. for the position errors: e_x, e_y, e_h and the attitude errors: e_Θ, e_Φ, e_Ψ) and 2) the current velocity of the state (i.e. the positional velocities: v_x, v_y, v_h and the angular velocities: q, p, r (Qiu et al. 2020b)

To conclude this section, there are quite a number of relevant studies that have used an EA to train an SNN to control some complex systems. The research of Qiu et al. and Howard et al. showed promising results for an SNN controller used to control a drone, which is in line with the goal of this thesis. However, all the SNNs presented here were limited to angle and position controllers, without a rate controller. So, the development of a fully neuromorphic rate controller for a drone still stays an open challenge. Additionally, the research of Qiu et al. (2020a) showed promising results for the implementation of an evolved online adaption rule that could be used to cross the reality gap. However, the controller was not implemented to achieve full position control, only height control was realized. Finally, it should be noted that all presented work was only implemented in simulation and none were actually tested on a real drone.

5.2.2. Hebbian Reinforcement Learning

The idea of Hebbian reinforcement learning is to combine the unsupervised Hebbian learning rule (e.g. STDP) with a reward signal, that provides information on the desired behavior of the controller. The most used type of Hebbian RL algorithm is the R-STDP (introduced in Section 4.4.3). This section will focus on prior research on SNN controllers that have been trained using R-STDP for different types of control tasks. More specifically, the distinction between the different implementations of the R-STDP and the achieved performances of these learned SNN controlled will be analyzed.

An SNN trained with R-STDP has proven to be a successful controller for different types of control tasks, such as robotic arms, driving robots and flapping drones (Pérez Fernández et al. (2021), Bing, Meschede, et al. (2019) & Clawson, Ferrari, et al. (2016)). One interesting aspect of R-STDP to analyze is the way the reward is created. The density of the reward function can differ quite a lot. In Ramezanlou et al. (2020), an SNN is used to control a 2 DoF robotic arm in simulation for the task of obstacle avoidance and target tracking. However, a sparse event-based reward function was used that only send out a positive reward when the target was reached and a negative reward if the obstacle was hit. With a sparse reward function, the credit assignment problem arises, since it becomes harder to exactly identify what previous

behavior has led to the received reward.

The are also SNN controllers that have been trained with more dense reward functions that were able to control a robotic arm (Spüler et al. (2015) & Pérez Fernández et al. (2021)) and an obstacle avoiding driving robot (Lu et al. (2021)). All these researches created a reward function that would minimize the error of the controlled state. For example, Spüler et al. (2015) used a reward function that would send out a reward based on the difference between the target and the actual joint angle for every 50 ms. In Pérez Fernández et al. (2021), the reward was sent out every 0.1 ms, which makes the reward function very dense.

Next to the types of reward functions discussed previously, there is an additional type of reward function that uses the controller's actual output and desired output to determine the reward. This type of reward function thus needs a supervised control signal. The supervised signal used in Bing, Baumann, et al. (2019) came from a prerecorded dataset and was used to train two separate controllers for a driving robot, one for obstacle avoidance and one for goal seeking. However, in Clawson, Ferrari, et al. (2016), the supervised signal came from an additional LQR controller. This enabled the use of online adaption, which was not possible when using a dataset. In this work, a robotic bee was able to follow a trajectory and stably hover using an SNN lateral controller.

So far, all these approaches described in this section, have used the reward signal directly to directly adjust the connection weights of the SNN controller, It is, however, also possible to train a critic SNN controller using R-STDP and then use this critic network to adapt the weights of the actor network. In Foderaro et al. (2010), an SNN critic network, which is trained using R-STDP, is used to adapt an SNN actor that is used as a longitudinal flight controller. The reward signal is dependent on the error between the target optimal controller output and the actual output of the SNN actor controller. This more indirect approach of adjusting the weights can be useful for potential applications where direct manipulation of the synaptic weights is impossible, such as for prosthetic devices implanted in the brain (Bing, Meschede, et al. 2019).

To conclude, the use of reward-based Hebbian learning algorithms for the training of SNN controllers has proven to be successful on a variety of control tasks. Moreover, the use of a biologically plausible Hebbian learning rule often enables the ability of online adaption for the controller. The type of reward function that is used has a large influence on whether or not a controller is able to learn to control a complex system. One of the more complex control tasks, the hovering of the RoboBee (Clawson, Ferrari, et al. 2016), used a very dense reward function that also required the knowledge of a desired outcome of the controller. It remains unclear if the controller would also be successful if a less supervised learning signal (e.g. the reference error) was used to construct the reward function. With regards to this MSc thesis, this type of reward-based Hebbian learning algorithm could potentially be used for the training of the Learned SNN controller for a MAV in combination with a dense reward function.

5.2.3. Evaluation of Learned SNN Controllers

After analyzing both learning algorithms in the previous subsection, it is interesting to compare the potential of each learning algorithm for the Learned SNN MAV controller. The EA can be used to search through a larger solution space than the Hebbian RL algorithms, not only the connection weights are optimized (as is often the case for the Hebbian RL algorithms), but also the network topology and even the neural parameters of each of the neurons can be adapted. In other words, the solution space of an SNN trained with the RL Hebbian algorithm is a subset of the solution space of the SNN trained using an EA. This implies that an SNN trained with EA should in theory always be able to find a solution that performs at least equal to or better than the SNN trained with the RL Hebbian algorithm. However, this will require more computation power and it is also limited to offline training and can thus not be used for online adaptation. Part of these disadvantages is not very relevant for the development of the SNN controller in this MSc thesis, since the training is performed offline and thus the computational power is not limiting.

On the other hand, a Hebbian RL algorithm can often be used for online training of the neural network, depending on how the reward function is constructed. The potential of online adaption makes this learning algorithm useful for the development of a robust MAV controller that is able to adjust to variations in drone

specifications or to extreme environmental conditions. In Qiu et al. (2020a), the researchers have shown the useful effect of the combination of offline training using EA and online training using a Hebbian learning rule. However, they have not implemented an additional rewarding signal to the Hebbian learning rule, which can potentially increase the performance gain of the online learning rule, since the rewarding signal helps in identifying desired and undesired behavior.

To conclude, for the development of the Learned SNN MAV controller, a combination of an EA for the offline training and a Hebbian RL algorithm for the online training of the SNN will be further analyzed. Both learning algorithm has shown promising results during a variety of control tasks and a combination of both can be a good candidate for the Learned SNN controller.

Part III

Additional Results & Conclusion

6

Additional Results to the Scientific Paper

This chapter will present and discuss additional results beyond those provided in the scientific paper. The first section will focus on the test results of the separated PD and integral controllers on the real-world blimp. The second section will cover an in-depth analysis of the cause of the large initial oscillations observed when using the SNN controller to control the negatively buoyant blimp.

6.1. Separate PD and Integral Spiking Controller

As stated in the scientific paper, the neuromorphic controller developed in this thesis consists of a separate spiking PD and Integral controller. To isolate the effect of each SNN controller, different levels of buoyancy have been used on the real-world blimp. The neutrally buoyant blimp does not require an integral to be present in the controller to effectively control the altitude of the blimp. This makes the neutrally buoyant blimp an ideal test case for the PD controller. For each different hidden layer structure of the spiking neural network (IWTA and/or Recurrent), the SNN with the lowest cost value on the training dataset was selected. This resulted in four SNN PD controllers that have each been tested on the neutrally buoyant blimp and the results are presented in Figure 6.2.

