Evolved Neuromorphic Control for High Speed Divergence-based Landings of MAVs

J. J. Hagenaars

Faculty of Aerospace Engineering

TU Delft

Evolved Neuromorphic Control for High Speed Divergence-based Landings of MAVs

by

J. J. Hagenaars

to obtain the degree of Master of Science
at the Delft University of Technology.

Student number:	4297091	
Project duration:	February, 2019 – February, 2020	
Readers:	Dr G. C. H. E. de Croon,	TU Delft, supervisor
	Prof. S. M. Bohte,	CWI
	Dr J. Kober,	TU Delft
	Dr E. van Kampen,	TU Delft
	F. Paredes-Vallés MSc,	TU Delft

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

TU Delft

Acknowledgements

I would like to start off by thanking my supervisors, Guido and Fede, for their invaluable guidance and backing. Guido, I never left your office without a big smile on my face and heaps of inspiration in my pocket. Fede, you were always available for questions, and your patience in answering them was never absent. I look forward to continue to work with you both. A word of appreciation also goes out to Kirk and Sander, who have been a major source of knowledge. Kirk, thank you for the meetings filled with practical tips, and for the foundation that this work is built upon. Sander, I am grateful that you have taken the time to meet with me several times, and I hope that we can continue this fruitful cooperation.

Evelien, thank you for being a never-ceasing source of support, humour and perspective. The past years of my life would have been empty without you.

Dear family and Familie Lups, though I may not always seem as appreciative, I really am thankful that you have been there for me since the beginning, and I cannot do without you all.

Boys of the Upper House, thanks for all the memes, drinks and long days. I would not have enjoyed coming to Delft every day as much if it were not for you. Veel dank ook aan Peter, hij is echt geweldig.

J. J. Hagenaars
Delft, February 2020

Abstract

Flying insects are capable of autonomous vision-based navigation in cluttered environments, reliably avoiding objects through fast and agile manoeuvres. Meanwhile, insect-scale micro air vehicles still lag far behind their biological counterparts, displaying inferior performance at a fraction of the energy efficiency. In light of this, it is in our interest to try and mimic flying insects in terms of their vision-based navigation capabilities, and consequently apply gained knowledge to a manoeuvre of relevance. This thesis does so through evolving spiking neural networks for controlling divergence-based landings of micro air vehicles, while minimising the network's spike rate. We demonstrate vision-based neuromorphic control for a real-world, continuous problem, as well as the feasibility of extending this controller to one that is end-to-end-learnt, and can work with an event-based camera. Furthermore, we provide insight into the resources required for successfully solving the problem of divergence-based landing, showing that high-resolution control can be learnt with only a single spiking neuron. Finally, we look at evolving only a subset of the spiking neural network's available hyperparameters, suggesting that the best results are obtained when all parameters are affected by the learning process.

Contents

List of Symbols

Math symbols

$\langle \cdot \rangle$	average of a stochastic variable
$\rightarrow$	approach
$\approx$	approximately equal
$\leftarrow$	assignment
$\doteq$	equality that is true by definition
$\exp(\cdot)$	exponential function
$\in$	is an element of
∞	to infinity and beyond
∇	nabla operator
$\neq$	inequality
$\Pr\{\cdot\}$	probability of an event occurring
$\dot{a}$	derivative of a w.r.t. time t
$\max_a$, $\arg\max_a f(a)$	(argument a of) maximum value of function $f(a)$
$\mathbf{a} \cdot \mathbf{b}$	dot product of vectors $\mathbf{a}$ and $\mathbf{b}$
$(a, b]$	interval that includes b but not a
$A \sim b$	sample random variable A from distribution b
$\mathrm{Cov}[\cdot]$	covariance of a stochastic variable
$\mathbb{E}[\cdot]$	expectation of a stochastic variable
$\mathbb{R}$	set of real numbers
Δ	difference operator
δ	Dirac delta function

Greek symbols

Δt	simulation time step duration
$\Delta\theta$	width of stochastic neural firing threshold region
α	learning rate/step size
β	scaling constant
γ	discount rate
δ	temporal-difference error
ε	probability of taking a non-greedy action
$\varepsilon(t)$	kernel describing presynaptic spike contributions for time-since-spike t
$\eta(t)$	kernel describing neural membrane afterpotential for time-since-spike t
$\boldsymbol{\theta}$	policy parameter vector
θ	neural firing threshold
$\kappa(t)$	kernel describing external current contributions for time-since-spike t
λ	eligibility trace decay rate
ν	neural activity
$\boldsymbol{\pi}$, $\pi(a \mid s)$, $\pi(s)$	policy (vector), stochastic or deterministic
$\hat{\rho}$	discrete version of instantaneous neural firing rate ρ
ρ	stochastic intensity of point process denoting instantaneous neural firing rate
σ, $\boldsymbol{\sigma}$	width/standard deviation (vector)
τ	time-to-contact
τ_a	time constant of decay of variable a

ω_x, ω_y	ventral flow components along X, Y

Latin symbols

A, a	action
A_+, A_-	magnitude of synaptic change due to long-term potentiation or depression
$\mathcal{A}(s)$	set of all actions available in state s
D	flow field divergence
e, $\mathbf{e}$	eligibility trace (vector)
P_j^{pre}, P_i^{post}	tracking variables for pre- and postsynaptic spike trace
u_{rest}	neural membrane potential in rest
f	focal length
g	gravitational acceleration
G	return
h	altitude above a surface
H	update target
$I(t)$	neural input current
$I(x, y, t)$	image intensity function
J	performance function
m	mass
N	amount of …
n	episode number
O, o	origin of a coordinate system
P	event polarity
p, q, r	rotational rates around X, Y, Z
q_π	action-value function following policy π
$\bar{R}$	mean reward
$\mathcal{R}$	set of all rewards
R, r	reward
$\mathcal{S}$	set of all states
$S(R)$	success signal as a monotonic function of reward R
S, s, $\mathbf{s}$	state (vector)
$\hat{t}$	time of previous neural firing
T	thrust
t	time
u	neural membrane potential
U	unsupervised part of reward-modulated learning rules
u, v	optical flow components
U, V, W	translational velocities along X, Y, Z
u_0, v_0, w_0	optical flow visual observables
V, Q	estimate of state-value and action-value function
$\hat{v}$, $\hat{q}$	parametrised estimate of state-value and action-value function
v_π	state-value function following policy π
W	weight
w, $\mathbf{w}$	weight or synaptic efficacy (vector)
X	presynaptic spike train
x, y	position on the image plane/in the pixel array
X, Y, Z	axes of the Cartesian coordinate system (metric position)
x_j, y_i	binary variables indicating a pre- or postsynaptic spike
Y	postsynaptic spike train

Sub- and superscripts

$*$	optimality
$+$, $-$	long-term potentiation and depression subscripts
f	neural firing index
h	actor neuron index

i	postsynaptic neuron index
j	presynaptic neuron index
k	place cell index
n	step index
pre, *post*	pre- and postsynaptic contribution subscripts
T	final time step of an episode
t	time step index

List of Abbreviations

A

AER	address-event representation
ANNarchy	Artificial Neural Networks Architect
API	application programming interface
ATIS	Asynchronous Time-based Image Sensor

B

BCM	Bienenstock-Cooper-Munro

C

CNN	convolutional neural network
CPU	central processing unit
CUDA	Compute Unified Device Architecture

D

DAVIS	Dynamic and Active-pixel Vision Sensor
DDPG	deep deterministic policy gradient
DP	dynamic programming
DQN	deep Q-network
DVS	Dynamic Vision Sensor

E

eDVS	Embedded Dynamic Vision Sensor

F

FoC	focus of contraction
FoE	focus of expansion

G

GPS	global positioning system

H

HJB	Hamilton-Jacobi-Bellman
HPC	hippocampal place cell

I

IMU	inertial measurement unit

L

LIF	leaky integrate-and-fire
LQR	linear quadratic regulator
LTD	long-term depression

LTP long-term potentiation

M

MDP Markov decision process
meDVS Miniature Embedded Dynamic Vision Sensor
MNIST modified National Institute of Standards and Technology

N

NEST Neural Simulation Tool
N-MNIST neuromorphic modified National Institute of Standards and Technology

O

ODE ordinary differential equation

P

PPO proximal policy optimisation
PSP postsynaptic potential

R

RPE reward prediction error

S

SLAYER spike layer error reassignment
SpiNNaker Spiking Neural Network Architecture
SRM spike response model
SRV stochastic real-valued

T

TD temporal-difference
TD-LTP temporal-difference-based long term potentiation
TD-STDP temporal-difference-based spike-timing-dependent plasticity
TLU threshold logic unit
TTC time-to-contact

U

UCB upper-confidence bound

V

VLSI very-large-scale integration

X

XOR exclusive OR

List of Figures

List of Tables

List of Listings

1

Introduction

Flying insects are everything we would like micro air vehicles (MAVs) to be: units that can navigate autonomously in cluttered environments through fast and agile manoeuvres, while being energy efficient and reliable with respect to vision-based obstacle avoidance. Though insect-scale MAVs like the DelFly (de Croon, de Clercq, Ruijsink, Remes & de Wagter, 2009; Karásek, Muijres, Wagter, Remes & de Croon, 2018) or the RoboBee (Ma, Chirarattananon, Fuller & Wood, 2013) have already been developed, these are still a far cry from achieving parity regarding efficient and intelligent flight. In light of this, it is in our interest to try and mimic flying insects in terms of their vision-based navigation capabilities, and consequently apply gained knowledge to a manoeuvre of relevance, such as landing.

Like all animals that can see, insects rely on patterns of visual motion, or *optical flow* (Gibson, 1950), for many important behaviours. During landing, for instance, honeybees have been observed to maintain a constant rate of expansion, or *divergence*, of the optical flow field to ensure a smooth approach (Baird, Boeddeker, Ibbotson & Srinivasan, 2013). Light-sensitive cells and neurons in the retinas of these animals allow them to perceive motion in a *spike-based* manner, with the neurons giving off electrical pulses in response to different levels of stimulation by the cells sensitive to brightness changes (Posch, Serrano-Gotarredona, Linares-Barranco & Delbruck, 2014). Underlying networks of interconnected neurons use the resulting temporal sequences of discrete *spikes* to come up with a motion estimate. The sparsity and asynchronicity of this process make it very energy efficient, and a great candidate for on-board optical flow estimation.

The desirable qualities of such an end-to-end spike-driven approach have inspired researchers to come up with artificial substitutes, which we call *neuromorphic*, for the components of the biological motion estimation process. *Event-based* cameras (Gallego et al., 2019; Posch et al., 2014), whose pixels asynchronously register brightness changes as *events*, take the place of the retina. *Spiking neural networks* (SNNs, Maass, 1997) assume the role of the underlying networks, subsequently transforming these event sequences into an estimate of visual motion. Analogous to these networks of biological neurons, SNNs carry out computations in a sparse and asynchronous manner, making them a natural fit for the sparse, temporal data generated by the event-based camera (Orchard & Etienne-Cummings, 2014; Roy, Jaiswal & Panda, 2019). This asynchronicity also implies that SNNs are inherently more energy efficient than the conventional artificial neural networks (ANNs) that operate synchronously, given that each spike costs a certain amount of energy (Tavanaei, Ghodrati, Kheradpisheh, Masquelier & Maida, 2019). This advantage carries over to hardware implementations of SNNs (Bouvier et al., 2019), meaning that it is possible to run large SNNs at an order of magnitude lower power consumption than comparable ANNs (Pfeiffer & Pfeil, 2018).

Despite these benefits, SNNs have not yet become widespread in vision-based robot control applications. The cause of this may lie partially in the difficulty of training SNNs: the discrete spiking nature of these networks severely limits the application of the gradient-based optimisation algorithms that work so well for ANNs. Instead, most learning is based on the relative timing of spikes (Caporale & Dan, 2008), often in combination with a surrogate gradient (Bohte, Kok & La Poutré, 2002; Shrestha & Orchard,

2018) or global reward signal (Florian, 2007; Frémaux, Sprekeler & Gerstner, 2013; Vasilaki, Frémaux, Urbanczik, Senn & Gerstner, 2009) to allow the specification of desired behaviour or goals. As far as vision-based robot control is concerned, these learning rules currently seem to be limited to simulated applications (Bing, Meschede, Chen, Knoll & Huang, 2020; Clawson, Ferrari, Fuller & Wood, 2016) or discrete problems (Zhao, Zeng & Xu, 2018), with no real-world, continuous control implementations that the authors are aware of. This leads us to believe that the available learning methods are, as of this moment, not mature enough for solving such complex tasks.

ANNs, on the other hand, have been employed successfully for real-world, vision-based continuous control. For instance, Scheper and de Croon (2020) used an *evolutionary* algorithm to optimise ANNs for performing divergence-based landings of MAVs. Solving the same control problem, but replacing the ANNs with SNNs, this work aims to demonstrate that SNNs can in fact already be taught to perform vision-based, continuous control in a real-world environment through the use of evolutionary learning. When applied to neural networks, this kind of learning is called *neuroevolution* (Floreano, Dürr & Mattiussi, 2008). The generality of the evolutionary framework, both in terms of suitable problems (Bäck, Fogel & Michalewicz, 1997) as well as the structure and characteristics of the evolved individuals (Fogel, 1997), adds to the strength of this approach.

As was argued at the beginning of this introduction, the authors believe it is in our interest to mimic nature regarding the vision system that gives flying insects their navigation abilities. To progress towards this goal, we aim to demonstrate here that SNNs are capable of controlling a real-world MAV landing when given an estimate of divergence. However, the biological accuracy of the process of estimating divergence is left largely unregarded. To still make a contribution on this front, we will demonstrate the feasibility of an interface with the work of Paredes-Vallés, Scheper and de Croon (2019), where an SNN is trained to extract optical flow quantities (including divergence) from event-based camera data. The combination of their work with ours will then result in an end-to-end-learnt, spike-driven pipeline that closely resembles its biological counterpart.

1.1 Motivation and research question

The aim of this thesis is to come up with an end-to-end trainable and bio-inspired system that is capable of vision-based navigation through event-based optical flow control. The motivation behind the pursuit of such a fully trainable, bio-inspired system is twofold: 1) the fact that the system is comprised of only SNNs processing event-based data means that it can be implemented on low-power neuromorphic hardware such as Intel's Loihi (Davies et al., 2018), which would in theory allow on-board optical flow control on insect-scale MAVs such as the DelFly (de Croon et al., 2009; Karásek et al., 2018), and 2) investigating the learning capacity of SNNs regarding complex problems like real-world control. Summarising the above in a research objective:

Achieving an end-to-end trainable, neuromorphic system capable of learning event-based optical flow control in the real world.

Clearly, this objective consists of several parts. An SNN will have to be configured properly for it to process the optical flow inputs optimally, while for the network to be able to run on board an MAV it should not be too computationally demanding. In order for the system to be trainable an evolutionary framework well-suited for this control problem has to be selected, in close correspondence with the appropriate objective functions and evolutionary mechanisms. Finally, an interface with the work of Paredes-Vallés et al. (2019) should be demonstrated to show the system's compatibility with event-based camera inputs. This leads to the following research question:

Can a spiking neural network be trained, using a neuroevolutionary framework, to map event-based optical flow visual observables to control inputs to a micro aerial vehicle, with the goal of letting it perform real-world landings?

Which, according to the above-mentioned parts, can be split up into sub-questions as follows:

- **What is the optimal configuration of the spiking neural network, and can this be run on board the micro aerial vehicle?**
- **Given the real-world control task, what is the optimal configuration of the neuroevolutionary framework?**
- **Can the spiking neural network, being trained in simulation, be used on board a micro aerial vehicle to perform real-world landings using conventional optical flow control?**
- **What constitutes an interface with the work of Paredes-Vallés et al. (2019), and does the implementation of such an interface affect the performance of the spiking neural network regarding its task of controlling optical-flow-based landings?**

1.2 Structure of this work

This thesis consists of four parts. Part I contains a standalone scientific paper that presents the main contributions of this thesis.

Part II gives an in-depth account of relevant literature on the topics of (event-based) optical flow control, reinforcement learning and reward-modulated neuromorphic computing and its use in vision-based navigation. In Chapter 2, optical flow is introduced along with conventional estimation methods, and its application in MAV navigation is discussed. Next, Chapter 3 quickly summarises current event-based cameras and event-based optical flow estimation methods, after which Chapter 4 covers the concept of reinforcement learning and its presence in organisms and robot control. Chapter 5 introduces spiking neural networks, goes over their learning rules and applications in soft- and hardware, and focuses on their use for vision-based navigation and MAV control. Finally, Chapter 6 synthesises the provided literature.

Next, Part III discusses the preliminary experiments performed to gain a deeper understanding of the topic. These experiments consist of testing the performance of various reward-modulated learning rules in a simple vertical simulator. Chapter 7 documents the simulation environment and settings in detail. In Chapter 8, the implementation of the various learning rules is explained, and their performance is evaluated and compared across numerous parameter settings. Chapter 9 discusses the outcomes of the experiments, and tries to provide guidance on which learning rule is most promising for the task at hand. Furthermore, it covers the relation between these preliminary experiments and the final paper presented in Part I.

Finally, Part IV contains the appendices, of which there is only one: Appendix A lists the default parameters for all simulation cases described in Chapters 7 and 8.