The scientific article only showed the best-performing spiking PD controller, which was selected based on the performance on the test dataset. This figure shows the altitude tracking of multiple step responses using all four evolved spiking neural network controllers on the neutrally buoyant blimp. The first three controllers (LIF, IWTA-LIF & R-LIF) effectively reduce the oscillations to a level similar to that of the non-spiking tuned PD controller. These steady-state oscillations are caused by the dead-zone present in the output of the motor controller. Among the spiking controllers, only the LIF PD controller succeeds in minimizing overshoot and oscillations to a similar level as that of the non-spiking controller.

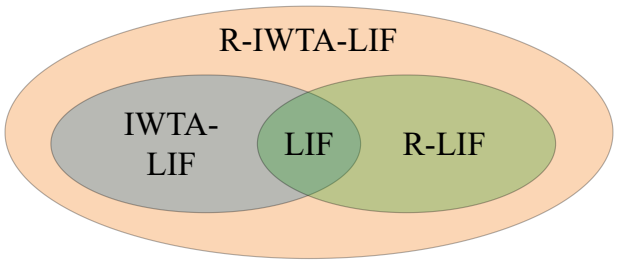


Figure 6.1: Visualization of the solution space of the different controllers that have been evolved. The size of each area is not relatively scaled towards the actual size of the solution space it represents

When analyzing the training results, it is important to note that the solution space of the best-performing spiking PD controller, the LIF SNN, is a subset of all the other evolved SNN controllers. This is visually represented in Figure 6.1. This means that the Evolutionary Algorithm still got stuck in a local optimum and that finding the global minimum remains a challenge. Although the additional threshold parameters (W^{th}, τ^{th}) and recurrent weights (W^{r}) have increased the solution space, it did not lead to a better-performing controller. A reason for this could be that the LIF SNN already has the ability to implement an approximation

of the derivative, used in Zaidel et al. (2021) and Stroobants, De Wagter, et al. (2023). Both studies have used two groups of neurons with different time constants: one with fast time constants to get an output proportional to the input and the other with slow time constants to get a delayed version of the input. Subtracting the one from the other leads to an approximation of the derivative.

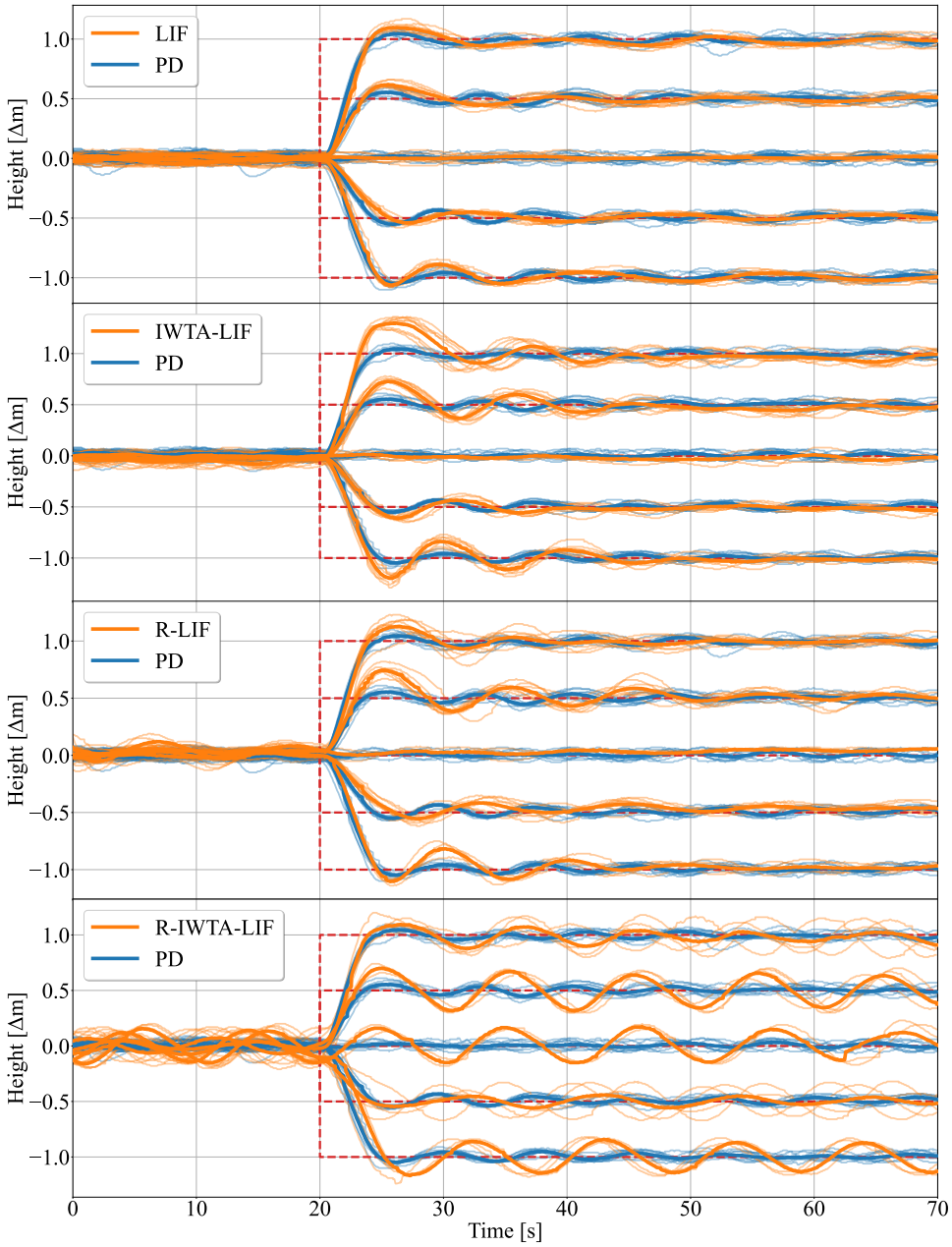


Figure 6.2: Real-world step responses of the altitude of the **neutrally buoyant** blimp using a conventional PD control and all different spiking PD controllers. Each setpoint was tested eight times for every controller and the average is shown by the thick line

All recorded results using the integral controller on the negatively buoyant blimp are shown in Figure 6.3. The full recorded dataset shows that only the IWTA-LIF integral controller is able to consistently minimize the steady-state error. The R-LIF shows a constant steady-state error of $\pm$5cm, while the combined R-IWTA-LIF shows inconsistent performance when reducing the steady-state error.

When comparing the altitude tracking performances of the PID and PD + IWTA-LIF controllers (as

depicted in the top plot of Figure 6.3), it's worth highlighting the variation in overshoot between the smaller ($\pm$0.5m) and larger ($\pm$1.0m) steps. As expected, the overshoot for the non-spiking PID controller increases for larger steps due to the build-up of the integral when it receives the new setpoint. However, the overshoot of the IWTA-LIF neurons does not increase significantly for larger step sizes. A possible cause of this behavior could be the delay that is present in the IWTA-LIF integrator, which will be further analyzed in the next section of this chapter. This delay prevents the initial build-up of the integral during the rise time of the transient response, resulting in less overshoot.

The IWTA-LIF integrator did not completely eliminate the steady-state error, while the IWTA integrator in Stroobants, De Wagter, et al. (2023) did successfully remove the steady-state error. A possible cause could be the differences in the implementation of the IWTA mechanism.