I

Scientific Paper

Evolved Neuromorphic Control for High Speed Divergence-based Landings of MAVs

J. J. Hagenaars*‡, F. Paredes-Vallés†‡, S. M. Bohte§, G. C. H. E. de Croon†‡

‡*Micro Air Vehicle Laboratory*
Delft University of Technology
Delft, The Netherlands

§*Centrum Wiskunde & Informatica*
Amsterdam, The Netherlands

***Abstract*—Flying insects are capable of autonomous vision-based navigation in cluttered environments, reliably avoiding objects through fast and agile manoeuvres. Meanwhile, insect-scale micro air vehicles still lag far behind their biological counterparts, displaying inferior performance at a fraction of the energy efficiency. In light of this, it is in our interest to try and mimic flying insects in terms of their vision-based navigation capabilities, and consequently apply gained knowledge to a manoeuvre of relevance. This paper does so through evolving spiking neural networks for controlling divergence-based landings of micro air vehicles, while minimising the network's spike rate. We demonstrate vision-based neuromorphic control for a real-world, continuous problem, as well as the feasibility of extending this controller to one that is end-to-end-learnt, and can work with an event-based camera. Furthermore, we provide insight into the resources required for successfully solving the problem of divergence-based landing, showing that high-resolution control can be learnt with only a single spiking neuron. Finally, we look at evolving only a subset of the spiking neural network's available hyperparameters, suggesting that the best results are obtained when all parameters are affected by the learning process.**

***Index Terms*—spiking neural networks, optical flow, micro air vehicles, neuroevolution**

I. Introduction

Flying insects are everything we would like micro air vehicles (MAVs) to be: units that can navigate autonomously in cluttered environments through fast and agile manoeuvres, while being energy efficient and reliable with respect to vision-based obstacle avoidance. Though insect-scale MAVs like the DelFly [1], [2] or the RoboBee [3] have already been developed, these are still a far cry from achieving parity regarding efficient and intelligent flight. In light of this, it is in our interest to try and mimic flying insects in terms of their vision-based navigation capabilities, and consequently apply gained knowledge to a manoeuvre of relevance, such as landing.

*MSc student, †supervisor. This work includes several software packages: 1) the evolutionary framework: `https://github.com/Huizerd/evolutionary`; 2) a vertical simulation environment: `https://github.com/Huizerd/gym-quad`; 3) an SNN library in C: `https://github.com/Huizerd/tinysnn`; 4) the Paparazzi autopilot software: `https://github.com/Huizerd/paparazzi`. Videos of the flight tests can be found on `https://www.youtube.com/playlist?list=PL_KSX9GOn2P9wfgUNIR_FVbx3FoXBOK68`.

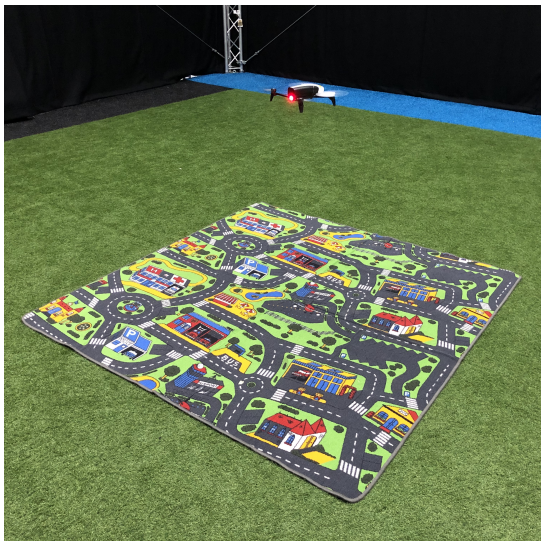


Figure 1. Parrot Bebop 2 quadrotor MAV.

Like all animals that can see, insects rely on patterns of visual motion, or *optical flow* [4], for many important behaviours. During landing, for instance, honeybees have been observed to maintain a constant rate of expansion, or *divergence*, of the optical flow field to ensure a smooth approach [5]. Light-sensitive cells and neurons in the retinas of these animals allow them to perceive motion in a *spike-based* manner, with the neurons giving off electrical pulses in response to different levels of stimulation by the cells sensitive to brightness changes [6]. Underlying networks of interconnected neurons use the resulting temporal sequences of discrete *spikes* to come up with a motion estimate. The sparsity and asynchronicity of this process make it very energy efficient, and a great candidate for on-board optical flow estimation.

The desirable qualities of such an end-to-end spike-driven approach have inspired researchers to come up with artificial substitutes, which we call *neuromorphic*, for the components of the biological motion estimation process. *Event-based* cam-

eras [6], [7], whose pixels asynchronously register brightness changes as *events*, take the place of the retina. *Spiking neural networks* (SNNs) [8] assume the role of the underlying networks, subsequently transforming these event sequences into an estimate of visual motion. Analogous to these networks of biological neurons, SNNs carry out computations in a sparse and asynchronous manner, making them a natural fit for the sparse, temporal data generated by the event-based camera [9], [10]. This asynchronicity also implies that SNNs are inherently more energy efficient than the conventional artificial neural networks (ANNs) that operate synchronously, given that each spike costs a certain amount of energy [11]. This advantage carries over to hardware implementations of SNNs [12], meaning that it is possible to run large SNNs at an order of magnitude lower power consumption than comparable ANNs [13].

Despite these benefits, SNNs have not yet become widespread in vision-based robot control applications. The cause of this may lie partially in the difficulty of training SNNs: the discrete spiking nature of these networks severely limits the application of the gradient-based optimisation algorithms that work so well for ANNs. Instead, most learning is based on the relative timing of spikes [14], often in combination with a surrogate gradient [15], [16] or global reward signal [17]–[19] to allow the specification of desired behaviour or goals. As far as vision-based robot control is concerned, these learning rules currently seem to be limited to simulated applications [20], [21] or discrete problems [22], with no real-world, continuous control implementations that the authors are aware of. This leads us to believe that the available learning methods are, as of this moment, not mature enough for solving such complex tasks.

ANNs, on the other hand, have been employed successfully for real-world, vision-based continuous control. For instance, [23] used an *evolutionary* algorithm to optimise ANNs for performing divergence-based landings of MAVs. Solving the same control problem, but replacing the ANNs with SNNs, this work aims to demonstrate that SNNs can in fact already be taught to perform vision-based, continuous control in a real-world environment through the use of evolutionary learning. When applied to neural networks, this kind of learning is called *neuroevolution* [24]. The generality of the evolutionary framework, both in terms of suitable problems [25] as well as the structure and characteristics of the evolved individuals [26], adds to the strength of this approach.

As was argued at the beginning of this introduction, the authors believe it is in our interest to mimic nature regarding the vision system that gives flying insects their navigation abilities. To progress towards this goal, we aim to demonstrate here that SNNs are capable of controlling a real-world MAV landing when given an estimate of divergence. However, the biological accuracy of the process of estimating divergence is left largely unregarded. To still make a contribution on this front, we will demonstrate the feasibility of an interface with the work of [27], where an SNN is trained to extract optical flow quantities (including divergence) from event-based camera data. The combination of their work with ours will then result in an end-to-end-learnt, spike-driven pipeline that closely resembles its biological counterpart.

This paper contains *three* main contributions. First, it demonstrates learnt neuromorphic control based on conventional optical flow for the real-world, continuous problem of optical flow landing, and argues that this can be extended to end-to-end-learnt control and event-based optical flow with the work of [27]. Second, it explores the concept of energy-minimising neuroevolution, and shows its positive influence on the resolution and smoothness of control. Third, it investigates the effect the number of hidden neurons and the set of evolving hyperparameters on performance on the divergence-based landing task, providing insight into the difficulty of the learning problem and the room energy-minimising neuroevolution leaves for pruning neurons.

The remainder of this paper is structured as follows. Section II provides related work concerning divergence-based landing, neuromorphic controllers, neuroevolution for robot control and the reality gap between simulation and the real world. The control problem, simulation environment, SNN configuration and neuroevolution procedure are discussed in Section III, while Section IV goes over the set-up of the performed experiments and lists their findings. Conclusions drawn from these findings are stated in Section V. Appendices A to E follow after that.

II. Related Work

A. Divergence-based Landing

Insects make use of a plethora of navigation strategies based on *visual observables* [28]. One of these observables is the flow-field divergence D, which quantifies the flow's rate of expansion and in physical terms corresponds to the ratio of vertical velocity and height. Divergence was found to be kept constant by honeybees when landing on vertical surfaces [5], a strategy which has subsequently been employed for optical-flow-based, vertical landings of MAVs, using a simple proportional controller in combination with a conventional camera [29]–[31] or an event-based cameras [23], [32]. Comparing these camera types, [23] showed a decreased divergence estimation error for the event-based variant. Additionally, the lower latency of these cameras makes them especially suitable for high-speed landings, as was confirmed by [32]. Despite these benefits, however, this work will make use of a conventional, frame-based camera to limit set-up complexity.

In the presence of measurement noise, delay and environmental disturbances (as is the case during real-world tests), the non-linear relation between control and sensing leads to vertical oscillations at a certain height above the surface for a certain fixed proportional control gain [33]. Delaying the onset of these *self-induced* oscillations can be done through gain adaptation [30], [31] or by responding asymmetrically, i.e., differently for increasing and decreasing divergence values [23].

B. Neuromorphic Robot Controllers

Neuromorphic control, meaning control using SNNs, can either be learnt or hand-engineered. Whereas the latter approach already lends itself for more sophisticated, real-world control tasks, fully learnt controllers have so far only been implemented for simpler or simulated problems. For instance, [34]–[36] already demonstrate real-world vision-based control of a wheeled robot with a neuromorphic processor, but their SNNs are (largely) made up of hand-designed and hand-tuned neuronal populations and connections. On the other hand, in [22], a reward-modulated learning rule is used to learn the synaptic weights of most connections. However, while the task of learning MAV obstacle avoidance presented there might seem similar to the landing problem we consider here, it is in fact converted to a much simpler (almost one-to-one) mapping between discrete inputs and outputs through preprocessing.

Regarding simulation, the authors of [21] show fully learnt neuromorphic control based on vision for the lane-keeping task developed in [37]. Events coming from an event-based camera excite a population of Poisson neurons, whose spikes are fed to a two-neuron SNN that outputs motor speeds for a two-wheeled robot. Although all synaptic weights are learnt through reward-modulated learning, the lane-keeping task is set up in such a way that its complexity remains limited: rewards are tailored to each individual neuron, so that increased firing inevitably results in a self-centering policy, independent of incoming Poisson spikes. It remains to be seen how well the reward-modulated learning rule would hold up for a truly global reward, or in the presence of real-world-like amounts of noise. In [20], the authors employ the same learning rule and a three-layer SNN for learning flight control of a simulated robotic insect. Despite seeming more difficult at first sight, reward is based on the deviation from the flight trajectory generated by a linear quadratic regulator (LQR), making it essentially a lane-keeping task similar to [21]. Still, a global reward is used, making this work of interest for further research. More examples of learnt neuromorphic robot control can be found in [38].

C. Neuroevolution for Control

Reviews of the neuroevolution field show the feasibility of using evolution for learning in ANNs [39] and SNNs [24]. Although many regarded the evolutionary approach to only be tractable for smaller networks, recent work has demonstrated this to be false, with over four million parameters being evolved to produce successful game-playing agents [40]. Apart from also scaling well in terms of compute (since individuals in a population can be evaluated in parallel), neuroevolution proved to often be more sample efficient than RL-based learning approaches [39]. Additionally, the population-based framework of evolution inherently promotes behavioural diversity in case of multiple objectives [39] and can be used for the optimisation of all parameters of a network, including learning rule characteristics [41].

Most recent control applications of evolution focus on ANN-based game playing or other simulated tasks [40], [42], [43]. More relevant, however, is the field of evolutionary robotics, which encompasses neuroevolution for real-world control. For instance, [23] demonstrate real-world event-based optical flow control of a landing MAV, where the ANN controller is evolved offline. It was shown that a small network of three layers (eight hidden neurons) was sufficient to perform continuous control, with only the weights being evolved. In [44], [45], neuroevolution of ANNs is demonstrated for real-world MAV control in the horizontal plane.

The number of recent real-world applications involving neuroevolution for SNN controllers seems to be limited. The authors of [46], [47] evolve SNNs for the control of a two-wheeled robot based on basic vision (e.g., light sensors). On the other hand, [48] learns a walking task for a six-legged robot offline, and then transfers the learnt policy to the real world. There seems to little consensus among these works about the to-be-evolved parameters: some mainly evolve the topology of the network, while others evolve only the SNN's weights and constants. The same can be seen for more recent applications of neuroevolution to SNNs for control in simulation. For example, [49] evolves SNN weights and neuron types, while [50], [51] follow the NEAT [52] approach, where either a connection or neuron is added/removed, or a connection weight is changed.

Another promising approach is the combination of learning and evolution, where offline evolution evolves the parameters of both network and learning rule, and online learning is used to deal with changing situations during operation. Several works demonstrate this combination for SNNs. Most recently, [53] demonstrated the successful classification of event-based camera data using an evolved SNN. However, like [54], the addition of online learning based on relative spike times did not seem to improve performance in general. Reward-modulated learning rules could prove to be a better option, something which is hinted at by the results of [55], where evolved ANNs that include modulatory neurons outperform those that do not. Still, more research is needed in this direction before a definitive conclusion can be drawn.

D. Crossing the Reality Gap

In evolutionary robotics, the *reality gap* is the discrepancy between simulation and the real world, for instance because agents exploit simulator inaccuracies, or because they cannot deal with real-world noise [56], [57]. Multiple ways of bridging this gap have been proposed [58], [59], of which one is the addition of all kinds of noise during simulation [60], and another is the varying of simulation parameters [61]. The authors of [62] adopt this approach for the evolution of divergence-based landing controllers for MAVs, varying delay, computational jitter, (proportional) sensor noise, thrust responsiveness and simulation frequency.

A very similar approach, which has recently surfaced in the field of deep learning, is the concept of *domain randomisation* [63]. Mainly applied to vision-based control, the

randomisation of the domain consists of randomly varying the textures, colours and shapes of relevant objects, as well as adding distracting, non-relevant objects. In this way, learnt agents are supposed to develop robustness to (irrelevant) changes in the environment, improving their transferability to the real world. The success of this approach has been demonstrated by [64] for vision-based navigation of an MAV through gates in various surroundings, and by [65] for car detection on the KITTI dataset.

III. Methodology

A. Landing with Optical Flow

The optical flow formulation used here originates from [66] and has subsequently been employed by many works on optical-flow-based landing [23], [30]–[33]. It starts from a downward-facing camera above a static planar scene, as depicted in Fig. 2. If the camera were to move, this ego-motion would cause a perceived optical flow, which can be summarised with the help of visual observables.

Consider the two reference frames in Fig. 2: the inertial world frame $\mathcal{W}$ and the moving camera frame $\mathcal{C}$, of which the latter is centred at the camera's focal point. Position in both frames is given by the coordinates (X, Y, Z), with (U, V, W) denoting the translational velocities along the respective axes. The Euler angles ϕ, θ and ψ describe the attitude of $\mathcal{C}$ with respect to $\mathcal{W}$ in terms of roll, pitch and yaw, and p, q and r give the corresponding rotational rates.

Starting from the pinhole camera model [67], we can relate camera ego-motion to optical flow, and subsequently to the visual observables. Again looking at Fig. 2, translational and rotational movements of the camera cause a point (x, y) to move across the field-of-view with optical flow components (u, v), leading to the following expression:

$$\begin{aligned} u &= -\frac{U_\mathcal{C}}{Z_\mathcal{C}} + \frac{W_\mathcal{C}}{Z_\mathcal{C}}x - q + ry + pxy - qx^2 \\ v &= -\frac{V_\mathcal{C}}{Z_\mathcal{C}} + \frac{W_\mathcal{C}}{Z_\mathcal{C}}y + p - rx - qxy + py^2 \end{aligned} \tag{1}$$

The structure of Eq. (1) implies that the optical flow of a point can be decomposed into a translational and a rotational component. With knowledge of the camera's rotational rates, the optical flow can be derotated, as is common in, e.g., MAV-based applications [23], [30]–[33], [66], leaving us with the purely translational optical flow components (u_T, v_T). If we furthermore consider the scene to be planar, the depth $Z_\mathcal{C}$ of all points in the camera's field-of-view can be described by:

$$Z_\mathcal{C} = Z_0 + \alpha X_\mathcal{C} + \beta Y_\mathcal{C} \tag{2}$$

with Z_0 the distance to the surface along $Z_\mathcal{C}$, and α and β the slopes of the planar scene relative to $X_\mathcal{C}$ and $Y_\mathcal{C}$, respectively [66]. Knowing that the projection of each point in $\mathcal{C}$ onto the camera's image plane can be expressed as $(x, y) = (X_\mathcal{C}/Z_\mathcal{C}, Y_\mathcal{C}/Z_\mathcal{C})$, we can rewrite Eq. (2):

$$\frac{Z_\mathcal{C} - Z_0}{Z_\mathcal{C}} = \alpha x + \beta y \tag{3}$$

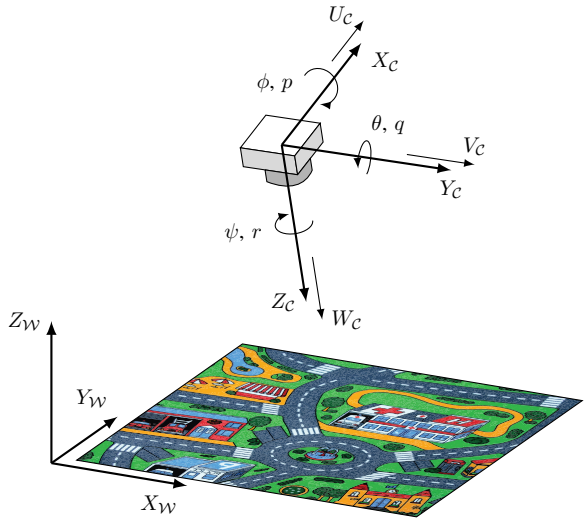


Figure 2. Definition of the inertial world frame $\mathcal{W}$ and the moving camera frame $\mathcal{C}$, as well as their respective translational and rotational velocities and Euler angles. Adapted from [68].