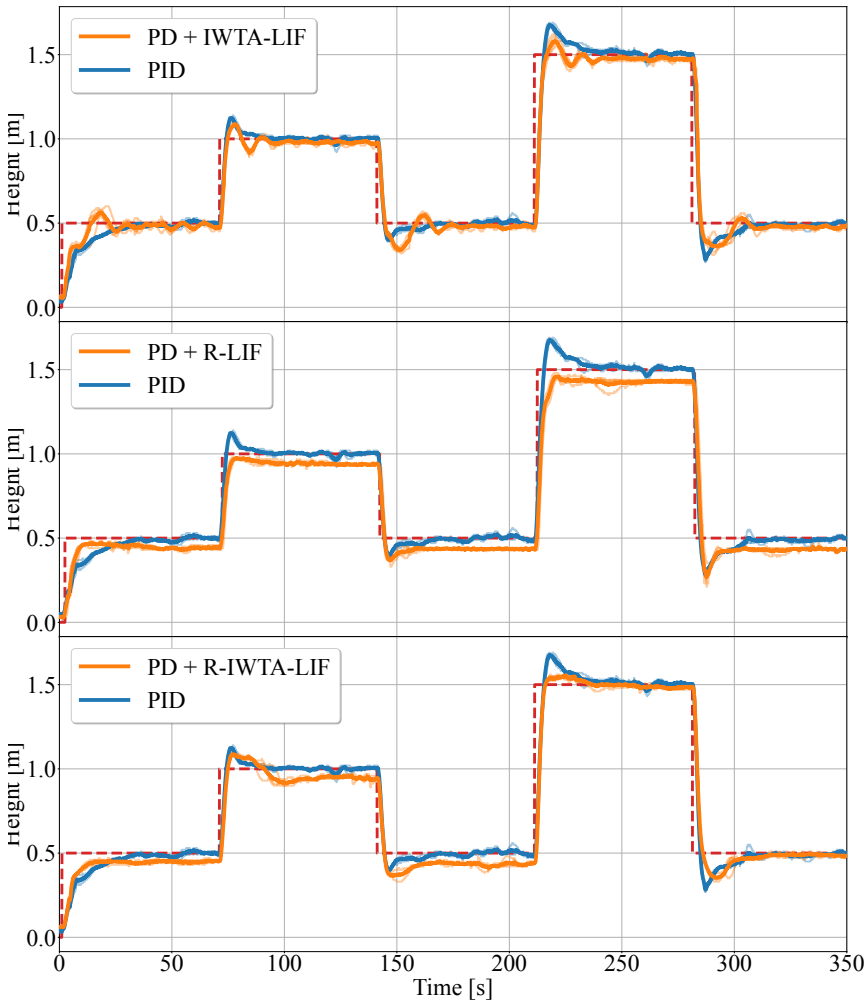


Figure 6.3: Real-world step responses of the altitude of the **negatively buoyant** blimp using a conventional PD controller and different spiking integrators. The average over five runs is shown for each controller.

To explore a new implementation of the IWTA, which enables more complex spiking patterns (such as frequency adaptation as discussed in Section 4.2.2), we have added a decay parameter to the threshold. This allows the threshold to return to the base value when no input is received. Introducing a decay parameter to the threshold also implicitly bounds it, removing the necessity for additional threshold bounds in the integration layer to prevent integral windup (as was done in Stroobants, Dupeyroux, et al. (2022)). Varying the threshold of the neurons in the hidden layer plays a crucial role in the approximation of the integral within the IWTA SNN. However, if the threshold decays, it can introduce errors in the integrated

error encoded by the spike activity of the SNN. This could lead to an offset in the integrated error, which then results in a steady-state error when controlling the blimp's altitude.

An alternative perspective regarding the cause of the steady-state error is that the training algorithm became trapped in a local minimum. The integral implementation of Stroobants, Dupeyroux, et al. (2022), where no decay parameter was used, was actually a subset of the solution space of the IWTA-LIF integrator controller trained in this work. This means that the EA could have learned to set all the decay parameters of the threshold to 1, which means no decay. This would be equal to the IWTA implementation in the previously mentioned study. Similarly to the spiking PD controller, it indicates that increasing the solution space with additional network parameters makes finding the global minimum challenging task. Even for Evolutionary Algorithms, which are known to be capable of dealing with complex non-linear solution spaces (Katoch et al. 2021), finding the global minimum is extremely difficult.

6.2. SNN Controller Performance Analysis

The altitude tracking performance of the combined spiking PD and integral controller is shown in Figure 6.4, which is identical to Figure 10 in the scientific paper. To further analyze the large oscillations present in the SNN controller during the downward step response, we looked at the output of the SNN controllers during both the upwards (indicated by A) and downwards (indicated by B) steps, presented in Figure 6.5 and 6.6 respectively. Each figure shows the separate and combined output of the PD and Integrator SNN controller, as well as the reference non-spiking PID controller.

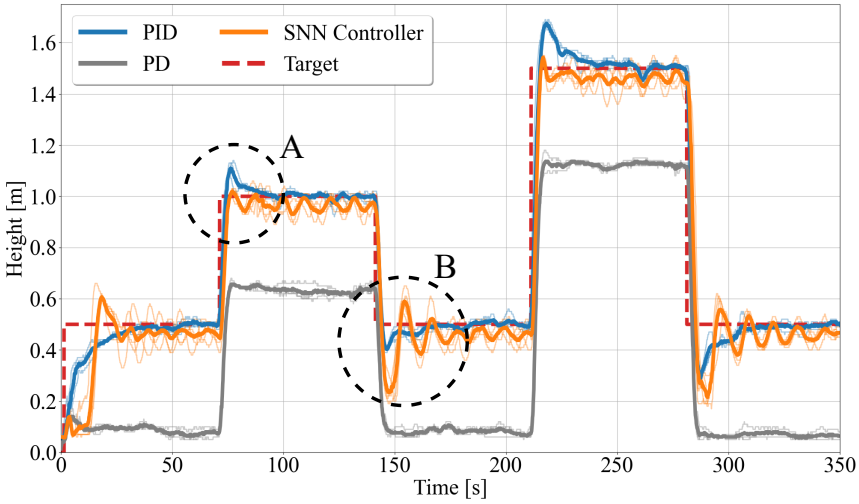


Figure 6.4: Real-world multi-step response of the altitude of the negatively buoyant blimp using a non-spiking PID & PD controller and the SNN controller, which is the combination of the SNN PD (LIF) and the SNN integral (IWTA-LIF) controller. The average over five runs is shown for each controller. Two areas of interest are highlighted by the letter A & B and are shown in Figure 6.5 and 6.6

There are a couple of interesting observations when reviewing the output of the SNN controllers during the upward step command shown in Figure 6.5. The spiking PD controller shows a slight delay when receiving a new setpoint, resulting in a lower maximum output compared to the non-spiking PD controller. Nevertheless, the spiking PD controller is capable of effectively dampening the movement, which minimizes the overshoot. When analyzing the output of the integrator controllers, it shows that the spiking integrator has a significantly delay compared to the non-spiking counterpart ($\pm$3s, which is the time difference between the peaks). This delay is caused by the time it takes for the IWTA mechanism to adapt the threshold before significantly causing changes in the spike frequency of the neurons in the hidden layer. The speed at which the threshold changes is a combination of the weights (W^{th}) and decay parameters ($\tau^{t}h$) of the IWTA SNN. Low weights and fast decay parameters result in a slow responding threshold adaptation mechanism, which results in an additional delay in the integration. Moreover, Figure 6.5 visualizes the steady-state error by the presence of a constant offset between the PID and SNN integrator controller.

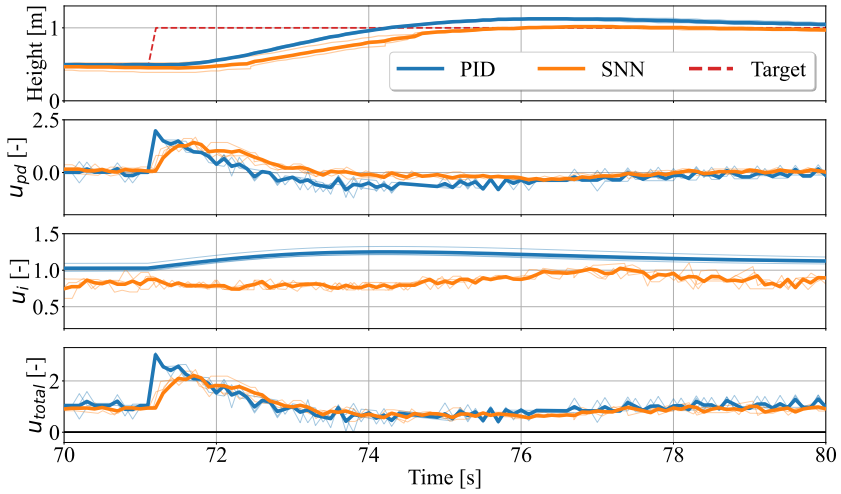


Figure 6.5: Output of the conventional PID and SNN controllers during a 10s interval indicated by **A** in Figure 6.4. The summed output of the PD and integral controller is shown in the lowest plot.

The same observations that have been made for the upward steps are also visible in the controller output for the downward steps of the controller, shown in Figure 6.6. The delay and the constant offset in the integrator are also clearly visible in this figure. However, there is still a big difference between the overshoot compared to the upward steps. The cause of these oscillations is visible in the lowest plot of Figure 6.6, displaying the summed output of the PD and Integral controller. The total output of the controller switches from sign during the transient response of the controllers. A change in the sign of total output results in a change in the orientation of the rotor, initializing a 180-degree rotation of the shaft to which the two rotors are attached. It takes 0.3 seconds to make the 180-degree rotation of the shaft [1]. The delay induced by the shaft rotation also adds some small oscillations to the non-spiking PID controller, shown in section B of Figure 6.4. However, these oscillations are significantly smaller than those present in the SNN Controller. To identify a possible cause for this difference, further examination will focus on the region where the controllers cross the y-axis.

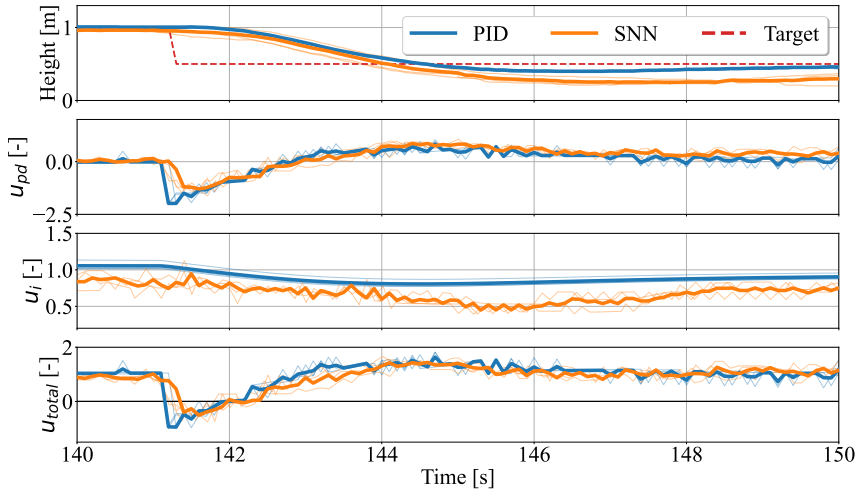


Figure 6.6: Output of the conventional PID and SNN controllers during a 10s time interval indicated by **B** in Figure 6.4. The summed output of the PD and integral controller is shown in the lowest plot.

The further zoomed-in results are presented in Figure 6.7, where each individual run is shown. The

[1] Datasheet Sub-micro Servo - SG51R: https://www.adafruit.com/product/2201

top and bottom plots display the blimp's altitude and the total motor controller command, respectively, within the first two seconds of the step response. Because the SNN controller output is noisier than the non-spiking PID, the motor command switches from sign multiple times. This means that the time window in which the rotors are not pointing in the correct orientation is larger for the spiking controller than for the non-spiking PID controller. This adds an additional delay to the SNN controller, which is most probably the cause of the large overshoot and oscillations during the downward step responses of the negatively buoyant blimp.

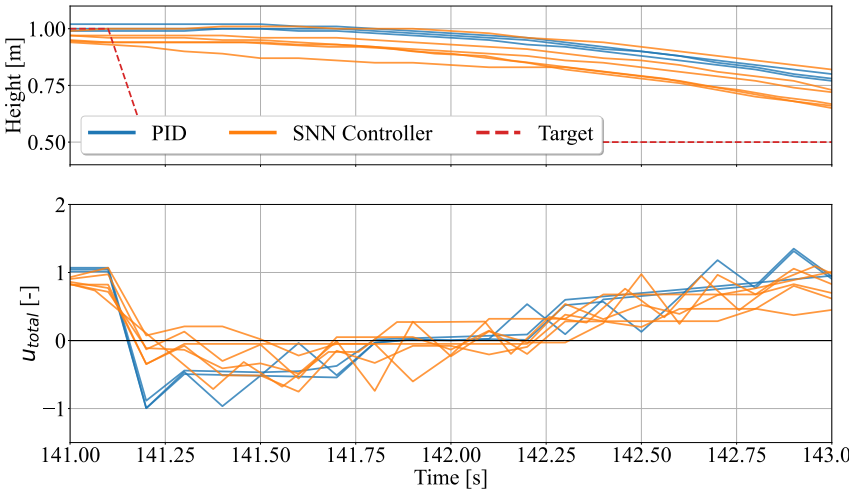


Figure 6.7: Output of the conventional PID and SNN controllers during a 2s time interval indicated by **B** in Figure 6.4. The summed output of the PD and integral controller is shown in the lowest plot.

7

Conclusion & Recommendations

The strict onboard power and weight constraints posed to flying robots create the need for energy-efficient control algorithms. While Spiking Neural Networks (SNNs) hold potential as a solution, the number of practical implementations of SNNs in the control domain is still limited due to the challenges associated with training these types of neural networks. To tackle these challenges and contribute to the existing research on this topic, the following objective was formulated:

> Developing an asynchronous neuromorphic feedback controller for accurate altitude tracking of a real-world blimp by using spiking neural networks, which are trained using a machine learning algorithm to mimic a conventional PID controller.

In order to achieve the research objective, we developed two complementary SNN controllers. Based on the required time constants necessary to model the controller's dynamics, we decided to develop a separate PD (fast τ) and Integrator (slow τ) to facilitate the training process. The SNN's parameters were optimized through an Evolutionary Algorithm (EA), facilitating extensive exploration of the solution space. The SNN consists of three layers of neurons, implemented using the Leaky-Integrate and Fire (LIF) neural model. The exploratory training approach allowed for an analysis of various additional hidden-layer configurations, recurrency and Input Weighted Threshold Adaptation (IWTA), which were based on prior research of SNN PID controllers.