Finally, by substituting Eq. (3) together with the camera's depth-scaled velocities $(\vartheta_x, \vartheta_y, \vartheta_z) = (U_\mathcal{C}/Z_0, V_\mathcal{C}/Z_0, W_\mathcal{C}/Z_0)$ into Eq. (1), we end up with the following expressions for translational optical flow [66]:

$$\begin{aligned} u_T &= (-\vartheta_x + \vartheta_z x)(1 - \alpha x - \beta y) \\ v_T &= (-\vartheta_y + \vartheta_z y)(1 - \alpha x - \beta y) \end{aligned} \tag{4}$$

As is apparent from Eq. (4), the depth-scaled velocities can be derived from the translational optical flow of several points in the camera's field-of-view. The *ventral flows* and *divergence*, which make up the visual observables and which quantify the average flows along $X_\mathcal{C}$ and $Y_\mathcal{C}$ and the divergence of the flow field, are then defined as [66]:

$$\omega_x = -\vartheta_x \qquad \omega_y = -\vartheta_y \qquad D = 2\vartheta_z \tag{5}$$

Following [31] and the subsequent implementations by [23], [32], divergence can be estimated through the relative, temporal variation in the distance between any two image points. Referred to as *size divergence* D_t, this method results in a reliable estimate of divergence $\hat{D}$ when averaged over a set of N_D pairs of points:

$$\begin{aligned} D_s(t) &= \frac{2}{\Delta t}\frac{l(t-\Delta t) - l(t)}{l(t-\Delta t)} \\ \hat{D}(t) &= \frac{1}{N}\sum_{i=1}^{N_D} D_s(t) \end{aligned} \tag{6}$$

where the factor 2 ensures compatibility with Eq. (5). Like [23], [32], this work employs a FAST corner detector [69] in combination with a pyramidal Lucas-Kanade feature tracker [70], [71] as implemented in the Paparazzi autopilot software[1]. To limit computational expense, N_D is capped at 100 points.

[1] http://wiki.paparazziuav.org/wiki/Main_Page

In addition to an estimate of divergence $\hat{D}$, an estimate of the derivative of divergence $\Delta\hat{D}$ can also be computed with a simple time difference:

$$\Delta\hat{D}(t) = \frac{\hat{D}(t) - \hat{D}(t - \Delta t)}{\Delta t} \tag{7}$$

As suggested by [23], the divergence derivative could play a role in controlling fast landings, where knowledge of changes in landing speed become more important.

Although the aim of the authors is to work towards end-to-end-learnt, neuromorphic optical flow control, they have opted to go with a frame-based instead of an event-based optical flow set-up here. It is believed that the added demonstration value of such a set-up does not weigh up to the increased complexity due to the integration of an event-based camera. Furthermore, as demonstrated by [23], [32], event-based optical flow estimation actually outperforms frame-based estimation in terms of accuracy, making for an easier control problem. Finally, the goal of end-to-end-learnt neuromorphic control implies estimation of optical flow is learnt, making it more important to demonstrate the feasibility of an interface with [27] than to implement a to-be-replaced event-based estimation technique.

B. Spiking Neural Network Architecture

SNNs fundamentally mimic the biological networks of neurons found in the animal brain. Data is sent through these networks as sparse sets of spikes, which makes them computationally more powerful [8], [12], more energy efficient [11], [72], [73] and better able to cope with rapidly changing inputs [74]. Various neuron models with differing levels of abstraction are available, ranging from accurate biophysical formulations [75] to more tractable models, of which the most-used are the *leaky integrate-and-fire* (LIF) [76] and the *spike-response* model (SRM) [77]. Regardless of the used model, the conceptual principles of each SNN are the same. Neurons are connected through *synapses*, which have a certain *efficacy* (weight) with which they conduct spikes. The alteration of this weight through learning is called *synaptic plasticity*. The neuron on the receiving end of a synapse is named *postsynaptic*, while the transmitting neuron is labelled *presynaptic*. Incoming spikes contribute to the *membrane potential* $u_i(t)$ of a neuron in an additive (*excitatory*) or subtractive (*inhibitory*) way. In case no inputs are received, $u_i(t)$ decays to a resting potential u_{rest}. On the other hand, if the quantity of inputs is large enough to push the membrane potential above a threshold θ_i, the neuron itself emits a spike, after which the potential is reset to u_{rest} and the neuron enters a *refractive period* during which membrane potential is stagnant.

Because this work aims to apply SNNs as a computational framework for real-time control, and only to a lesser extent strives to achieve a completely accurate representation of biological processing, we opted for an SNN consisting of LIF neurons. The success of previous, related applications of the LIF model helps in this respect [27], [78]–[80].

Although neuronal dynamics are considered to be continuous, they have to be discretised in some way for use in computer simulations. Taking the LIF model for membrane potential $u_i(t)$ and discretising it with a forward Euler technique, we end up with:

$$u_i(t) = u_i(t - \Delta t) \cdot \tau_{u_i} + \alpha_{u_i} i_i(t) \tag{8}$$

where we assumed $u_{rest} = 0$ and take the membrane decay as a factor τ_{u_i} instead of a scaled time constant. $i_i(t)$ is the forcing function working on the postsynaptic neuron i, which usually corresponds to the presynaptic spikes multiplied by their respective synaptic weights, i.e., $i_i(t) = \sum_j w_{ij} s_j(t)$. Instead of spikes, the neuron can also receive presynaptic currents $c_j(t)$, i.e., $i_i(t) = \sum_j w_{ij} c_j(t)$. The influence of the forcing function on the membrane potential is scaled with a constant α_{u_i}.

Neuron i emits a binary spike s_i at time t if $u_i(t) = \theta_i$. Immediately after, the membrane potential is reset to u_{rest}, and the neuron may enter into a refractory period, interrupting dynamics for a few milliseconds. Though implementing this could be done through a simple counter, we opted to neglect this feature to somewhat limit the number of hyperparameters.

Instead of keeping θ_i fixed, it can be made dependent on the neuron's firing rate. By increasing the threshold slightly for each emitted spike, excessive firing can be prevented, while continually decaying the threshold ensures the neuron stays responsive to small inputs. Furthermore, threshold decay ensures spiking can occur no matter the weight initialisation, meaning that learning can always take place. An adaptive LIF neuron's threshold can thus be represented as follows:

$$\theta_i(t) = \theta_i(t - \Delta t) \cdot \tau_{\theta_i} + \alpha_{\theta_i} s_i(t) \tag{9}$$

The binary nature of SNNs requires functions that transform real-valued signals to binary spikes and vice-versa, i.e., *encodings* and *decodings*. Two types of encodings are used in this work. The first makes use of two non-spiking neurons per input observation, one for positive and one for negative observation values. These neurons give off a proportional current, with at most one neuron being active at a given time. The non-spiking nature of these neurons furthermore allows them to be responsive to changing inputs, and similar mechanisms are actually observed in the biological retina's photoelectric cells [6], [81]. More specifically, the current $c_i(t)$ coming out of each neuron can be expressed as:

$$c_i(t) = |f(o_i(t))| \tag{10}$$

where $o_i(t)$ is the observation variable belonging to neuron i, and $f(\cdot)$ is a clamping function, i.e., $\min(0, \cdot)$ in case of the negative neuron and $\max(0, \cdot)$ in case of the positive neuron.

The second encoding is based on the idea of *place cells* [82], which are neurons in the animal brain that have been observed to fire when its host finds itself in a certain spatial location, with different neurons coupled to different locations, thus providing a sort of 'cognitive map'. The authors of [18], [19] adapt this concept to encode ranges of real-valued observations by distributing place cells modelled by inhomogeneous Poisson processes uniformly across these ranges, with the firing

probability of any cell dependent on the distance between the current value and the centre of the cell. This work employs a non-spiking variant, which instead gives out a proportional current:

$$c_i(t) = \exp\left(-\frac{\|\mathbf{o}(t) - \boldsymbol{\pi}_i\|^2}{2\boldsymbol{\sigma}_p^2}\right) \tag{11}$$

where $\mathbf{o}(t)$ is the current observation vector, $\boldsymbol{\pi}_i$ contains the place cell's centre for each observation variable and $\boldsymbol{\sigma}_p^2$ is a vector regulating the width of place cells per observation variable. Here, we always take $\boldsymbol{\sigma}_p$ equal to the distance between two adjacent, uniformly distributed centres to ensure enough overlap.

Instead of a uniform distribution of place cells, it might be preferred to have a non-uniform distribution, for instance to provide higher resolution for values close to zero, which is also a characteristic of the output layer in [27]. In this case, $\boldsymbol{\pi}$ can follow a cubic spacing function. Fig. 3 illustrates the two layouts. For both cases, $\sigma_p = 5$.

As Fig. 3 illustrates, observations that lie far outside the outermost place cell centres may lead to no current at all, and thus no spiking (and no control). To prevent this, observations are clamped to the range between the outer place cell centres, i.e., $\mathbf{o}(t) = \max(\mathbf{r}_1, \min(\mathbf{o}(t), \mathbf{r}_2))$, such that activity is identical to observations that lie exactly on these centres.

For decoding binary spikes to real-valued scalars (actions) in a range $[r_1, r_2]$, the postsynaptic *trace*, which is essentially just a low-pass filter over postsynaptic spikes, can be used, together with, e.g., a simple scaling:

$$\begin{aligned} a_i(t) &= r_1 + (r_2 - r_1) \cdot \frac{X_i(t)}{\eta} \\ X_i(t) &= X_i(t - \Delta t) \cdot \tau_{x_i} + \alpha_{x_i} s_i(t) \end{aligned} \tag{12}$$

Different combinations of α and τ lead to different decoding dynamics. For instance, little decay and large additions due to incoming spikes allow higher values to be reached, but also impede the trace to come down quickly again, making dynamics rather slow. On the other hand, high α and τ would make for a very fast-adapting decoding, but also one that would have problems giving out a temporally stable value.

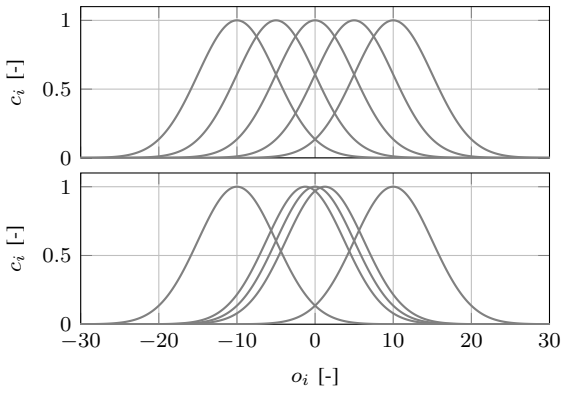


Figure 3. Position of uniformly distributed and cubically distributed place cells in the observation space.

Decodings different from the one mentioned above, such as the weighted vector of neuron activities in [19], were also tested in preliminary experiments, but these were found to have inferior performance, while adding complexity.

The SNN used for the control task in this work is kept relatively simple, with only a single hidden layer of not more than 20 adaptive LIF neurons, and a single output LIF neuron (we consider vertical control to be one-dimensional, with the SNN controller setting the thrust). The adaptive neurons in the hidden layer ensure sufficient initial spiking, easing learning during the first few generations. A similar-sized ANN in [23] demonstrated quite high-resolution control, thus proving the feasibility of using only a small network. In fact, there might even be a certain degree of redundancy in these smaller networks, both in terms of hyperparameter overlap as well as connections and hidden neurons.

C. Evolving Energy-efficient Neuromorphic Controllers

The vast majority of evolutionary algorithms starts off with the same four components:

- A *population* of randomly initialised candidate solutions (*individuals*), each having a certain policy
- A method to evaluate the *fitness* (performance) of these individuals
- A way to alter individuals, generating new policies
- A selection mechanism to select well-performing individuals and filter out bad ones

The tasks of each of these four components can be carried out in different ways. Alteration of individuals can happen, for instance, through *mutation* or *crossover*, which modifies an existing individual or combines parts of others, respectively, to create *offspring*. There seems to be little consensus in existing literature [83]–[86] as to what an evolutionary framework for SNNs should look like. We opt for a mutation-only approach, because crossover tends to work best when natural building blocks are available. The distributed knowledge representation in neural networks does not seem to satisfy this requirement, which could lead to destroyed policies or the permutation problem, where different networks have the same policy [87].

Selection is carried out using the multi-objective genetic algorithm *NSGA-II* [88]. The ability to deal with multiple objectives gives us the ability to specify different aspects of a 'good' policy, and also gives a flexible framework for adding objectives later. The spectrum of policies resulting from a multi-objective optimisation allows us to distinguish various optimal behaviours, as demonstrated by [23] for the evolution of divergence-based landing controllers. Alternatives to NSGA-II would be an *evolution strategies* approach, such as *MO-CMA-ES* [89], or another genetic algorithm, *MOEA/D* [90]. Both evolution strategies and genetic algorithms have seen successful large-scale implementations for ANNs [40], [42]. Although [91] showed MOEA/D outperforming NSGA-II for some problems, we use the latter here because of its good performance in [23] and its availability in DEAP [92], a distributed Python framework for evolutionary algorithms.

The objectives used in this work are almost identical to those used in [23] to evolve landing controllers. In order to stimulate fast landing, f_1 is the time it takes to land, i.e., reach an altitude of 0.05 m (to prevent infinite D and account for landing leg length). Next, f_2, which is the final altitude, encourages landing in general. f_3, the final velocity, ensures landings are soft. As mentioned before, one of the benefits of SNNs is their decreased energy consumption compared to ANNs [11]. The authors of [93] looked into further minimising this consumption when converting ANNs to SNNs, by including loss function terms representing connection sparsity or predicted number of spikes. Though their results show that network activity can be reduced by more than half while maintaining accuracy, there seems to be no clear winner as to which of these methods works best. We therefore opt to include the most straightforward of the two, total spike rate of the complete network, as an additional objective f_4.

At the start of the evolutionary process, a population of μ SNN individuals is initialised with random weights, while hyperparameters are set either according to the standard initial configuration (in case they will be mutated) or a set of selected values (in case they are fixed). The individuals are then evaluated in a set of randomised environments to determine their fitness (f_1, f_2, f_3) or (f_1, f_2, f_3, f_4).

After this, the steps for each generation are the same and as follows:

1) Select a Pareto front of non-dominated individuals from the population. The remaining individuals compete in a tournament based on dominance and *crowding distance* (see [88]) for the leftover spots, such that we end up with a selection of μ individuals.
2) Form offspring λ through mutation of the selected individuals.
3) Randomise the environments and subsequently re-evaluate both the population and its offspring. The fact that the population is evaluated again decreases the chance that individuals will be selected simply because they received 'easy' environments, which is realistic given the high stochasticity of our evaluations.
4) Select a new population of μ individuals from both the old population and its offspring $(\mu+\lambda)$ using NSGA-II.

Depending on the case, different mutations can take place when creating the offspring. The mutation probability P_{mut} is fixed to 0.3 for all mutations, meaning that parameters have a 30% chance of being replaced by a mutated variant. Table I lists the distributions per parameter from which these replacements are sampled. Note that the distribution for synaptic weights is identical to the one used in [23]. Threshold θ_i is only mutated for non-adaptive neurons, i.e., those in the output layer.

During the evolution, a *hall of fame* is maintained, which contains the all-time Pareto front of best individuals. After N_{gen} generations, all individuals in the hall-of-fame are evaluated by letting them perform 250 landings and quantifying the median and inter-quartile range (IQR) for each evolutionary objective. Then, the best-performing individuals are selected.

Table I
SAMPLING DISTRIBUTIONS OF MUTATED PARAMETERS

Parameter	Distribution
w_{ij}	$\mathcal{U}(-w_{ij} - 0.05, 2w_{ij} + 0.05)$
α_{u_i}, α_{θ_i}, α_{x_i}	$\mathcal{U}(\alpha - 2/3, \alpha + 2/3)$, clamped to $[0, 2]$
τ_{u_i}, τ_{θ_i}, τ_{x_i}	$\mathcal{U}(\tau - 1/3, \tau + 1/3)$, clamped to $[0, 1]$
θ_i	$\mathcal{U}(\theta_i - 1/3, \theta_i + 1/3)$, clamped to $[0, 1]$

So even though the hall-of-fame may still contain individuals that got 'lucky' during evolution, these will be identified and disregarded during the subsequent evaluation.

D. Randomised Vertical Simulation Environment

The vertical simulation environment in which individuals are evaluated makes use of domain randomisation and artificial noise to improve transferability to the real world. The available observations are the divergence $\hat{D}$ and its derivative $\Delta\hat{D}$. The simulated MAV is just a unit mass under influence of gravity, and control happens in 1D with the SNN controller selecting a thrust setpoint T_{sp}. This leads to the following dynamics model [23]:

$$\begin{aligned} h(t) &= h(t-\Delta t) + \Delta t \cdot v(t-\Delta) \\ v(t) &= v(t-\Delta t) + \Delta t \cdot T(t-\Delta t) + w(t) \\ T(t) &= T(t-\Delta t) + \Delta t \cdot \frac{T_{sp} \cdot g - T(t-\Delta t)}{\Delta t + \tau_T} \end{aligned} \tag{13}$$

where the states altitude h, velocity v and thrust T (in ms^{-2}) are updated using the forward Euler method. τ_T represents the spin-up and spin-down time of the rotors. The thrust setpoint T_{sp} selected by the SNN is clamped to a realistic range of acceleration for the MAV, namely $[-0.8, 0.5]$ g. w is added vertical wind, according to the model in [23]:

$$w(t) = w(t-\Delta t) + \Delta t \cdot \frac{\mathcal{N}(0, \sigma_{wind}^2) - w(t-\Delta t)}{\Delta t + \sigma_{wind}} \tag{14}$$

with σ_{wind} the standard deviation of the normally distributed wind.

Noise is added to the divergence estimation according to the model in [33]. The observed divergence $\hat{D}$ is the result of adding a delay δ_D to the ground-truth divergence, along with white noise and proportional white noise:

$$\begin{aligned} \hat{D}(t) = D(t - \delta_D \cdot \Delta t) &+ \mathcal{N}(0, \sigma_D^2) \\ &+ D(t - \delta_D \cdot \Delta t) \cdot \mathcal{N}(0, \sigma_{D_{prop}}^2) \end{aligned} \tag{15}$$

where σ_D and $\sigma_{D_{prop}}$ are the standard deviations for the added noise and proportional noise, respectively. Additionally, computational jitter is introduced similarly to [23], in order to simulate the case in which the estimated divergence is not updated due to, e.g., insufficient corners. Each time step, there is the probability P_{jitter} that the estimated divergence from the previous step is used (for a maximum of one step).