The results presented in this thesis show that the combination of both the spiking PD and Integrator controller is capable of controlling the blimp's altitude to a setpoint in both neutrally and non-neutrally buoyant conditions. The spiking PD controller exhibited a rapid response to control errors while effectively mitigating overshoot and large oscillations. In parallel, the spiking integral controller was designed to minimize steady-state errors arising from the blimp's non-neutral buoyancy-induced drift. This controller learned to perform integration of the error using IWTA within the hidden layer of the network.

The tracking performance of the SNN controller on the neutrally buoyant blimp was almost identical to that of the non-spiking PID controller. However, when employing the SNN controller on the non-neutrally buoyant blimp, there is some overshoot and a small steady-state error ($\pm$ 4cm) present. The overshoot results from the delay introduced by the rotating shaft to which the rotors are attached, while the steady-state error arises due to imperfections in the SNN integrator. The additional decay parameter in the IWTA and the complexity of the solution space of the spiking controller resulted in discrepancies between a perfect integrator and the evolved SNN integrator. The recommendation below will include possible solutions that could minimize this undesired behavior. Despite the limitation within the blimp's current drivetrain and the SNN integrator, the evolved SNN controller has showcased its ability to maintain stable altitude control, employing just 160 spiking neurons.

Recommendations for Future Work

The conclusion above confirms that we have achieved the essential parts of the research objective, although the accuracy of the tracking performance is debatable. However, due to the limited time available for this research, unlimited refinements of the results are not possible. With the insights acquired throughout the

research process, this section compiles all the recommendations for future work regarding this research topic.

The recommendations are categorized into two groups. The first category includes recommendations aimed at enhancing future results using a similar experimental setup as used in this work. The second category focuses on recommendations that were beyond the scope of this research but would be interesting topics for future research. These recommendations are stated below:

Recommendations to Improve the Blimp's Altitude Tracking Performance

1. **SNN Integrator Controller**: The spiking integrator of the SNN controller used IWTA within the network to enable integration. Although the integrator was able to effectively minimize the steady-state error of the blimp, a small residual steady-state error persisted, and a significant delay was evident in the controller's output. Possible solutions that could minimize the delay and/or steady-state error are as follows:

    - **Increase controller frequency**: This reduces the absolute delay of the integrator at the expense of higher power consumption.
    - **Change IWTA implementation**: The threshold of the IWTA-LIF network was simulated as a leaky integrator. The additional decay parameter, in comparison to the implementation in Stroobants, De Wagter, et al. (2023), adds to the delay present in the integrator. Constraining this parameter such that the threshold does not decay could decrease the delay and steady-state error of the integrator.

    It should be noted that, although the delay is not desired when we want to exactly mimic an integrator, some delay could have beneficial effects when dealing with slow system dynamics. A delay in the integrator could prevent the integration from directly building up when receiving a step command, which then minimizes overshoot, as shown during the upward step response of the SNN controller.
2. **Blimp's Hardware**: The shaft mechanism that is used to rotate the orientation of the rotors with 180 degrees, adds a delay to the control system, which causes overshoot and oscillations. To mitigate or entirely eliminate this delay, one can opt for an actuator with a higher rotational speed or replace the rotating shaft completely with a design consisting of an additional set of rotors that point in the opposite direction. Alternatively, a motor controller capable of driving the DC motors in reverse could also solve the problem.
3. **Training Algorithm**: In this research, the primary focus was on utilizing Evolutionary Algorithms for training the SNN to mimic the PD and integral controllers. However, due to the large and complex solution space of the SNN controller, it is very difficult for the training algorithm to find the global minimum. As the training in this work is supervised, one could opt to train the SNNs using error-backpropagation with surrogate gradients (Neftci et al. 2019). However, training SNNs using gradient-based algorithms is challenging due to the sensitivity to initial conditions, exploding/vanishing gradients and the susceptibility to local minima (Bing, Meschede, et al. 2019).

    Nonetheless, there are also more sophisticated Evolutionary Algorithms (EAs) that exhibit promising results in global optimization performance, especially for multimodal solution spaces (Szynkiewicz 2019). One of these algorithms is a variation of the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) used in this work, called Bi-population CMA-ES (Hansen 2009). By maintaining two separate populations, it is more effective in exploring and exploiting complex solution spaces. Utilizing a more advanced EA could enhance the training results of the SNN controllers.

Recommendations to Broaden the Scope of the Research

1. **Neuromorphic Hardware** For this thesis, we used the CPU of the Raspberry Pi Zero W 2 to execute all calculations necessary to run the spiking and non-spiking controllers. To fully harness the advantages of the asynchronous and energy-efficient characteristics of the SNN, the utilization of neuromorphic hardware, such as the Loihi processor (Davies et al. 2018), becomes essential.

    Additionally, to create a completely asynchronous control loop, it is necessary to include event-driven sensors, such as event-based cameras. The combination of neuromorphic sensing and processing is already employed in the field of robotic control, such as for the control of free-flying drones. (Paredes-Vallés et al. 2023).

2. **Analysing Different Neuron Models** There are many different models that can be used to simulate the dynamics of a spiking neuron. The SNNs evolved in this work were based on the Leaky-Integrate and Fire neuron models, chosen for their simplicity and efficiency, thereby demanding minimal computational power. There are more complex and biologically plausible neural models, such as the Izhikevich model, that could be used to create more complex spiking patterns/behavior. One of these interesting characteristics of this model is *frequency adaption*, where a neuron's spiking activity is dependent on the rate of change of the input. This feature could potentially be used to approximate a derivative, for example.

   During the first phase of the thesis, we also examined the use of the more complex Izhikevich neural model. However, the intricate dynamics and the large number of additional network parameters of this neural model posed significant challenges in the training of the SNN. Finding a suitable training method to effectively train SNNs based on more complex neural models could help in the development of a more biologically plausible controller. This could potentially deepen our understanding of the processes occurring in a biological brain.

References

Ang, K. H., G. Chong, and Y. Li (2005). “PID control system analysis, design, and technology”. In: *IEEE Transactions on Control Systems Technology* vol. 13.4, pp. 559–576. DOI: 10.1109/TCST.2005.847331.

Åström, K. and T. Hägglund (1995). *PID Controllers*. Setting the standard for automation. International Society for Measurement and Control. URL: https://books.google.nl/books?id=FsyhngEACAAJ.

Bartolozzi, C., A. Glover, and E. Donati (2022). “Neuromorphic Sensing, Perception and Control for Robotics”. In: *Handbook of Neuroengineering*, pp. 1–31. DOI: 10.1007/978-981-15-2848-4_116-1.

Bellec, G., D. Salaj, A. Subramoney, et al. (2018). “Long short-term memory and learning-to-learn in networks of spiking neurons”. In: *Advances in Neural Information Processing Systems*, pp. 787–797. DOI: https://doi.org/10.48550/arXiv.1803.09574. URL: http://arxiv.org/abs/1803.09574.

Bing, Z., I. Baumann, Z. Jiang, et al. (2019). “Supervised learning in SNN via reward-modulated spike-timing-dependent plasticity for a target reaching vehicle”. In: *Frontiers in Neurorobotics* vol. 13. DOI: 10.3389/fnbot.2019.00018.

Bing, Z., C. Meschede, F. Röhrbein, et al. (2019). “A survey of robotics control based on learning-inspired spiking neural networks”. In: *Frontiers in Neurorobotics* vol. 12. DOI: 10.3389/fnbot.2018.00035.

Bohte, S. M., H. La Poutré, and J. N. Kok (2002). “Error-Backpropagation in Temporally Encoded Networks of Spiking Neurons”. In: *Neurocomputing* vol. 48, pp. 17–37. DOI: https://doi.org/10.1016/S0925-2312(01)00658-0. URL: https://www.sciencedirect.com/science/article/pii/S0925231201006580.

Bohte, S. M. (2011). “Error-Backpropagation in Networks of Fractionally Predictive Spiking Neurons”. In: *Artificial Neural Networks and Machine Learning – ICANN 2011*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 60–68.

Bonilla, L., J. Gautrais, S. Thorpe, et al. (2022). “Analyzing time-to-first-spike coding schemes: A theoretical approach”. In: *Frontiers in Neuroscience* vol. 16. DOI: 10.3389/fnins.2022.971937.

Bouvier, M., A. Valentian, T. Mesquida, et al. (2019). *Spiking neural networks hardware implementations and challenges: A survey*. DOI: 10.1145/3304103.

Clawson, T. S., S. Ferrari, S. B. Fuller, et al. (2016). “Spiking neural network (SNN) control of a flapping insect-scale robot”. In: *2016 IEEE 55th Conference on Decision and Control, CDC 2016*. Institute of Electrical and Electronics Engineers Inc., pp. 3381–3388. DOI: 10.1109/CDC.2016.7798778.

Clawson, T. S., T. C. Stewart, C. Eliasmith, et al. (2018). “An adaptive spiking neural controller for flapping insect-scale robots”. In: *2017 IEEE Symposium Series on Computational Intelligence, SSCI 2017 - Proceedings* vol. 2018-January, pp. 1–7. DOI: 10.1109/SSCI.2017.8285173.

Cohen, I., Y. Huang, J. Chen, et al. (2009). “Pearson correlation coefficient”. In: *Noise reduction in speech processing*, pp. 1–4.

Davies, M., N. Srinivasa, T. H. Lin, et al. (2018). “Loihi: A Neuromorphic Manycore Processor with On-Chip Learning”. In: *IEEE Micro* vol. 38.1, pp. 82–99. DOI: 10.1109/MM.2018.112130359.

DeWolf, T., P. Jaworski, and C. Eliasmith (2020). “Nengo and Low-Power AI Hardware for Robust, Embedded Neurorobotics”. In: *Frontiers in Neurorobotics* vol. 14. DOI: 10.3389/fnbot.2020.568359.

Dupeyroux, J., S. Stroobants, and G. C. De Croon (2022). “A toolbox for neuromorphic perception in robotics”. In: *Proceedings - 2022 8th International Conference on Event-Based Control, Communication, and Signal Processing, EBCCSP 2022*. Institute of Electrical and Electronics Engineers Inc. DOI: 10.1109/EBCCSP56922.2022.9845664.

Eliasmith, C. (2003). “Computation, Representation, and Dynamics in neurobiological systems”. In: *Journal of Chemical Information and Modeling* vol. 53.9, pp. 1–377.

Emran, B. J. and H. Najjaran (2018). “A review of quadrotor: An underactuated mechanical system”. In: *Annual Reviews in Control* vol. 46, pp. 165–180. DOI: 10.1016/j.arcontrol.2018.10.009. URL: https://doi.org/10.1016/j.arcontrol.2018.10.009.

Foderaro, G., C. Henriquez, and S. Ferrari (2010). “Indirect training of a spiking neural network for flight control via spike-timing-dependent synaptic plasticity”. In: *Proceedings of the IEEE Conference on Decision and Control*, pp. 911–917. DOI: 10.1109/CDC.2010.5717260.

Furber, S. B., F. Galluppi, S. Temple, et al. (2014). “The SpiNNaker project”. In: *Proceedings of the IEEE* vol. 102.5, pp. 652–665. DOI: 10.1109/JPROC.2014.2304638.

Gallego, G., T. Delbruck, G. Orchard, et al. (2022). “Event-Based Vision: A Survey”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* vol. 44.1, pp. 154–180. DOI: 10.1109/TPAMI.2020.3008413. URL: http://arxiv.org/abs/1904.08405%20http://dx.doi.org/10.1109/TPAMI.2020.3008413.

Gavrilov, A. V. and K. O. Panchenko (2016). “Methods of learning for spiking neural networks. A survey”. In: *2016 13th International Scientific-Technical Conference on Actual Problems of Electronic Instrument Engineering, APEIE 2016 - Proceedings* vol. 2.November, pp. 455–460. DOI: 10.1109/APEIE.2016.7806372.

Gerstner, W., W. M. Kistler, R. Naud, et al. (2014). *Neural Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press. DOI: 10.1017/CBO9781107447615.

Glatz, S., J. Martel, R. Kreiser, et al. (2019). “Adaptive motor control and learning in a spiking neural network realised on a mixed-signal neuromorphic processor”. In: *Proceedings - IEEE International Conference on Robotics and Automation*. Vol. vol. 2019-May, pp. 9631–9637. DOI: 10.1109/ICRA.2019.8794145. URL: https://inilabs.com/products/pushbot/.

Gonzalez-Alvarez, M., J. Dupeyroux, F. Corradi, et al. (2022). “Evolved neuromorphic radar-based altitude controller for an autonomous open-source blimp”. In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 85–90. DOI: 10.1109/ICRA46639.2022.9812149.

Hansen, N. (2009). “Benchmarking a BI-population CMA-ES on the BBOB-2009 function testbed”. In: *GECCO (Companion)*. DOI: 10.1145/1570256.1570333.

Hebb, D. O. (1949). *The organization of behavior; a neuropsychological theory*.

Heryanto, M., H. Suprijono, B. Yudho, et al. (2017). “Attitude and altitude control of a quadcopter using neural network based direct inverse control scheme”. In: *Advanced Science Letters* vol. 23, pp. 4060–4064. DOI: 10.1166/asl.2017.8328.

Hodgkin, A. and A. Huxley (1952). “A quantitative description of membrane current and its application to conduction and excitation in nerve”. In: *J Physiol*. DOI: 10.1113/jphysiol.1952.sp004764.

Howard, D. and A. Elfes (2014). “Evolving spiking networks for turbulence-tolerant quadrotor control”. In: *Artificial Life 14 - Proceedings of the 14th International Conference on the Synthesis and Simulation of Living Systems, ALIFE 2014*. MIT Press Journals, pp. 431–438. DOI: 10.7551/978-0-262-32621-6-ch071.

Islam, M., M. Okasha, and M. M. Idres (2017). “Dynamics and control of quadcopter using linear model predictive control approach”. In: *IOP Conference Series: Materials Science and Engineering* vol. 270.1. DOI: 10.1088/1757-899X/270/1/012007.

Izhikevich, E. M. (2003). *Simple model of spiking neurons*. DOI: 10.1109/TNN.2003.820440.

— (2004). “Which model to use for cortical spiking neurons?” In: *IEEE Transactions on Neural Networks* vol. 15.5, pp. 1063–1070. DOI: 10.1109/TNN.2004.832719.

— (2007). “Solving the distal reward problem through linkage of STDP and dopamine signaling”. In: *Cerebral Cortex* vol. 17.10, pp. 2443–2452. DOI: 10.1093/cercor/bhl152.

Izhikevich, E. M. and N. S. Desai (2003). “Relating STDP to BCM”. In: *Neural Computation* vol. 15.7, pp. 1511–1523. DOI: 10.1162/089976603321891783.