Table II
SAMPLING DISTRIBUTIONS OF ENVIRONMENT PARAMETERS [23]

Parameter	Distribution
δ_D	$\mathcal{U}\{1,4\}$ steps
σ_D	$\mathcal{U}(0.05, 0.15)$ s^{-1}
$\sigma_{D_{prop}}$	$\mathcal{U}(0.0, 0.25)$ s^{-1}
τ_T	$\mathcal{U}(0.005, 0.04)$ s
Δt	$\mathcal{U}(0.02, 0.0333)$ s
P_{jitter}	$\mathcal{U}(0.0, 0.2)$

Added delay and noise, motor dynamics, simulation frequency and jitter are randomised to minimise the reality gap. Table II lists the distributions from which these are drawn.

The evaluation of an individual consists of four landings, from initial altitudes $h_0 = 2, 4, 6, 8$ m. The environment is bounded in altitude and time: $[0.05, h_0 + 5]$ m and 30 s, respectively. Individuals start out without initial velocity and acceleration, and are left to settle for 0.5 s. Each landing has its own, differently randomised environment, with parameters (as in Table II) being redrawn at the start of each generation, such that all individuals experience the same four environments. Fitnesses are averaged across the four landings, with individuals that do not manage to land receiving extra punishment. It might be argued that the averaging of fitnesses across different altitudes leads to a biased f_1 (time to land), however tests indicated that scaling f_1 with initial altitude did not improve performance.

IV. EXPERIMENTS

A. Experimental Set-up

1) Simulation: Several cases are considered in simulation. *20-base* is the baseline configuration given in Table III in Appendix A. *20-sm* adds a fourth evolutionary objective, f_4, to minimise the SNN's total spike rate. *20-sm-pu* and *20-sm-pc* additionally implement place cell encodings (only divergence), in order to potentially show a successful interface between the SNN in this work and the one in [27]. *1-sm* and *0-sm* are set up to investigate the SNN's redundancy regarding hidden neurons, while *20-sm-wt* and *20-sm-w* try to quantify the effect fixing addition/decay constants and thresholds has on performance, respectively. Table IV in Appendix A gives the exact variations for all non-default cases.

20-base, *20-sm* and *1/0-sm* are each initialised four times (meaning four evolutions), after which the final hall-of-fame individuals are combined for analysis. *20-sm-pu/pc*, *20-sm-wt* and *20-sm-w* are all initialised twice to save computational expense. All initial synaptic weights are drawn from a uniform distribution $\mathcal{U}(0, 1)$. Simulations are carried out in Python 3.6 on a laptop running Ubuntu 18.04 LTS, equipped with an Intel i7-7700HQ quad-core CPU (eight virtual cores) and 16 GB of memory.

The framework used here for simulating SNNs is *PySNN*[2], which is written on top of the popular deep learning package *PyTorch*[3] and allows both quick prototyping and efficient execution through a modular approach of `Neuron` and `Connection` objects.

2) Real World: The MAV used in this work is a Parrot Bebop 2 running the Paparazzi autopilot software on its 780 MHz dual-core ARM Cortex A9 processor. Landings start from an initial altitude of roughly 4 m. Horizontal guidance and ground-truth measurements are provided by an OptiTrack[4] motion capture system. Optical flow is estimated using the Bebop's downward-facing CMOS camera, with texture being provided by a car play mat, as is shown in Fig. 1. Each landing run starts off with 10 seconds of hover, in order to reliably calibrate the nominal thrust needed for hover. Optical flow is estimated at a rate of approximately 45 Hz, while the custom vertical control loop implementing the divergence-based landing runs at roughly 512 Hz. Measurements are logged at approximately 100 Hz.

To run the SNN on board, a framework for building small spiking networks has been developed in C. This framework, called *TinySNN*, is based on the above-mentioned PySNN, allowing an almost seamless transfer of networks from simulation to the real-world hardware. Links to TinySNN and the accompanying Paparazzi code can be found at the bottom of the first page.

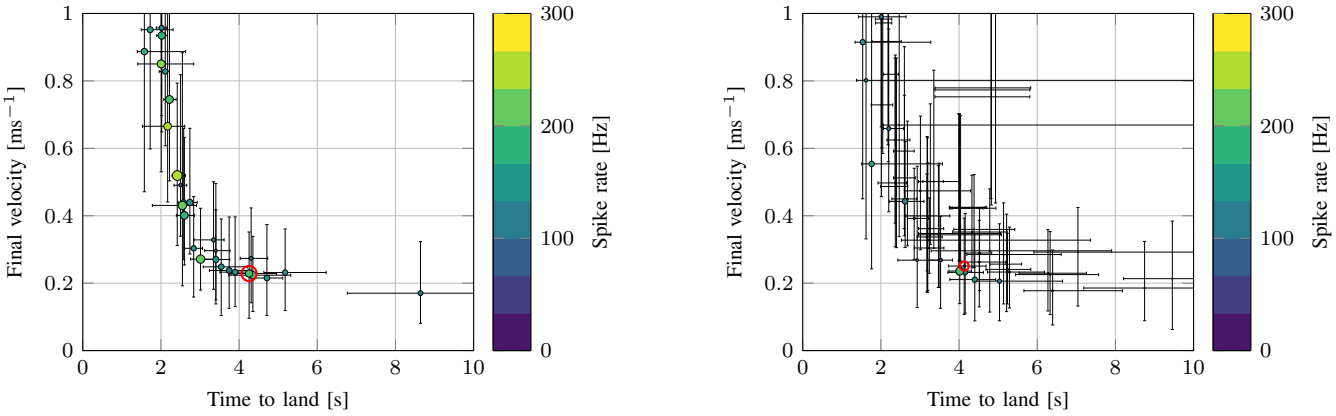
As discussed in [23], linearly transforming the thrust setpoint T_{sp} to rotor commands leads to poor tracking performance due to unmodelled drag and non-linear aerodynamic effects due to a descent through the propeller downwash. To solve this, a PI controller was used to convert the thrust setpoint to motor commands, as was suggested by [45]. The gains of this controller, together with other settings related to real-world tests, can be found in Table VI in Appendix B.

B. Effect of Energy Minimisation on Control Resolution

While the concept of energy minimisation is appealing for any type of controller, SNNs are particularly suited because of their sparse, spiking nature. However, including minimal spike rate as an objective during the evolutionary process may have an influence on the control resolution of the resulting individuals. For instance, a certain increase in desired thrust may be achieved through several output neuron spikes that each increase its postsynaptic trace somewhat (small α_{x_i}), or through a single, more effective spike (large α_{x_i}). However, the larger the α_{x_i}, the smaller the general control resolution, given that fewer intermediate values of postsynaptic trace (and thus desired thrust) can be achieved. Therefore, it is important to quantify the effect of spike minimisation on control resolution.

Fig. 4 displays two Pareto fronts (based on median performance) of evolved individuals side by side: one optimised for minimal spiking (*20-sm*), and one where this objective was

[2]https://github.com/BasBuller/PySNN
[3]https://pytorch.org/
[4]https://optitrack.com/



(a) *20-base*: not optimising for spike rate.

(b) *20-sm*: optimising for minimal spike rate.

Figure 4. Sensitivity analysis Pareto front (based on median performance) of individuals in the final hall of fame for *20-base* and *20-sm*. Error bars indicate the 25$^{\text{th}}$ and 75$^{\text{th}}$ percentile of performance over 250 evaluations in terms of time to land and final velocity from a starting height of 4 m. The radius of the dots is proportional to the spike rate's IQR, and the colour indicates the median. Selected individuals are indicated by a red circle. The dimension responsible for final altitude (f_2) of the controllers is not shown in this figure, as this was almost consistently minimised for all of them.

left out (*20-base*). The performance (and sensitivity) in terms of time to land and final velocity is similar for both evolutions, but the spike-optimised population achieves this performance with much fewer spikes: the median total spike rate for *20-base* is 162.3 Hz, while that of *20-sm* is 40.8 Hz.

Comparing Fig. 4a with the Pareto fronts of ANN controllers in [23], two differences immediately become apparent: 1) the variance in performance of ANN-based individuals is much smaller, especially for the slower-landing individuals ($t > 6$ s), and 2) these slower-landing individuals take up a larger part of the resulting controllers and have a much lower median final velocity ($v < 0.1$ ms^{-1}) in the case of evolving ANNs. Some methods for promoting slower-landing individuals were tried (e.g., an increased weighting factor for f_3 or a spike decoding based on a weighted vector of neuron activities), but to no avail.

Testing two selected individuals from both evolutions in simulation, we see in Fig. 5 that the flight profiles when starting from a height of 4 m look roughly similar across five runs in a randomised environment. Differences in terms of variability in time to land and the final velocity can be explained by the difference in sensitivity of the selected individuals. A more notable disparity is observed in the vertical velocity profiles and the thrust setpoint T_{sp}, which are more flattened and which have a smaller magnitude of oscillation, respectively, for the spike-optimised individual. Though one could think this is because of the decreased spiking of the output neuron, this is not the case, as indicated by Fig. 6, where we show both networks, their synaptic weights and the average firing rate per neuron side by side. The exact firing rates of both output neurons are 21.2 Hz and 19.6 Hz for *20-base* and *20-sm*, respectively.

It is immediately clear from Fig. 6 that only very few neurons contribute to the actual actions of both individuals. In fact, it seems to be that, for both networks, only a single path from input to output layer causes all output neuron activity. Fig. 17 in Appendix C confirms this, where the spike traces of the responsible hidden neurons ($i = 7$ for *20-base* and $i = 11$ for *20-sm*) are identical to those of the respective output neurons, with both $R^2 = 1$. And while there is still a lot of 'redundant' spiking going on in the individual from *20-base*, this has almost entirely been eliminated in the network belonging to *20-sm*, with all but a few hidden neurons completely dormant, and a lot of inhibitory synaptic connections. It seems

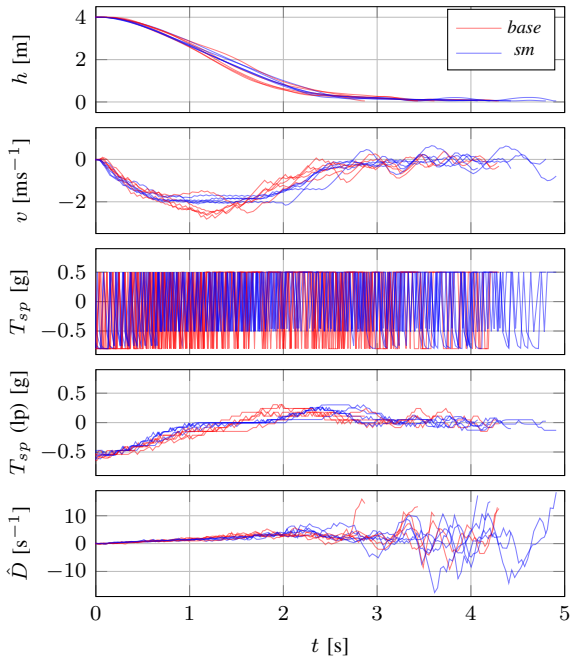
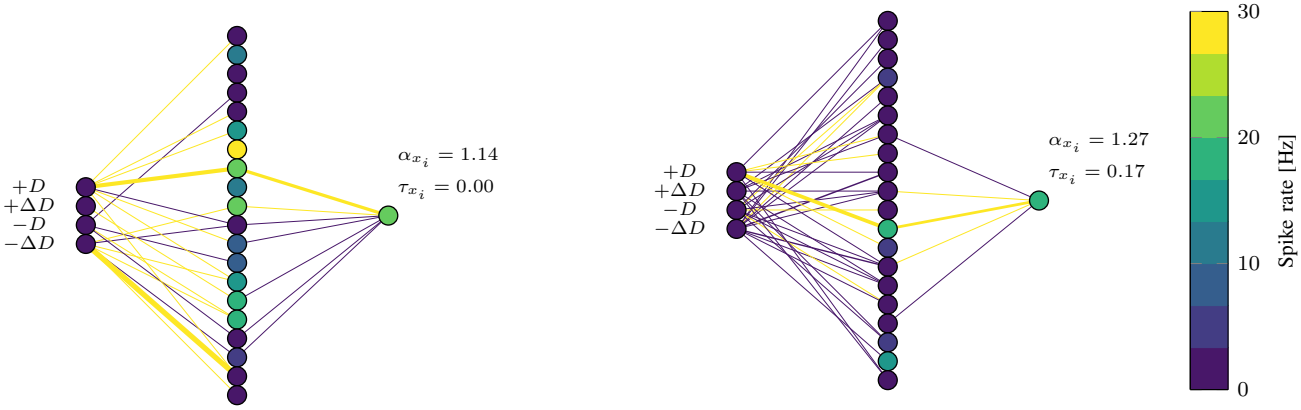


Figure 5. Height, velocity, thrust setpoint (raw and 20-step moving average) and divergence for five simulated runs of selected individuals from *20-base* and *20-sm* (see Fig. 4) in a randomised environment.



(a) *20-base*: not optimising for spike rate.

(b) *20-sm*: optimising for minimal spike rate.

Figure 6. Average firing rates (in Hz) and synaptic weights of selected individuals from *20-base* and *20-sm* for the five simulated runs displayed in Fig. 5. Vertex colour is proportional to neuron firing rate, while synaptic weight is directly proportional to edge weight in points. Edge colours indicate inhibitory (purple) or excitatory (yellow) synapses. Synaptic connections with a weight $|w_{ij}| < 0.05$ are not shown. The labels indicate the quantity encoded by each input neuron.

reasonable to assume that further evolutionary optimisation would eliminate also the remaining redundant spike activity.

Given that the spike traces of the output neurons for both cases are also pretty similar, the observed difference in T_{sp} must be the result of different addition and decay constants. And indeed: while *20-sm*'s output neuron has $\tau_{x_i} = 0.174$, the individual from *20-base* has an output neuron with $\tau_{x_i} = 0.0$. In combination with the neuron's $\alpha_{x_i} > 1$, this means that T_{sp} is either -0.8 g or 0.5 g. In other words: the SNN from *20-base* acts as a pure *bang-bang* controller (also called on-off control [94]), while the controller from *20-sm* allows for some intermediate values of T_{sp}. Opposite to what was expected, it seems therefore that energy minimisation has a positive effect on control resolution.

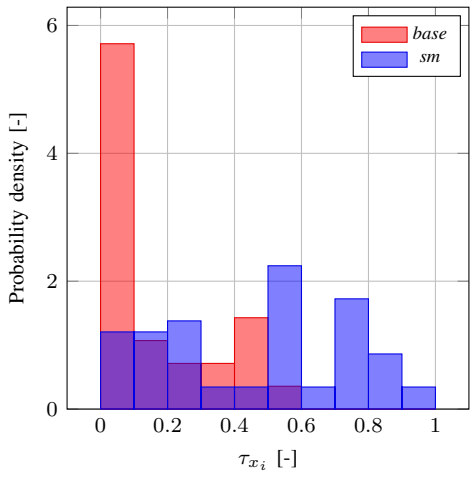


Figure 7. Probability density histograms of the values of τ_{x_i} of the output neuron for all individuals on the Pareto front for *20-base* and *20-sm* (see Fig. 4). The Mann-Whitney U test confirms, with a p-value of 3.15e−7, that these histograms are drawn from different distributions. Medians for both cases are 0.006 and 0.571, respectively.

To support this claim, Fig. 7 shows the probability density histograms of the values of τ_{x_i} of the output neuron for all Pareto individuals from *20-base* and *20-sm*. The (statistically significant) absence of a large peak near zero for *20-sm* confirms that adding spike minimisation as an objective contributes to smoother control, with fewer output neuron decays equal (or very close) to zero.

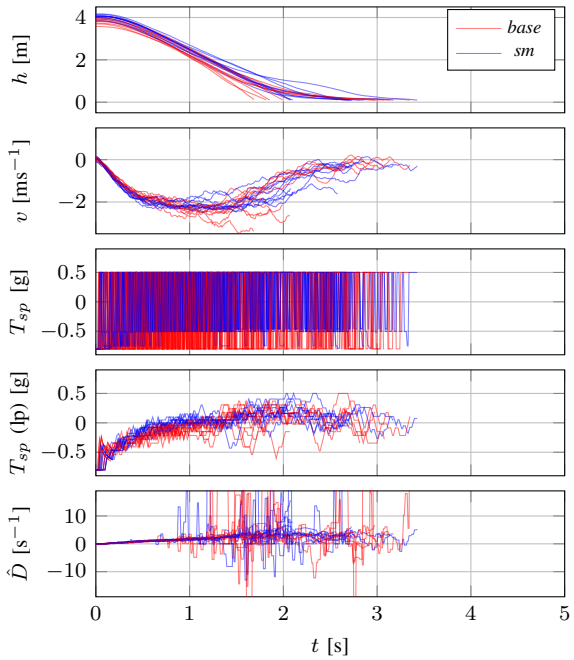


Figure 8. Height, velocity, thrust setpoint (raw and 20-step moving average) and divergence for ten **real-world** flight tests of selected individuals from *20-base* and *20-sm* (see Fig. 4). Runs ended when $h < 0.1$ m (instead of the $h < 0.05$ m in simulation) to account for the offset created by the MAV's landing legs at initialisation. The length of the t-axis is kept at 5 s for easier comparison with Fig. 5.

Real-world flight tests of the selected individuals from *20-base* and *20-sm* can be seen in Fig. 8. The latter spike-minimised controller seems much more able to cross the reality gap, with touchdowns almost as smooth as in simulation. The *20-base* individual, on the other hand, quite often lands with a final velocity in excess of 2 ms^{-1}, something which was not observed in simulation. It seems, therefore, that the bang-bang control evolved in *20-base* is ineffective in the real world, while the slightly smoother control policy of *20-sm* transfers rather well. This observation is confirmed by [23], who argue that quick alteration of completely spinning up and down the rotors causes control to be dependent on motor dynamics (spin-up and spin-down time τ_T). Given that these dynamics are almost certainly different for simulation and real world, bang-bang controllers will likely not transfer well. This might also explain why both individuals in Fig. 8 perform much faster landings in the real world (around 3 s) than in simulation (around 5 s). The decreased oscillatory magnitude of the bang-bang control by the *20-sm* individual, however, allows for sufficiently smooth control, while the larger magnitude of the *20-base* controller does not. Approaching the control smoothness of the ANNs in [23] would most likely require a different spike decoding, or a clamping on the output neuron's trace decay, to prevent near-complete decays. Section IV-D will look into the effectiveness of the latter approach for networks with fewer hidden neurons.

C. Towards End-to-End-Learnt Control

One of the aims of this paper is to demonstrate a successful interface between the output layer of [27] and the landing controller developed here, in order to advance towards fully learnt, neuromorphic optical flow control. To this end, the cases here consists of two types of place cell encodings for divergence: one uniformly distributed (*20-sm-pu*), and one with cubically distributed centres (*20-sm-pc*). The output layer of the SNN in [27] correlates the optical flow visual observables (which include divergence) with the activity of certain neurons. The sensitivity of these neurons is higher for values close to zero, something which is also true for the cubically distributed place cells of *20-sm-pc*. Therefore, we regard a demonstration of successful landings by SNN controllers from this case as a proxy for a successful interface with [27].

Fig. 9 shows the individuals on the median-performance Pareto front for *20-sm* and *20-sm-pu/pc*. While the fronts are mostly similar, there do seem to be more slow-landing individuals with lower touchdown velocities for *20-sm-pu* and *20-sm-pc*. This is most likely a coincidence, as there is no direct connection between the encoding of divergence and the smoothness of evolved control policy. Nonetheless, it is interesting to further examine the characteristics of these controllers.

Simulated runs of the three selected individuals from Fig. 9 are displayed in Fig. 10. From this, we can conclude that a place cell encoding is suitable for the learning problem presented here. It should therefore also be possible to achieve successful landings using the divergence estimation network from [27].

Examining the performed landings in Fig. 10 in more detail, we see that the controllers from *20-sm-pu/pc* take a bit longer to land, but do so with a smoother touchdown and much decreased oscillations in velocity. This is consistent with the median performance of the individuals as stated in Fig. 9. Also interesting are the plots of the desired thrust T_{sp}, which show

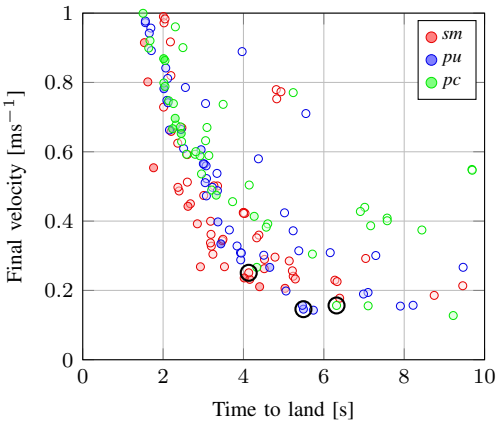


Figure 9. Pareto front (based on median performance over 250 evaluations) of individuals in the final hall of fame for *20-sm* and *20-sm-pu/pc*. The dot's colour shade is proportional to the spike rate median: lighter means a higher rate. As opposed to Fig. 4, the dot size is constant, meaning that no sensitivity information can be inferred from this plot. Selected individuals are indicated by a black circle. The dimension responsible for final altitude (f_2) of the controllers is not shown in this figure, as this was almost consistently minimised for all of them.

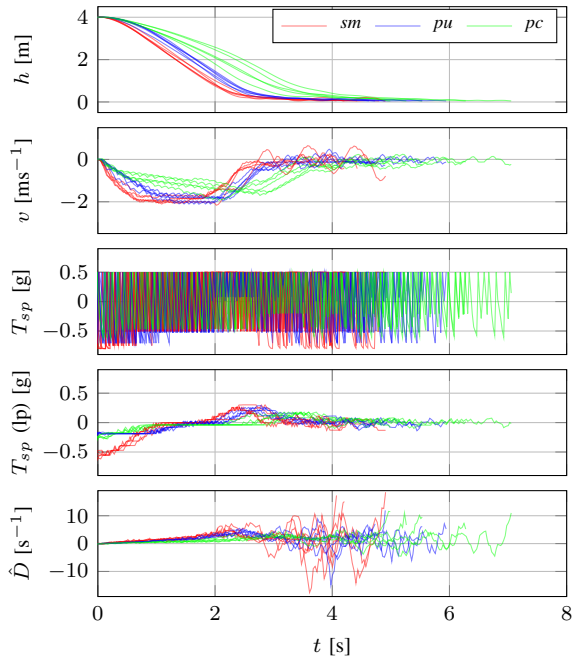


Figure 10. Height, velocity, thrust setpoint (raw and 20-step moving average) and divergence for five simulated runs of selected individuals from *20-sm* and *20-sm-pu/pc* (see Fig. 9) in a randomised environment.

less bang-bang behaviour for both *20-sm-pu* and *20-sm-pc*. As mentioned before, this is mainly the result of the decay constant τ_{x_i} of the output neurons, which is 0.34 and 0.61 for *20-sm-pu* and *20-sm-pc*, respectively. Figs. 18a and 18b in Appendix D also list these values and additionally show the structure and activity of the SNNs. As was the case with the networks in Fig. 6, there is a single path from input to output that is responsible for most of the control. Looking at the labels next to the input neurons, which indicate the centres (in s^{-1}) of the place cells encoding divergence, we see that both these paths originate from place cells representing positive values of divergence. In fact, the same is true for the networks in Fig. 6, where the responsible paths start from the positive divergence neuron.

Another way to look at the strategy of evolved controllers is through visualisation of the transient and steady-state responses. Fig. 11 compares the transient response for *20-sm* and *20-sm-pu/pc*. The steady-state response, as well as an enlarged version of the transient response, can be found in Fig. 19a in Appendix E. Transient responses are made up of multiple simulated landings for which divergence and desired thrust are recorded, while the steady-state response is quantified by subjecting the individuals to a constant observation (divergence only for *20-sm-pu/pc*) for a certain number of time steps. Both the transient and steady-state response show a range of thrust setpoints that seems to correspond with the range of actions in Fig. 10. Also clearly visible are the number of different thrust levels each controller can achieve: while the *20-sm* controller on the left only has three levels, the individuals from *20-sm-pu/pc* can choose from a greater number of setpoints, which seems to be beneficial for control smoothness. Furthermore, the range of possible T_{sp}'s is smaller for the latter individuals. One could argue that this results in less responsive control, however the performance in Fig. 10 suggests that values close to the the thrust bounds are not necessary for smooth landing control that is still sufficiently quick. Given that both the greater number of possible setpoints, as well as the thrust range, are likely to be the result of smaller addition constants in combination with less decay, we will try to promote these in the next section. A more detailed analysis of the transient and steady-state response of the various controllers, as well as a comparison with the often-used proportional controller [29]–[32] will also be given there.

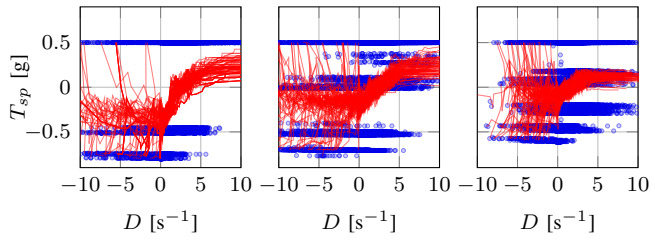


Figure 11. Transient responses of selected individuals from *20-sm* (left), *20-sm-pu* (middle) and *20-sm-pc* (right) as given in Fig. 9. To obtain these, observed divergence D and thrust setpoint T_{sp} were recorded during 100 simulated landings (blue dots) and subsequently sorted by increasing divergence and passed through a 40-step moving average (red lines). An enlarged version can be found in Fig. 19a in Appendix E.

D. Promoting Smooth Control for a Single-Neuron SNN

One of the insightful outcomes of the energy-minimising evolution presented previously is an indication of the minimum amount of resources needed for solving the problem of landing based on divergence. Given the large number of dormant neurons and weak connections in the evolved SNNs of *20-base* and especially *20-sm*, one would be inclined to think networks with only a couple of hidden neurons would perform just as well. Therefore, the networks in this section have either a single hidden neuron (*1-sm*) or no hidden layer at all (*0-sm*). Fig. 6 suggests that controllers with only one hidden neuron should be able to land successfully, given the large number of dormant neurons and weak connections in both evolved SNNs. Evidently, this says little about the feasibility of landing controllers without a hidden layer. Recent research [95] implies, however, that single neurons may be more capable than has thus far been assumed: by responding in a damped manner to strong stimuli, certain kinds of neurons in the human brain are able to solve the XOR problem, which was previously thought to require multi-layer networks. Looking at our available divergence encodings, it could be that these allow a similar reduction in necessary network complexity.

In order to promote smooth control, the mutation of the various addition and decay constants is clamped to a range $[0, 1]$ and $[0.3, 1]$, respectively. In this way, complete decay of voltage, trace and threshold is prevented, while the smaller additions promote smoother traces and prevent uselessly large contributions to neuron voltages (given that thresholds θ_i are bound to $[0, 1]$). Table V in Appendix A lists these adjustments.

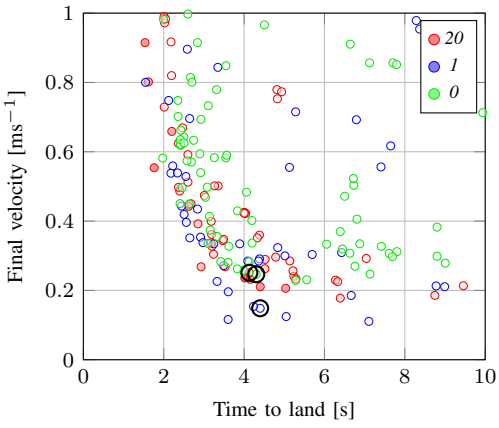


Figure 12. Pareto front (based on median performance over 250 evaluations) of individuals in the final hall of fame for *20-sm* and *1/0-sm*. The dot's colour shade is proportional to the spike rate median: lighter means a higher rate. As opposed to Fig. 4, the dot size is constant, meaning that no sensitivity information can be inferred from this plot. Selected individuals are indicated by a black circle. The dimension responsible for final altitude (f_2) of the controllers is not shown in this figure, as this was almost consistently minimised for all of them.

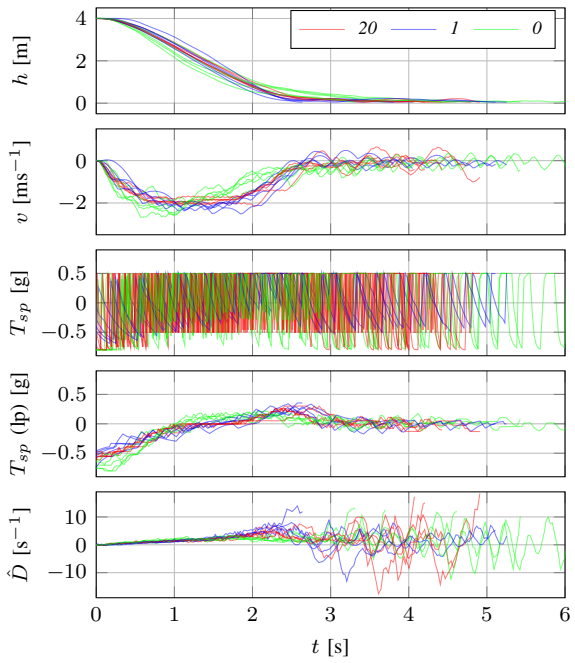


Figure 13. Height, velocity, thrust setpoint (raw and 20-step moving average) and divergence for five simulated runs of selected individuals from *20-sm* and *1/0-sm* (see Fig. 12) in a randomised environment.

Fig. 12 compares the median-performance Pareto front for *20-sm* and *1/0-sm*. It immediately becomes clear that the front of *1-sm* outperforms those of the other cases. Most likely, this is the result of a combination of an increase in evolutionary resources available per neuron and the prevention of inferior parameter combinations by clamping addition and decay constants. A quick comparison of *1-sm* and *0-sm* indicates that, while Pareto individuals from *0-sm* may be able to land successfully, having a hidden layer (even though it consists of only a single neuron) will lead to better performance.

Fig. 13 shows simulated landing runs of selected individuals from *20-sm* and *1/0-sm*, with all having a roughly equal time-to-land performance. The plots of vertical speed confirm that *1-sm* performs better in terms of final velocity. Interesting, however, is the way in which this performance is accomplished. As can be seen from the two graphs for thrust setpoint T_{sp}, *1-sm* is characterised by a very slow output neuron trace decay and few output spikes. This leads to a desired thrust that is slowly varying most of the time, but which can increase suddenly through the emission of a spike, resulting in the small 'hops' in vertical speed during descent. As the ground nears, the magnitude of these hops seems to decrease, leading to superior low-altitude control compared to *20-sm* and *0-sm*.

Nevertheless, the landings performed by the single-neuron controller (*0-sm*) also look promising. Like 3a, trace decay is slower, which allows a larger number of thrust setpoints to be selected. It remains to be seen, however, whether the oscillations are slow enough to transfer well to the real world.

Figs. 18c and 18d in Appendix D give the exact values of α_{x_i} and τ_{x_i}, along with the synaptic weights and the firing

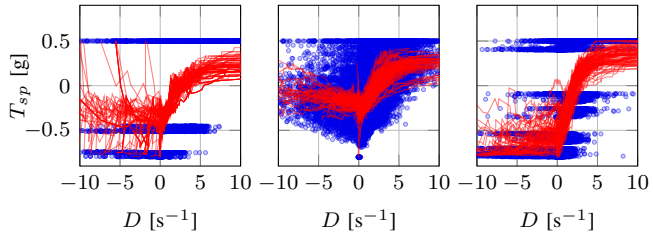


Figure 14. Transient responses of selected individuals from *20-sm* (left), *1-sm* (middle) and *0-sm* (right) as given in Fig. 12. To obtain these, observed divergence D and thrust setpoint T_{sp} were recorded during 100 simulated landings (blue dots) and subsequently sorted by increasing divergence and passed through a 40-step moving average (red lines). An enlarged version can be found in Fig. 19b in Appendix E.

rate of the neurons during the runs in Fig. 13. Though the network from *1-sm* has seemingly meaningful connections to both the $+D$ and $-D$ input neurons, we know from *20-base*, *20-sm* and *20-sm-pu/pc* that a connection to $+D$ should suffice. Nonetheless, it is possible that this pair of an excitatory and inhibitory connection contributes to smooth control.

Fig. 14 compares the transient response of (from left ro right) *20-sm*, *1-sm* and *0-sm*. The response of *1-sm* shows a much larger number of possible thrust setpoints than any other individual, due to its relatively small α_{x_i} and τ_{x_i} close to unity. The controller from *0-sm*, on the other hand, only has five levels, which have to cover the entire range $[-0.8, 0.5]$ g. It is therefore likely that *0-sm* will not transfer as well to the real world as *1-sm*.

Examining the transient responses in Fig. 19 in Appendix E more closely, we can see that those of *1-sm* and *0-sm* seem to be more tightly grouped together, especially for $D < 0$. This may be the result of having only a few spiking neurons, which allows for less variance in the first place. Coupled to this is the number of connections to neurons encoding negative divergence: while these may be weak, they can influence the output trace, and their contributions can be quite significant if the currents coming from other encoding neurons are zero.

The steady-state response of the selected individuals from *1-sm* and *0-sm* is also displayed in Fig. 19b in Appendix E. Like *20-sm*, the response shows a gradient in the vertical direction due to the availability of ΔD. The strength of this gradient corresponds to the synaptic weights in Figs. 18c and 18d: *1-sm* only has weak/zero connections to neurons encoding ΔD, while *0-sm* has a stronger link to $+\Delta D$.

Apart from individual characteristics, Fig. 19 also tells us something about the difficulty and nature of the problem of divergence-based landing. Clearly, the transient responses of the individuals all take some kind of sigmoid shape, though it may be more of a half sigmoid for $D \geq 0$, and a sloped line for $D < 0$. In comparison, the response given by a proportional divergence controller would be a straight line, with its slope dependent on the controller's gain. The steady-state plot of such a controller would have an even gradient along D, of course plateauing at the thrust bounds.

The problem and its solution are therefore certainly non-

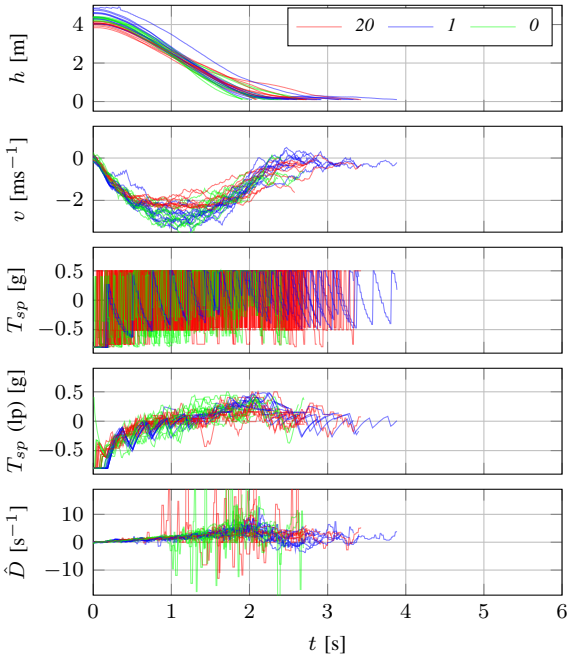


Figure 15. Height, velocity, thrust setpoint (raw and 20-step moving average) and divergence for ten **real-world** flight tests of selected individuals from *20-sm* and *1/0-sm* (see Fig. 12). Runs ended when $h < 0.1$ m (instead of the $h < 0.05$ m in simulation) to account for the offset created by the MAV's landing legs at initialisation. The length of the t-axis is kept at 6 s for easier comparison with Fig. 13.