Jimenez-Fernandez, A., G. Jimenez-Moreno, A. Linares-Barranco, et al. (2012). “A neuro-inspired spike-based PID motor controller for multi-motor robots with low cost FPGAs”. In: *Sensors* vol. 12.4, pp. 3831–3856. DOI: 10.3390/s120403831.

Johansson, R. S. and I. Birznieks (2004). “First spikes in ensembles of human tactile afferents code complex spatial fingertip events”. In: *Nature Neuroscience* vol. 7.2, pp. 170–177. DOI: 10.1038/nn1177.

Joubert, A., B. Belhadj, O. Temam, et al. (2012). “Hardware spiking neurons design: Analog or digital?” In: *Proceedings of the International Joint Conference on Neural Networks*. DOI: 10.1109/IJCNN.2012.6252600.

Juarez-Lora, A., V. H. Ponce-Ponce, H. Sossa, et al. (2022). “R-STDP Spiking Neural Network Architecture for Motion Control on a Changing Friction Joint Robotic Arm”. In: *Frontiers in Neurorobotics* vol. 16.May. DOI: 10.3389/fnbot.2022.904017.

Katoch, S., S. S. Chauhan, and V. Kumar (2021). “A review on genetic algorithm: past, present, and future”. In: *Multimedia Tools and Applications* vol. 80. DOI: https://doi.org/10.1007/s11042-020-10139-6.

Kreiser, R., A. Renner, Y. Sandamirskaya, et al. (2018). “Pose Estimation and Map Formation with Spiking Neural Networks: Towards Neuromorphic SLAM”. In: *IEEE International Conference on Intelligent Robots and Systems* October, pp. 2159–2166. DOI: 10.1109/IROS.2018.8594228.

Lines, J., A. Covelo, R. Gómez, et al. (2017). “Synapse-specific regulation revealed at single synapses is concealed when recording multiple synapses”. In: *Frontiers in Cellular Neuroscience* vol. 11. DOI: 10.3389/fncel.2017.00367.

Lu, H., J. Liu, Y. Luo, et al. (2021). “An autonomous learning mobile robot using biological reward modulate STDP”. In: *Neurocomputing* vol. 458, pp. 308–318. DOI: 10.1016/j.neucom.2021.06.027. URL: https://doi.org/10.1016/j.neucom.2021.06.027.

Maass, W. (1997). “Networks of spiking neurons: The third generation of neural network models”. In: *Neural Networks* vol. 10.9, pp. 1659–1671. DOI: 10.1016/S0893-6080(97)00011-7.

Menouar, H., I. Guvenc, K. Akkaya, et al. (2017). “UAV-enabled intelligent transportation systems for the smart city: Applications and challenges”. In: *IEEE Communications Magazine* vol. 55.3, pp. 22–28. DOI: 10.1109/MCOM.2017.1600238CM.

Merolla, P. A., J. V. Arthur, R. Alvarez-Icaza, et al. (2014). “A million spiking-neuron integrated circuit with a scalable communication network and interface”. In: *Science* vol. 345.6197, pp. 668–673. DOI: 10.1126/science.1254642.

Michalewicz, Z. (1994). “GAs: What Are They?” In: *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 13–30. DOI: 10.1007/978-3-662-07418-3_2. URL: https://doi.org/10.1007/978-3-662-07418-3_2.

Mooney, J. G. and E. N. Johnson (2014). “A Comparison of Automatic Nap-of-the-earth Guidance Strategies for Helicopters”. In: *Journal of Field Robotics* vol. 29.2, pp. 1–17. DOI: 10.1002/rob. URL: http://onlinelibrary.wiley.com/doi/10.1002/rob.21514/abstract.

Moradi, S., N. Qiao, F. Stefanini, et al. (2018). “A Scalable Multicore Architecture with Heterogeneous Memory Structures for Dynamic Neuromorphic Asynchronous Processors (DYNAPs)”. In: *IEEE Transactions on Biomedical Circuits and Systems* vol. 12.1, pp. 106–122. DOI: 10.1109/TBCAS.2017.2759700.

Neckar, A., S. Fok, B. V. Benjamin, et al. (2019). “Braindrop: A Mixed-Signal Neuromorphic Architecture with a Dynamical Systems-Based Programming Model”. In: *Proceedings of the IEEE* vol. 107.1, pp. 144–164. DOI: 10.1109/JPROC.2018.2881432.

Neftci, E. O., H. Mostafa, and F. Zenke (2019). “Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-based optimization to spiking neural networks”. In: *IEEE Signal Processing Magazine* vol. 36.6, pp. 51–63. DOI: 10.1109/MSP.2019.2931595.

Ning, N., K. Huang, and L. Shi (2012). “Artificial neuron with somatic and axonal computation units: Mathematical and neuromorphic models of persistent firing neurons”. In: *Proceedings of the International Joint Conference on Neural Networks* June. DOI: 10.1109/IJCNN.2012.6252428.

Okasha, M., J. Kralev, and M. Islam (2022). “Design and Experimental Comparison of PID, LQR and MPC Stabilizing Controllers for Parrot Mambo Mini□Drone”. In: *Aerospace* vol. 9.6. DOI: 10.3390/aerospace9060298.

Panda, P., S. A. Aketi, and K. Roy (2020). “Toward Scalable, Efficient, and Accurate Deep Spiking Neural Networks With Backward Residual Connections, Stochastic Softmax, and Hybridization”. In: *Frontiers in Neuroscience* vol. 14. DOI: 10.3389/fnins.2020.00653.

Paredes-Vallés, F., J. Hagenaars, J. Dupeyroux, et al. (2023). “Fully neuromorphic vision and control for autonomous drone flight”. In: pp. 1–18. URL: http://arxiv.org/abs/2303.08778.

Pérez, J., J. A. Cabrera, J. J. Castillo, et al. (2018). “Bio-inspired spiking neural network for nonlinear systems control”. In: *Neural Networks* 104, pp. 15–25. DOI: 10.1016/j.neunet.2018.04.002.

Pérez Fernández, J., M. Alcázar Vargas, J. M. Velasco García, et al. (2021). “A biological-like controller using improved spiking neural networks”. In: *Neurocomputing* vol. 463, pp. 237–250. DOI: 10.1016/j.neucom.2021.08.005.

Qiu, H., M. Garratt, D. Howard, et al. (2019). “Evolving Spiking Neural Networks for Nonlinear Control Problems”. In: *Proceedings of the 2018 IEEE Symposium Series on Computational Intelligence, SSCI 2018*, pp. 1367–1373. DOI: 10.1109/SSCI.2018.8628848.

— (2020a). “Towards crossing the reality gap with evolved plastic neurocontrollers”. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pp. 130–138. DOI: https://doi.org/10.1145/3377930.3389843.

— (2020b). “Evolving Spiking Neurocontrollers for UAVs”. In: *2020 IEEE Symposium Series on Computational Intelligence, SSCI 2020* vol. 2, pp. 1928–1935. DOI: 10.1109/SSCI47803.2020.9308275.

Ramezanlou, M. T., V. Azimirad, S. V. Sotubadi, et al. (2020). “Spiking Neural Controller for Autonomous Robot Navigation in Dynamic Environments”. In: *2020 10h International Conference on Computer and Knowledge Engineering, ICCKE 2020*. Institute of Electrical and Electronics Engineers Inc., pp. 544–548. DOI: 10.1109/ICCKE50421.2020.9303687.