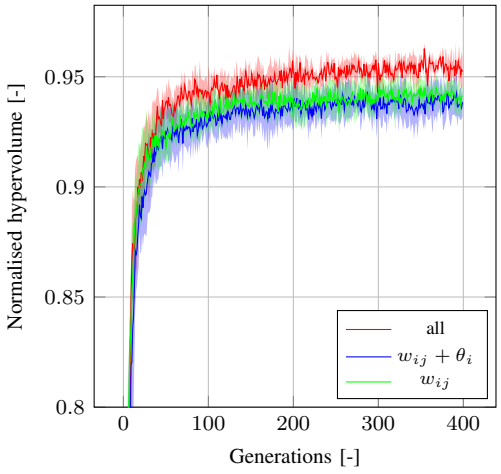


Figure 16. Normalised hypervolumes during the evolution of *20-sm*, *20-sm-wt* and *20-sm-w*. Note that *20-sm* is initialised four times, while the subcases of *20-sm-wt* and *20-sm-w* are each initialised twice. Solid lines indicate the mean over these initialisations, while the coloured fills give the standard deviation.

linear, and not entirely trivial. Actually, as [23] points out, the solution is quite complex if one wants to minimise the oscillations that follow from optical flow control. On the other hand, given that the transient solution has a sigmoid shape, a functioning controller could also be made of a single ANN neuron with a sigmoidal activation function. The beauty of the SNN, however, is that it can approximate this function through its inherent dynamics, which makes for an arguably simpler solution.

Fig. 15 compares the real-world performance of *1-sm* and *0-sm* to that of the individual from *20-sm*. All land more or less equally fast, but *1-sm* seems to touch down with the least amount of residual velocity. As was predicted, its slower control dynamics helped in a successful transfer from simulation. Still, real-world landings take about half a second less (before decelerating close to the ground) than the landings in simulation, so there obviously still is some discrepancy.

E. Evolving Parameter Subsets and Optimality

Apart from a redundancy in neurons, the large number of hyperparameters in SNNs could mean that some of these are also (partially) redundant, in the sense that the effect of changing these parameters can also be achieved by changing one or more other parameters. This would mean that we could fix at least some of the hyperparameters and optimise the others through evolution, while achieving equally good results. *20-sm-wt* and *20-sm-w* are set up to look into this, by offsetting addition and decay constants (α's and τ's) from the evolved values of the selected individual from *20-sm*, and seeing how well the evolution of weights and thresholds (*20-sm-wt*) or weights only (*20-sm-w*) can compensate for these offsets.

One way of quantifying how well an optimisation algorithm does its job is tracking how much of the total hypervolume of the 3D/4D fitness space it is able to 'consume'. As evolution continues, the Pareto front consisting of the best individuals should leave less and less room for optimisation. Fig. 16 presents the normalised hypervolume occupied by the optimisation. We distinguish three categories: optimising all hyperparameters and weights (*20-sm*, four initialisations), optimising weights and thresholds (*20-sm-wt*, three different settings with each two initialisations), and optimising weights only (*20-sm-w*, three different settings with each two initialisations).

First of all, it seems that most runs would have benefitted from more generations, as the slope of the plots indicates that progress has not halted yet. Second, we see the largest occupied hypervolume for the runs that involve the evolution of all hyperparameters and weights. On the other hand, there does not seem to be a significant difference in optimisation between evolving weights and thresholds or weights only. While this might seem strange at first, note that only the threshold of the non-adaptive output neuron is mutated, as Section III already mentioned. Apparently, the influence of the exact output neuron threshold is therefore minor.

The findings presented by Fig. 16 suggest that it might be a good idea for SNN learning frameworks to consider more than just the synaptic weights. By including hyperparameters such as thresholds and addition and decay constants during learning, it is likely that better results are obtained. Though it might not always be possible to integrate the tuning of these hyperparameters directly in the rule, it should certainly be preferred to modify them using an evolutionary algorithm, instead of optimising them by hand.

V. Conclusion

In this paper, we have demonstrated that evolved neuromorphic controllers are capable of real-world, continuous control in the form of landing MAVs based on divergence. While still making use of a conventional optical flow estimation method here, we have shown the feasibility of an interface between this work and that of [27], therefore arguing that our controller can be extended to an end-to-end-learnt, event-based spiking controller.

Furthermore, by minimising the amount of spikes during evolution, we have provided insight into the resources required for successfully solving the problem of divergence-based landing. As it turned out, SNNs with hidden layers of a single neuron or no hidden layer at all were also able to land smoothly, all the while doing so at a fraction of the energy (spikes) required. A study of the hyperparameters of the best-performing controllers allowed the promotion of smooth and transferable control policies through limitation of the mutation range of trace, threshold and voltage decay.

Finally, we have looked at evolving only a subset of the SNN's available hyperparameters. The outcome of this suggested that, as opposed to popular practice, better results can be obtained by including all hyperparameters in the optimisation process, instead of tuning these by hand or not tuning at all.

Future work should focus on achieving the previously described end-to-end-learnt pipeline, as well as incorporating energy minimisation into the popular NEAT framework. Expansion to other manoeuvres, vehicles and real-world environments should determine whether learning demonstrated here generalises well.

References

[1] G. C. H. E. de Croon, K. M. E. de Clercq, R. Ruijsink, B. Remes, and C. de Wagter, "Design, Aerodynamics, and Vision-Based Control of the DelFly," *International Journal of Micro Air Vehicles*, vol. 1, no. 2, pp. 71–97, Jun. 2009. [Online]. Available: https://doi.org/10.1260/175682909789498288

[2] M. Karásek, F. T. Muijres, C. D. Wagter, B. D. W. Remes, and G. C. H. E. de Croon, "A tailless aerial robotic flapper reveals that flies use torque coupling in rapid banked turns," *Science*, vol. 361, no. 6407, pp. 1089–1094, Sep. 2018. [Online]. Available: http://science.sciencemag.org/content/361/6407/1089

[3] K. Y. Ma, P. Chirarattananon, S. B. Fuller, and R. J. Wood, "Controlled Flight of a Biologically Inspired, Insect-Scale Robot," *Science*, vol. 340, no. 6132, pp. 603–607, May 2013. [Online]. Available: https://science.sciencemag.org/content/340/6132/603

[4] J. J. Gibson, *The Perception of the Visual World*. Boston: Houghton Mifflin Company, 1950.

[5] E. Baird, N. Boeddeker, M. R. Ibbotson, and M. V. Srinivasan, "A universal strategy for visually guided landing," *Proceedings of the National Academy of Sciences*, vol. 110, no. 46, pp. 18 686–18 691, Nov. 2013. [Online]. Available: https://www.pnas.org/content/110/46/18686

[6] C. Posch, T. Serrano-Gotarredona, B. Linares-Barranco, and T. Delbruck, "Retinomorphic Event-Based Vision Sensors: Bioinspired Cameras With Spiking Output," *Proceedings of the IEEE*, vol. 102, no. 10, pp. 1470–1484, Oct. 2014.

[7] G. Gallego, T. Delbruck, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza, "Event-based Vision: A Survey," *arXiv:1904.08405 [cs]*, Apr. 2019, arXiv: 1904.08405. [Online]. Available: http://arxiv.org/abs/1904.08405

[8] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, Dec. 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608097000117

[9] G. Orchard and R. Etienne-Cummings, "Bioinspired Visual Motion Estimation," *Proceedings of the IEEE*, vol. 102, no. 10, pp. 1520–1536, Oct. 2014.

[10] K. Roy, A. Jaiswal, and P. Panda, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, pp. 607–617, Nov. 2019. [Online]. Available: https://www.nature.com/articles/s41586-019-1677-2

[11] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks," *Neural Networks*, vol. 111, pp. 47–63, Mar. 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608018303332

[12] M. Bouvier, A. Valentian, T. Mesquida, F. Rummens, M. Reyboz, E. Vianello, and E. Beigne, "Spiking Neural Networks Hardware Implementations and Challenges: A Survey," *J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 2, pp. 22:1–22:35, Apr. 2019. [Online]. Available: http://doi.acm.org/10.1145/3304103

[13] M. Pfeiffer and T. Pfeil, "Deep Learning With Spiking Neurons: Opportunities and Challenges," *Frontiers in Neuroscience*, vol. 12, Oct. 2018. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6209684/

[14] N. Caporale and Y. Dan, "Spike Timing–Dependent Plasticity: A Hebbian Learning Rule," *Annual Review of Neuroscience*, vol. 31, no. 1, pp. 25–46, 2008. [Online]. Available: https://doi.org/10.1146/annurev.neuro.31.060407.125639

[15] S. M. Bohte, J. N. Kok, and H. La Poutré, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, no. 1, pp. 17–37, Oct. 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231201006580

[16] S. B. Shrestha and G. Orchard, "SLAYER: Spike Layer Error Reassignment in Time," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 1412–1421. [Online]. Available: http://papers.nips.cc/paper/7415-slayer-spike-layer-error-reassignment-in-time.pdf

[17] R. V. Florian, "Reinforcement Learning Through Modulation of Spike-Timing-Dependent Synaptic Plasticity," *Neural Computation*, vol. 19, no. 6, pp. 1468–1502, Apr. 2007. [Online]. Available: https://doi.org/10.1162/neco.2007.19.6.1468

[18] E. Vasilaki, N. Frémaux, R. Urbanczik, W. Senn, and W. Gerstner, "Spike-Based Reinforcement Learning in Continuous State and Action Space: When Policy Gradient Methods Fail," *PLOS Computational Biology*, vol. 5, no. 12, p. e1000586, Dec. 2009. [Online]. Available: https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1000586

[19] N. Frémaux, H. Sprekeler, and W. Gerstner, "Reinforcement Learning Using a Continuous Time Actor-Critic Framework with Spiking Neurons," *PLOS Computational Biology*, vol. 9, no. 4, p. e1003024, Apr. 2013. [Online]. Available: https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1003024

[20] T. S. Clawson, S. Ferrari, S. B. Fuller, and R. J. Wood, "Spiking neural network (SNN) control of a flapping insect-scale robot," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, Dec. 2016, pp. 3381–3388.

[21] Z. Bing, C. Meschede, G. Chen, A. Knoll, and K. Huang, "Indirect and direct training of spiking neural networks for end-to-end control of a lane-keeping vehicle," *Neural Networks*, vol. 121, pp. 21–36, Jan. 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608019301595

[22] F. Zhao, Y. Zeng, and B. Xu, "A Brain-Inspired Decision-Making Spiking Neural Network and Its Application in Unmanned Aerial Vehicle," *Frontiers in Neurorobotics*, vol. 12, 2018. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnbot.2018.00056/full

[23] K. Y. W. Scheper and G. C. H. E. de Croon, "Evolution of robust high speed optical-flow-based landing for autonomous MAVs," *Robotics and Autonomous Systems*, vol. 124, p. 103380, Feb. 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889019302404

[24] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: from architectures to learning," *Evolutionary Intelligence*, vol. 1, no. 1,

pp. 47–62, Mar. 2008. [Online]. Available: https://doi.org/10.1007/s12065-007-0002-4

[25] T. Bäck, D. B. Fogel, and Z. Michalewicz, *Handbook of Evolutionary Computation*. CRC Press, Jan. 1997. [Online]. Available: https://www.taylorfrancis.com/books/9780367802486

[26] D. B. Fogel, "The Advantages of Evolutionary Computation," in *Biocomputing and Emergent Computation: Proceedings of BCEC97*. World Scientific Press, 1997, pp. 1–11. [Online]. Available: http://dl.acm.org/citation.cfm?id=648178.749054

[27] F. Paredes-Vallés, K. Y. W. Scheper, and G. C. H. E. de Croon, "Unsupervised Learning of a Hierarchical Spiking Neural Network for Optical Flow Estimation: From Events to Global Motion Perception," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2019.

[28] J. R. Serres and F. Ruffier, "Optic flow-based collision-free strategies: From insects to robots," *Arthropod Structure & Development*, vol. 46, no. 5, pp. 703–717, Sep. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S146780391730066X

[29] B. Herissé, T. Hamel, R. Mahony, and F. Russotto, "Landing a VTOL Unmanned Aerial Vehicle on a Moving Platform Using Optical Flow," *IEEE Transactions on Robotics*, vol. 28, no. 1, pp. 77–89, Feb. 2012.

[30] G. C. H. E. de Croon, "Monocular distance estimation with optical flow maneuvers and efference copies: a stability-based strategy," *Bioinspiration & Biomimetics*, vol. 11, no. 1, p. 016004, Jan. 2016. [Online]. Available: https://doi.org/10.1088%2F1748-3190%2F11%2F1%2F016004

[31] H. W. Ho, G. C. H. E. de Croon, E. van Kampen, Q. P. Chu, and M. Mulder, "Adaptive Gain Control Strategy for Constant Optical Flow Divergence Landing," *IEEE Transactions on Robotics*, vol. 34, no. 2, pp. 508–516, Apr. 2018.

[32] B. J. Pijnacker Hordijk, K. Y. W. Scheper, and G. C. H. E. de Croon, "Vertical landing for micro air vehicles using event-based optical flow," *Journal of Field Robotics*, vol. 35, no. 1, pp. 69–90, 2018. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21764

[33] H. W. Ho and G. C. H. E. de Croon, "Characterization of Flow Field Divergence for MAVs Vertical Control Landing," in *AIAA Guidance, Navigation, and Control Conference*. San Diego, California, USA: American Institute of Aeronautics and Astronautics, Jan. 2016. [Online]. Available: http://arc.aiaa.org/doi/10.2514/6.2016-0106

[34] F. Galluppi, C. Denk, M. C. Meiner, T. C. Stewart, L. A. Plana, C. Eliasmith, S. Furber, and J. Conradt, "Event-based neural computing on an autonomous mobile platform," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 2862–2867, iSSN: 1050-4729.

[35] M. B. Milde, H. Blum, A. Dietmüller, D. Sumislawska, J. Conradt, G. Indiveri, and Y. Sandamirskaya, "Obstacle Avoidance and Target Acquisition for Robot Navigation Using a Mixed Signal Analog/Digital Neuromorphic Processing System," *Frontiers in Neurorobotics*, vol. 11, 2017. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnbot.2017.00028/full

[36] H. Blum, A. Dietmüller, M. Milde, J. Conradt, G. Indiveri, and Y. Sandamirskaya, "A neuromorphic controller for a robotic vehicle equipped with a dynamic vision sensor," in *Robotics Science and Systems, RSS 2017*. Berlin, Germany: Proceedings of Robotics: Science and Systems 2017, Jul. 2017. [Online]. Available: http://www.roboticsproceedings.org/rss13/p35.pdf

[37] J. Kaiser, J. C. V. Tieck, C. Hubschneider, P. Wolf, M. Weber, M. Hoff, A. Friedrich, K. Wojtasik, A. Roennau, R. Kohlhaas, R. Dillmann, and J. M. Zöllner, "Towards a framework for end-to-end control of a simulated vehicle with spiking neural networks," in *2016 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, Dec. 2016, pp. 127–134, iSSN: null.

[38] 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, 2018. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnbot.2018.00035/full

[39] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Machine Intelligence*, vol. 1, no. 1, pp. 24–35, Jan. 2019. [Online]. Available: https://www.nature.com/articles/s42256-018-0006-z

[40] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning," *arXiv:1712.06567 [cs]*, Apr. 2018, arXiv: 1712.06567. [Online]. Available: http://arxiv.org/abs/1712.06567

[41] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, and K. Kavukcuoglu, "Population Based Training of Neural Networks," *arXiv:1711.09846 [cs]*, Nov. 2017, arXiv: 1711.09846. [Online]. Available: http://arxiv.org/abs/1711.09846

[42] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution Strategies as a Scalable Alternative to Reinforcement Learning," *arXiv:1703.03864 [cs, stat]*, Sep. 2017, arXiv: 1703.03864. [Online]. Available: http://arxiv.org/abs/1703.03864

[43] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castañeda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, N. Sonnerat, T. Green, L. Deason, J. Z. Leibo, D. Silver, D. Hassabis, K. Kavukcuoglu, and T. Graepel, "Human-level performance in 3D multiplayer games with population-based reinforcement learning," *Science*, vol. 364, no. 6443, pp. 859–865, May 2019. [Online]. Available: https://science.sciencemag.org/content/364/6443/859

[44] P. Szczawinski, M. Duarte, S. Oliveira, and A. L. Christensen, "Toward Evolved Vision-based Control for a Quadrocopter," in *Proceedings of the 9th Conference on Telecommunications*, 2013, pp. 153–156.

[45] K. Y. W. Scheper and G. C. H. E. de Croon, "Abstraction, Sensory-Motor Coordination, and the Reality Gap in Evolutionary Robotics," *Artificial Life*, vol. 23, no. 2, pp. 124–141, May 2017. [Online]. Available: https://doi.org/10.1162/ARTL_a_00227

[46] D. Floreano, J.-C. Zufferey, and J.-D. Nicoud, "From Wheels to Wings with Evolutionary Spiking Circuits," *Artificial Life*, vol. 11, no. 1-2, pp. 121–138, Jan. 2005. [Online]. Available: https://doi.org/10.1162/1064546053278900

[47] R. Batllori, C. B. Laramee, W. Land, and J. D. Schaffer, "Evolving spiking neural networks for robot control," *Procedia Computer Science*, vol. 6, pp. 329–334, Jan. 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050911005254

[48] N. Takase, J. Botzheim, and N. Kubota, "Evolving spiking neural network for robot locomotion generation," in *2015 IEEE Congress on Evolutionary Computation (CEC)*, May 2015, pp. 558–565, iSSN: 1941-0026.

[49] J. Pérez, J. A. Cabrera, J. J. Castillo, and J. M. Velasco, "Bio-inspired spiking neural network for nonlinear systems control," *Neural Networks*, vol. 104, pp. 15–25, Aug. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608018301229

[50] A. Vandesompele, F. Walter, and F. Röhrbein, "Neuro-evolution of spiking neural networks on SpiNNaker neuromorphic hardware," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, Dec. 2016, pp. 1–6.

[51] H. Qiu, M. Garratt, D. Howard, and S. Anavatti, "Evolving Spiking Neural Networks for Nonlinear Control Problems," in *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, Nov. 2018, pp. 1367–1373.

[52] K. O. Stanley and R. Miikkulainen, "Evolving Neural Networks through Augmenting Topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, Jun. 2002. [Online]. Available: https://doi.org/10.1162/106365602320169811

[53] L. Salt, D. Howard, G. Indiveri, and Y. Sandamirskaya, "Parameter Optimization and Learning in a Spiking Neural Network for UAV Obstacle Avoidance Targeting Neuromorphic Processors," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–14, 2019.

[54] R. V. Florian, "Spiking Neural Controllers for Pushing Objects Around," in *From Animals to Animats 9*, ser. Lecture Notes in Computer Science, S. Nolfi, G. Baldassarre, R. Calabretta, J. C. T. Hallam, D. Marocco, J.-A. Meyer, O. Miglino, and D. Parisi, Eds. Berlin, Heidelberg: Springer, 2006, pp. 570–581.

[55] A. Soltoggio, P. Durr, C. Mattiussi, and D. Floreano, "Evolving neuromodulatory topologies for reinforcement learning-like problems," in *2007 IEEE Congress on Evolutionary Computation*, Sep. 2007, pp. 2471–2478, iSSN: 1941-0026.

[56] S. Nolfi and D. Floreano, *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. Scituate, MA, USA: Bradford Company, 2004.

[57] J. C. Bongard, "Evolutionary Robotics," *Communications of the ACM*, vol. 56, no. 8, pp. 74–83, Aug. 2013.

[58] F. Silva, M. Duarte, L. Correia, S. M. Oliveira, and A. L. Christensen, "Open Issues in Evolutionary Robotics," *Evolutionary Computation*,

vol. 24, no. 2, pp. 205–236, Nov. 2015. [Online]. Available: https://doi.org/10.1162/EVCO_a_00172

[59] S. Nolfi, J. Bongard, P. Husbands, and D. Floreano, "Evolutionary Robotics," in *Springer Handbook of Robotics*, ser. Springer Handbooks, B. Siciliano and O. Khatib, Eds. Cham: Springer International Publishing, 2016, pp. 2035–2068. [Online]. Available: https://doi.org/10.1007/978-3-319-32552-1_76

[60] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," in *Advances in Artificial Life*, ser. Lecture Notes in Computer Science, F. Morán, A. Moreno, J. J. Merelo, and P. Chacón, Eds. Berlin, Heidelberg: Springer, 1995, pp. 704–720.

[61] N. Jakobi, "Half-baked, Ad-hoc and Noisy: Minimal Simulations for Evolutionary Robotics," in *Fourth European Conference on Artificial Life*. MIT Press, 1997, pp. 348–357.

[62] K. Y. W. Scheper, "Abstraction as a Tool to Bridge the Reality Gap in Evolutionary Robotics," 2019. [Online]. Available: https://repository.tudelft.nl/islandora/object/uuid%3A389f453e-f7ff-4fea-a353-32755cf9a9e1

[63] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 23–30, iSSN: 2153-0866.

[64] A. Loquercio, E. Kaufmann, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Deep Drone Racing: From Simulation to Reality with Domain Randomization," *arXiv:1905.09727 [cs]*, May 2019, arXiv: 1905.09727. [Online]. Available: http://arxiv.org/abs/1905.09727

[65] J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, and S. Birchfield, "Training Deep Networks With Synthetic Data: Bridging the Reality Gap by Domain Randomization," 2018, pp. 969–977. [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2018_workshops/w14/html/Tremblay_Training_Deep_Networks_CVPR_2018_paper.html

[66] G. C. H. E. de Croon, H. W. Ho, C. De Wagter, E. van Kampen, B. Remes, and Q. P. Chu, "Optic-Flow Based Slope Estimation for Autonomous Landing," *International Journal of Micro Air Vehicles*, vol. 5, no. 4, pp. 287–297, Dec. 2013. [Online]. Available: https://doi.org/10.1260/1756-8293.5.4.287

[67] H. C. Longuet-Higgins and K. Prazdny, "The interpretation of a moving retinal image," *Proceedings of the Royal Society of London. Series B. Biological Sciences*, vol. 208, no. 1173, pp. 385–397, Jul. 1980. [Online]. Available: https://royalsocietypublishing.org/doi/abs/10.1098/rspb.1980.0057

[68] F. Paredes-Vallés, "Neuromorphic Computing of Event-Based Data for High-Speed Vision-Based Navigation," Master's thesis, Delft University of Technology, Delft, The Netherlands, 2018. [Online]. Available: http://resolver.tudelft.nl/uuid:aa13959b-79b9-4dfc-b5e0-7c501d9d3e2f

[69] E. Rosten and T. Drummond, "Machine Learning for High-Speed Corner Detection," in *Computer Vision – ECCV 2006*, ser. Lecture Notes in Computer Science, A. Leonardis, H. Bischof, and A. Pinz, Eds. Springer Berlin Heidelberg, 2006, pp. 430–443.

[70] B. D. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, ser. IJCAI'81, vol. 2. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1981, pp. 674–679, event-place: Vancouver, BC, Canada. [Online]. Available: http://dl.acm.org/citation.cfm?id=1623264.1623280

[71] J.-Y. Bouguet, "Pyramidal Implementation of the Affine Lucas Kanade Feature Tracker," 2000. [Online]. Available: http://robots.stanford.edu/cs223b04/algo_tracking.pdf

[72] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, "Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification," *Frontiers in Neuroscience*, vol. 11, Dec. 2017. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5770641/

[73] D. Zambrano and S. M. Bohte, "Fast and Efficient Asynchronous Neural Computation with Adapting Spiking Neural Networks," *arXiv:1609.02053 [cs]*, Sep. 2016, arXiv: 1609.02053. [Online]. Available: http://arxiv.org/abs/1609.02053

[74] S. J. Thorpe, A. Delorme, and R. Van Rullen, "Spike-based strategies for rapid processing," *Neural Networks*, vol. 14, no. 6, pp. 715–725, Jul. 2001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608001000831

[75] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of Physiology*, vol. 117, no. 4, pp. 500–544, 1952. [Online]. Available: https://physoc.onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1952.sp004764

[76] R. B. Stein, "A Theoretical Analysis of Neuronal Variability," *Biophysical Journal*, vol. 5, no. 2, pp. 173–194, Mar. 1965. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0006349565867091

[77] W. M. Kistler, W. Gerstner, and J. L. v. Hemmen, "Reduction of the Hodgkin-Huxley Equations to a Single-Variable Threshold Model," *Neural Computation*, vol. 9, no. 5, pp. 1015–1045, Jul. 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.5.1015

[78] G. Haessig, X. Berthelon, S.-H. Ieng, and R. Benosman, "A Spiking Neural Network Model of Depth from Defocus for Event-based Neuromorphic Vision," *Scientific Reports*, vol. 9, no. 1, p. 3744, Mar. 2019. [Online]. Available: https://www.nature.com/articles/s41598-019-40064-0

[79] K. Stewart, E. Neftci, G. Orchard, and S. B. Shrestha, "On-chip Few-shot Learning with Surrogate Gradient Descent on a Neuromorphic Processor," *arXiv:1910.04972 [cs]*, Oct. 2019, arXiv: 1910.04972. [Online]. Available: http://arxiv.org/abs/1910.04972

[80] P. U. Diehl, D. Neil, J. Binas, M. Cook, S. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *2015 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2015, pp. 1–8.

[81] A. Borst and M. Helmstaedter, "Common circuit design in fly and mammalian motion vision," *Nature Neuroscience*, vol. 18, no. 8, pp. 1067–1076, Aug. 2015. [Online]. Available: https://www.nature.com/articles/nn.4050

[82] J. O'Keefe and J. Dostrovsky, "The hippocampus as a spatial map: Preliminary evidence from unit activity in the freely-moving rat," *Brain Research*, vol. 34, pp. 171–175, 1971.

[83] Y. Jin, R. Wen, and B. Sendhoff, "Evolutionary Multi-objective Optimization of Spiking Neural Networks," in *Artificial Neural Networks – ICANN 2007*, ser. Lecture Notes in Computer Science, J. M. de Sá, L. A. Alexandre, W. Duch, and D. Mandic, Eds. Springer Berlin Heidelberg, 2007, pp. 370–379.

[84] N. G. Pavlidis, O. K. Tasoulis, V. P. Plagianakos, G. Nikiforidis, and M. N. Vrahatis, "Spiking neural network training using evolutionary algorithms," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 4, Jul. 2005, pp. 2190–2194 vol. 4.

[85] A. Belatreche, L. P. Maguire, M. Mcginnity, and Q. X. Wu, "Evolutionary design of spiking neural networks," *New Mathematics and Natural Computation*, vol. 02, no. 03, pp. 237–253, Nov. 2006. [Online]. Available: https://www.worldscientific.com/doi/abs/10.1142/S179300570600049X

[86] G. López-Vázquez, M. Ornelas-Rodriguez, A. Espinal, J. A. Soria-Alcaraz, A. Rojas-Domínguez, H. J. Puga-Soberanes, J. M. Carpio, and H. Rostro-Gonzalez, "Evolutionary Spiking Neural Networks for Solving Supervised Classification Problems," 2019. [Online]. Available: https://www.hindawi.com/journals/cin/2019/4182639/abs/

[87] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, Sep. 1999.

[88] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[89] C. Igel, N. Hansen, and S. Roth, "Covariance Matrix Adaptation for Multi-objective Optimization," *Evolutionary Computation*, vol. 15, no. 1, pp. 1–28, Mar. 2007. [Online]. Available: https://doi.org/10.1162/evco.2007.15.1.1

[90] Q. Zhang and H. Li, "MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, Dec. 2007.

[91] H. Li and Q. Zhang, "Multiobjective Optimization Problems With Complicated Pareto Sets, MOEA/D and NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 284–302, Apr. 2009.

[92] F.-A. Fortin, F.-M. D. Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary Algorithms Made Easy," *Journal of Machine Learning Research*, vol. 13, no. Jul, pp. 2171–2175, 2012. [Online]. Available: http://www.jmlr.org/papers/v13/fortin12a.html

[93] D. Neil, M. Pfeiffer, and S.-C. Liu, "Learning to be efficient: algorithms for training low-latency, low-compute deep spiking neural networks," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, ser. SAC '16. Pisa, Italy: Association for

Computing Machinery, Apr. 2016, pp. 293–298. [Online]. Available: https://doi.org/10.1145/2851613.2851724

[94] I. Flügge-Lotz, *Discontinuous Automatic Control*. Princeton University Press, 1953, google-Books-ID: aSzWCgAAQBAJ.

[95] A. Gidon, T. A. Zolnik, P. Fidzinski, F. Bolduan, A. Papoutsi, P. Poirazi, M. Holtkamp, I. Vida, and M. E. Larkum, "Dendritic action potentials and computation in human layer 2/3 cortical neurons," *Science*, vol. 367, no. 6473, pp. 83–87, Jan. 2020. [Online]. Available: https://science.sciencemag.org/content/367/6473/83

APPENDIX A
CASE CONFIGURATIONS

Table III lists the baseline configuration, denoted by *20-base* used during the simulations. SNN hyperparameters that are listed as genes are mutated during evolution, with the given values as initial values. If parameters differ per layer, they are stated separately. As mentioned before, θ_i is only mutated for non-adaptive LIF neurons, i.e., the output layer.

Table III
BASELINE CONFIGURATION (20-BASE)

Parameter	Value
N_{gen}	400
μ	100
λ	100
P_{mut}	0.3
Objectives	f_1, f_2, f_3
Genes	w_{ij}, α_{u_i}, α_{θ_i}, α_{x_i}, τ_{u_i}, τ_{θ_i}, τ_{x_i}, θ_i
g	9.81 ms^{-2}
h_0	$[2, 4, 6, 8]$ m
Environment bounds	$[0.05, h_0 + 5]$ m, 30 s
Hidden layer	20 neurons
Neuron type (hidden, output)	Adaptive LIF, LIF
θ_i	0.2
α_{u_i}	0.2
α_{θ_i}	0.2
α_{x_i}	1.0
τ_{u_i}	0.8
τ_{θ_i}	0.8
τ_{x_i}	0.8
Encoding	Neuron pairs for $\hat{D}$ and $\Delta\hat{D}$
η	1.0
Output bounds	$[-0.8, 0.5]$ g

Table IV states the variations per case with respect to the baseline configuration. *20-sm* looks at the effect of minimising spiking, while *20-sm-pu* and *20-sm-pc* study the use of place cell encodings for divergence (uniformly and cubically distributed centres, respectively). *1-sm* and *0-sm* investigate redundancy in terms of hidden neurons, and try to quantify the effect of limiting the mutation of addition and decay constants to a smaller range. The adjusted distributions from which these parameters are sampled can be found in Table V. *20-sm-wt* and *20-sm-w* consist of multiple subcases that test the effect of changes in non-mutated hyperparameters. The values of these parameters in *20-sm-wt-eq* and *20-sm-w-eq* are taken from an evolved baseline individual (see Fig. 4). *20-sm-wt-l/h* and *20-sm-w-l/h* then vary these parameters to see whether the changes can be compensated by mutating weights and thresholds or weights only, respectively. For *20-sm-wt-l* and *20-sm-w-l*, α_{v_i} of the output neuron was decreased by only 0.03 to prevent a 'dead' neuron. Additionally, θ_i of the output neuron was decreased by only 0.02 for *20-sm-w-l* to prevent a continuously spiking neuron.

Table IV
CONFIGURATION VARIATIONS PER CASE

Case	Parameter	Value
20-sm	Objectives	f_1, f_2, f_3, f_4
20-sm-pu, *20-sm-pc*	Objectives	f_1, f_2, f_3, f_4
	Encoding	Place cells for $\hat{D}$
	Observation bounds	$[-10, 10]$ s^{-1}
	Place cell centres	11
	Place cell spacing	uniform, cubic
	Place cell width	2 s^{-1}
1-sm, *0-sm*	Objectives	f_1, f_2, f_3, f_4
	Hidden layer	1, 0 neuron(s)
20-sm-wt-l, *20-sm-wt-eq*, *20-sm-wt-h*	Objectives	f_1, f_2, f_3, f_4
	Genes	w_{ij}, θ_i
	α_{u_i}, α_{θ_i}, α_{x_i}	-0.3, $=$, $+0.3$
	τ_{u_i}, τ_{θ_i}, τ_{x_i}	-0.2, $=$, $+0.2$
20-sm-w-l, *20-sm-w-eq*, *20-sm-w-h*	Objectives	f_1, f_2, f_3, f_4
	Genes	w_{ij}
	θ_i output	-0.02, $=$, $+0.2$
	α_{u_i}, α_{θ_i}, α_{x_i}	-0.3, $=$, $+0.3$
	τ_{u_i}, τ_{θ_i}, τ_{x_i}	-0.2, $=$, $+0.2$

Table V
SAMPLING DISTRIBUTIONS OF MUTATED PARAMETERS FOR 1/0-SM

Parameter	Distribution
α_{u_i}, α_{θ_i}, α_{x_i}	$\mathcal{U}(\alpha - 1/3, \alpha + 1/3)$, clamped to $[0, 1]$
τ_{u_i}, τ_{θ_i}, τ_{x_i}	$\mathcal{U}(\tau - 1/3, \tau + 1/3)$, clamped to $[0.3, 1]$

APPENDIX B
FLIGHT TEST CONFIGURATION

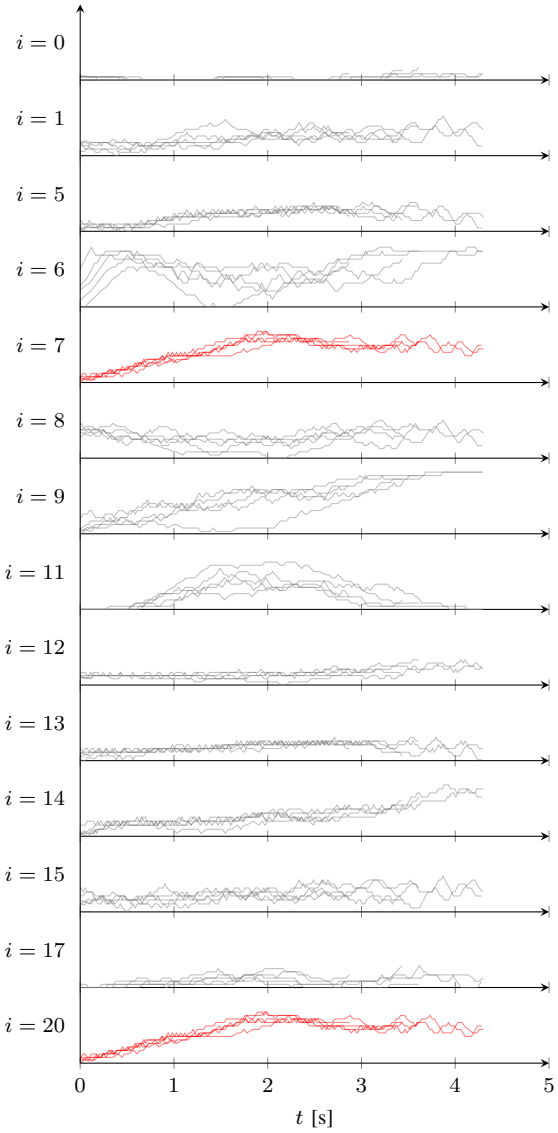
Table VI gives the configuration used for the real-world flight tests of selected individuals from *20-base*, *20-sm* and *1/0-sm*. The lower altitude bound is increased by 0.05 m compared to the simulations to account for the offset created by the MAV's landing legs at initialisation. Thrust effectiveness scales the output command of the PI controller to motor commands.