Santoso, F., M. A. Garratt, and S. G. Anavatti (2018). “A self-learning TS-fuzzy system based on the C-means clustering technique for controlling the altitude of a hexacopter unmanned aerial vehicle”. In: *Proceeding - ICAMIMIA 2017: International Conference on Advanced Mechatronics, Intelligent Manufacture, and Industrial Automation*, pp. 46–51. DOI: 10.1109/ICAMIMIA.2017.8387555.

Schuman, C., T. E. Potok, R. M. Patton, et al. (2017). “A Survey of Neuromorphic Computing and Neural Networks in Hardware”. In: DOI: https://doi.org/10.48550/arXiv.1705.06963. URL: http://arxiv.org/abs/1705.06963.

Schuman, C., C. Rizzo, J. McDonald-Carmack, et al. (2022). “Evaluating Encoding and Decoding Approaches for Spiking Neuromorphic Systems”. In: *ACM International Conference Proceeding Series*. Association for Computing Machinery. DOI: 10.1145/3546790.3546792.

Shafiee, M., Z. Zhou, L. Mei, et al. (2021). “Unmanned aerial drones for inspection of offshore wind turbines: A mission-critical failure analysis”. In: *Robotics* vol. 10.1, pp. 1–27. DOI: 10.3390/robotics10010026.

Shen, X., X. Lin, and P. De Wilde (2008). “Oscillations and spiking pairs: Behavior of a neuronal model with STDP learning”. In: *Neural Computation* vol. 20.8, pp. 2037–2069. DOI: 10.1162/neco.2008.08-06-317.

Shrestha, A., H. Fang, Z. Mei, et al. (2022). “A survey on neuromorphic computing: Models and hardware”. In: *IEEE Circuits and Systems Magazine* vol. 22.2, pp. 6–35. DOI: 10.1109/MCAS.2022.3166331.

Slijkhuis, F. S., S. W. Keemink, and P. Lanillos (2022). “Closed-form control with spike coding networks”. In: pp. 1–10. DOI: https://doi.org/10.48550/arXiv.2212.12887. URL: http://arxiv.org/abs/2212.12887.

Spüler, M., S. Nagel, and W. Rosenstiel (2015). “A spiking neuronal model learning a motor control task by reinforcement learning and structural synaptic plasticity”. In: *Proceedings of the International Joint Conference on Neural Networks*. Institute of Electrical and Electronics Engineers Inc. DOI: 10.1109/IJCNN.2015.7280521.

Stagsted, R. K., A. Vitale, A. Renner, et al. (2020a). “Event-based PID controller fully realized in neuromorphic hardware: A one DoF study”. In: *IEEE International Conference on Intelligent Robots and Systems*. Vol. vol. 2, pp. 10939–10944. DOI: 10.1109/IROS45743.2020.9340861.

— (2020b). “Towards neuromorphic control: A spiking neural network based PID controller for UAV”. In: *Robotics: Science and Systems XVI*. Vol. vol. 1. Robotics: Science and Systems Foundation. DOI: 10.15607/RSS.2020.XVI.074. URL: http://www.roboticsproceedings.org/rss16/p074.pdf.

Stein, R. B. (1965). “A Theoritical Analysis of Neural Variability”. In: *Biophys*. DOI: 10.1016/s0006-3495(65)86709-1.

Stroobants, S., C. De Wagter, and G. C. H. E. de Croon (2023). “Neuromorphic Control using Input-Weighted Threshold Adaptation”. In: *Proceedings of the 2023 International Conference on Neuromorphic Systems*, pp. 1–8. URL: http://arxiv.org/abs/2304.08778.

Stroobants, S., J. Dupeyroux, and G. De Croon (2022). “Design and implementation of a parsimonious neuromorphic PID for onboard altitude control for MAVs using neuromorphic processors”. In: *Proceedings of the International Conference on Neuromorphic Systems 2022*, pp. 1–7.

Szynkiewicz, P. (2019). “A Comparative Study of PSO and CMA-ES Algorithms on Black-box Optimization Benchmarks”. In: *Journal of Telecommunications and Information Technology* 8. DOI: 10.26636/jtit.2018.127418.

Taherkhani, A., A. Belatreche, Y. Li, et al. (2020). “A review of learning in biologically plausible spiking neural networks”. In: *Neural Networks* vol. 122, pp. 253–272. DOI: 10.1016/j.neunet.2019.09.036.

Tang, G. (2022). “Biologically inspired spiking neural networks for energy-efficient robot learning and control”. PhD thesis. DOI: https://doi.org/doi:10.7282/t3-pqtw-3j28.

Tavanaei, A., M. Ghodrati, S. R. Kheradpisheh, et al. (2019). *Deep learning in spiking neural networks*. DOI: 10.1016/j.neunet.2018.12.002.

Vandesompele, A., F. Walter, and F. Rohrbein (2017). “Neuro-evolution of spiking neural networks on SpiNNaker neuromorphic hardware”. In: *2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016*. DOI: 10.1109/SSCI.2016.7850250.

Vitale, A., A. Renner, C. Nauer, et al. (2021). “Event-driven vision and control for UAVs on a neuromorphic chip”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 103–109.

Wang, S., T. H. Cheng, and M. H. Lim (2022). “A hierarchical taxonomic survey of spiking neural networks”. In: *Memetic Computing* vol. 14.3, pp. 335–354. DOI: 10.1007/s12293-022-00373-w. URL: https://doi.org/10.1007/s12293-022-00373-w.

Webb, A., S. Davies, and D. Lester (2011). “Spiking neural PID controllers”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 7064 LNCS. PART 3, pp. 259–267. DOI: 10.1007/978-3-642-24965-5_28.

Wu, D., X. Yi, and X. Huang (2022). “A Little Energy Goes a Long Way: Build an Energy-Efficient, Accurate Spiking Neural Network From Convolutional Neural Network”. In: *Frontiers in Neuroscience* vol. 16. DOI: 10.3389/fnins.2022.759900.

Yin, B., F. Corradi, and S. M. Bohté (2020). “Effective and Efficient Computation with Multiple-timescale Spiking Recurrent Neural Networks”. In: *ACM International Conference Proceeding Series*. DOI: 10.1145/3407197.3407225. URL: http://arxiv.org/abs/2005.11633.

Zaidel, Y., A. Shalumov, A. Volinski, et al. (2021). “Neuromorphic NEF-Based Inverse Kinematics and PID Control”. In: *Frontiers in Neurorobotics* vol. 15. DOI: 10.3389/fnbot.2021.631159.

Zhang, D., A. Loquercio, X. Wu, et al. (2022). “A Zero-Shot Adaptive Quadcopter Controller”. In: DOI: https://doi.org/10.48550/arXiv.2209.09232. URL: http://arxiv.org/abs/2209.09232.

Zhao, J., E. Donati, and G. Indiveri (2020). “Neuromorphic Implementation of Spiking Relational Neural Network for Motor Control”. In: *Proceedings - 2020 IEEE International Conference on Artificial Intelligence Circuits and Systems, AICAS 2020*. Institute of Electrical and Electronics Engineers Inc., pp. 89–93. DOI: 10.1109/AICAS48895.2020.9073829.

Zulu, A. and S. John (2014). “A Review of Control Algorithms for Autonomous Quadrotors”. In: *Open Journal of Applied Sciences* vol. 4.14, pp. 547–556. DOI: 10.4236/ojapps.2014.414053.