Table VI
FLIGHT TEST CONFIGURATION

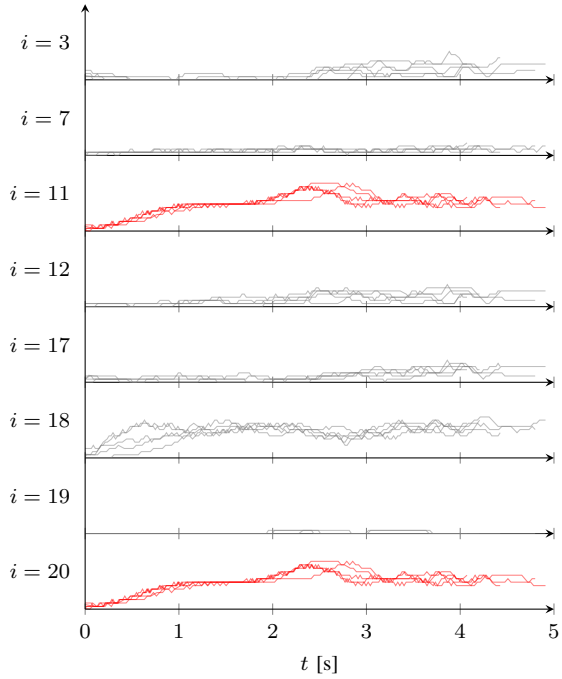
Parameter	Value
h_0	4 m
Altitude bounds	$[0.1, \rightarrow)$ m
Controller gains (P, I)	0.7, 0.3
Thrust effectiveness	0.05
N_D	100 pairs

APPENDIX C
20-BASE AND 20-SM: NEURON SPIKE TRACES

Fig. 17 displays the spike activity of each (active) neuron in the networks from *20-base* and *20-sm* during the five simulated runs given in Fig. 5. Each line is the result of a 20-step moving average over a neuron's emitted spikes. The red graphs indicate the activity of a 'responsible' neuron in the hidden layer and the output neuron, which correlate with $R^2 = 1$. Despite the spike minimisation taking place for *20-sm*, there still are some neurons whose firing contributes little to nothing. It is believed that this can be eliminated through further optimisation.



(a) *20-base*: not optimising for spike rate.

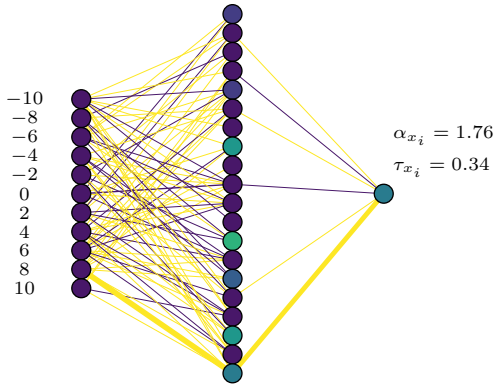


(b) *20-sm*: optimising for minimal spike rate.

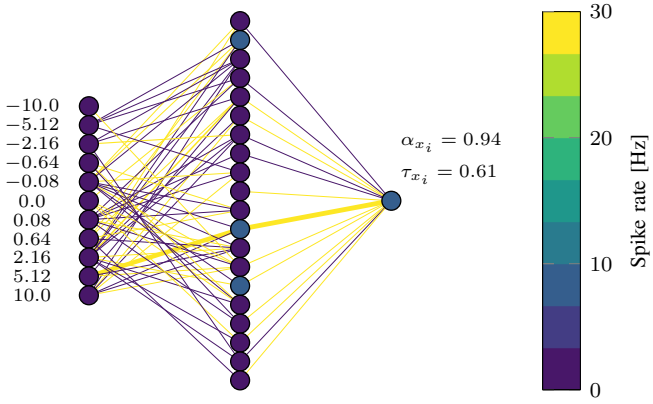
Figure 17. Spike trace (moving average over 20 steps) of active neurons in the hidden layer and output layer of selected individuals from *20-base* and *20-sm* for the five simulated runs displayed in Fig. 5. Neurons are counted downwards from the top of the hidden layer, with neuron $i = 20$ indicating the output neuron (see Fig. 6). Red traces correspond to the responsible hidden neuron and the output neuron that correlate with $R^2 = 1$.

APPENDIX D
20-SM-PU/PC AND 1/0-SM: NETWORKS

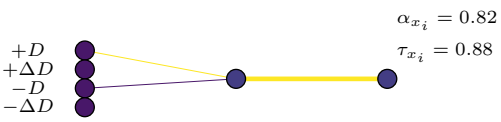
Fig. 18 gives the network structure and activity of selected individuals from *20-sm-pu/pc* and *1/0-sm* for the five simulated runs displayed in Figs. 10 and 13, respectively. The labels next to the input neurons indicate the thing they encode: a specific value of divergence in the case of *20-sm-pu* and *20-sm-pc*, and positive/negative (derivative of) divergence for *1-sm* and *0-sm*. The given values of α_{x_i} and τ_{x_i} represent the addition and decay constants of the spike trace of each output neuron (following Eq. (12)).



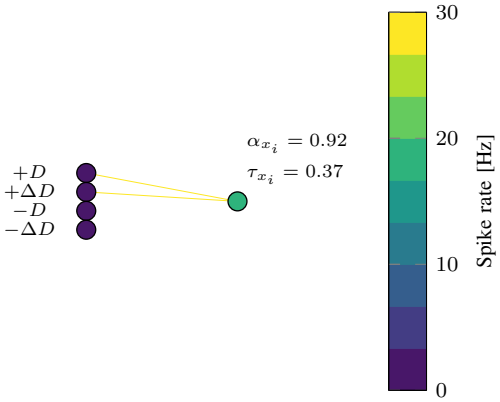
(a) *20-sm-pu*: uniformly distributed place cell encoding.



(b) *20-sm-pc*: cubically distributed place cell encoding.



(c) *1-sm*: a single hidden neuron.

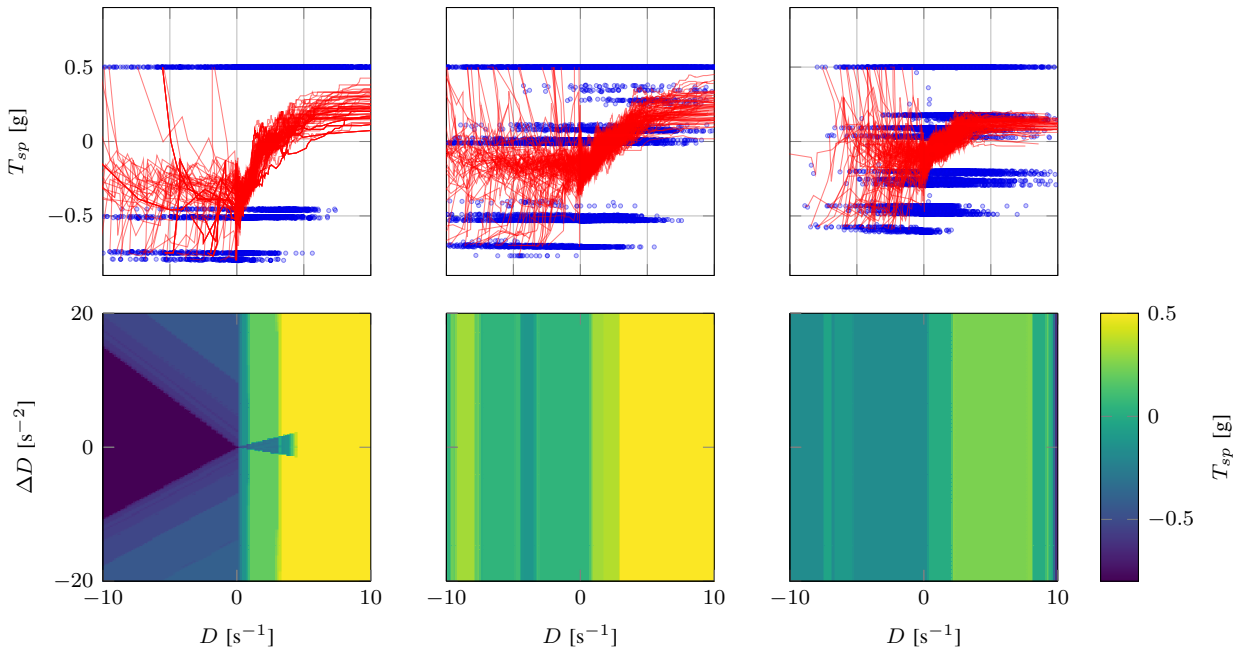


(d) *0-sm*: no hidden neuron.

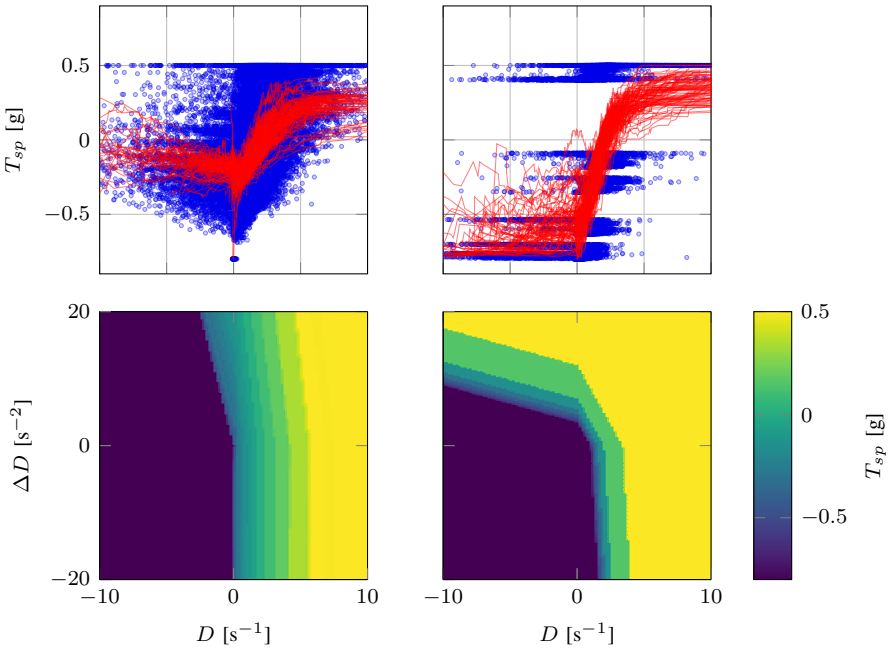
Figure 18. Average firing rates (in Hz) and synaptic weights of selected individuals from *20-sm-pu/pc* and *1/0-sm* for the five simulated runs displayed in Figs. 10 and 13, respectively. Vertex colour is proportional to neuron firing rate, while synaptic weight is directly proportional to edge weight in points. Edge colours indicate inhibitory (purple) or excitatory (yellow) synapses. Synaptic connections with a weight $|w_{ij}| < 0.05$ are not shown. The labels indicate the centres (in terms of divergence) of each place cell (*20-sm-pu/pc*), or the quantity encoded by each neuron (*1/0-sm*).

APPENDIX E
20-SM, 20-SM-PU/PC AND 1/0-SM: RESPONSES

Fig. 19 visualises the transient response (red lines, blue dots) and steady state response (surface plots) of selected individuals from *20-sm*, *20-sm-pu/pc* and *1/0-sm*.



(a) *20-sm* (left), *20-sm-pu* (middle) and *20-sm-pc* (right).



(b) *1-sm* (left) and *0-sm* (right).

Figure 19. Steady-state and transient response of selected individuals from *20-sm*, *20-sm-pu/pc* and *1/0-sm* as given in Figs. 9 and 12. Steady-state response is obtained by subjecting the SNNs to 100 time steps of the same observation and subsequently averaging the last 50 steps. The transient response is made up of 100 simulated landings during which observed divergence D and thrust setpoint T_{sp} are recorded (blue dots), which are then sorted by increasing divergence and passed through a 40-step moving average (red lines).

II

Literature Study

2

Optical Flow Control of MAVs

When the eye moves relative to the visible environment, a moving pattern of light hits the retina. The resulting distribution of apparent velocities of these patterns is called *optical flow*, and it supplies crucial information about the ego-motion of the observer, but also about the 3D structure of the visual scene (Gibson, 1950). For instance, a high-motion patch surrounded by a low-motion region would indicate a nearby obstacle, and an expanding flow with the point of expansion in front would result from forward motion (Borst, Haag & Reiff, 2010). Replacing the eye with an artificial vision sensor, the concept remains the same, except the information about ego-motion could now be used for the control of MAVs when other information such as the global positioning system (GPS) is unavailable or too inaccurate. This chapter serves as an introduction to optical flow modelling and its application for control of MAVs. Section 2.1 presents the mathematical foundations of optical flow modelling, and introduces various methods for estimating optical flow with conventional cameras. Next, Section 2.2 lists some biologically-inspired applications of optical flow techniques with the goal of navigating some environment.

2.1 Optical flow modelling and estimation

This work uses the formulation by Longuet-Higgins and Prazdny (1980) to describe the mathematical model of optical flow. This description is based on the *pinhole camera model* for perspective projection, which assumes that 1) the camera aperture (opening that controls the amount of light entering the camera) can be characterised as a pinhole (point), and 2) the retina can be seen as a plane (the so-called image plane). These assumptions greatly simplify the equations for perspective projection, but note that they are invalid for cameras with a wide field-of-view, which need more advanced models (Scaramuzza & Fraundorfer, 2011).

2.1.1 The pinhole camera model

Figure 2.1 presents the optical flow formulation by Longuet-Higgins and Prazdny (1980). World point A, which has coordinates $(X,\, Y,\, Z)^\top$ in the observer's reference frame $OXYZ$, is projected onto the *focal/image plane* $(x,\, y)^\top$ as point a. The origin O defines the location of the *aperture* of the visual sensor, and the Z-axis is the optical axis or line-of-sight. The intersection o between this optical axis and the image plane is called the *principal point*, and it is exactly one focal length f away from the aperture O, such that $o = (0,\, 0,\, f)^\top = (0,\, 0,\, 1)^\top$ (where we have taken $f = 1$ for convenience).

Consider a monocular observer (such as a camera) moving through a static environment. In this arbitrary ego-motion, the observer's reference frame is subject to translational velocities $(U,\, V,\, W)$ and rotational velocities $(p,\, q,\, r)$ along and around its axes, respectively. The velocity components of A relative to the frame of the moving observer are then as follows:

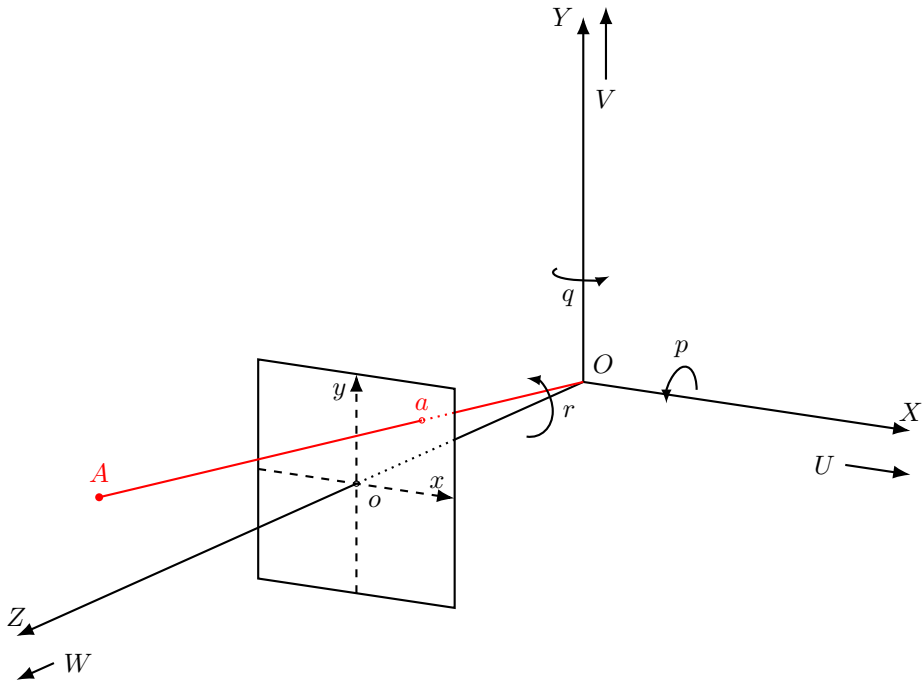


Figure 2.1: Pinhole camera model. Adapted from Longuet-Higgins and Prazdny (1980).

$$\begin{aligned} \dot{X} &= -U - qZ + rY \\ \dot{Y} &= -V - rX + pZ \\ \dot{Z} &= -W - pY + qX \end{aligned} \tag{2.1}$$

Obviously, the velocities of A are opposite to those of the observer, hence the minuses. The retinal position of A, a, can be derived from Figure 2.1 as $(x, y)^\top = (X/Z, Y/Z)^\top$. The ego-motion of the observer causes this point to move across the image plane (retina) with velocity $(u, v)^\top = (\dot{x}, \dot{y})^\top$. Subsequently, u and v can be computed from the time derivative of x and y respectively, expressed in terms of the velocity components of A:

$$\begin{aligned} u &= \dot{X}/Z - X\dot{Z}/Z^2 = (-U/Z - q + ry) - x\,(-W/Z - py + qx) \\ v &= \dot{Y}/Z - Y\dot{Z}/Z^2 = (-V/Z - rx + p) - y\,(-W/Z - py + qx) \end{aligned} \tag{2.2}$$

Or, written in another form:

$$u = u^T + u^R \qquad v = v^T + v^R \tag{2.3}$$

$$u^T = (-U + xW)/Z \qquad v^T = (-V + yW)/Z \tag{2.4}$$

$$u^R = -q + ry + pxy - qx^2 \qquad v^R = -rx + p + py^2 - qxy \tag{2.5}$$

Which implies that the optical flow of a point on the image plane can be split up into a translational and a rotational component.

2.1.2 Derivation of visual observables

With the optical flow of a point on the image plane now related to the ego-motion of an observer in a static environment, the next step is to determine this ego-motion and the structure of the environment. The assumption of a static environment implies that the observer's ego-motion states (p, q, r, U, V, W)

are equal for all world points, while depth Z varies for each of those points. Hence, by combining multiple world points, we could solve for depths and ego-motion. Because this is a complex and expensive computation, it is as of now irrelevant for high-speed visual navigation where on-board hardware is limited. Instead, summarising quantities regarding the observer's motion can be extracted by making use of a set of simplifying assumptions. These so-called *visual observables* are described and derived in the remainder of this section.

Derotation

If the observer has access, through other sensors, to its rotational rates (p, q, r), the rotational components of the optical flow can be corrected for. This process is called *derotation*, and the regular availability of rotational rate sensors (e.g., gyroscopes as part of the inertial measurement unit (IMU)) in MAVs means that it is common practice in applications like vision-based navigation (e.g., de Croon et al., 2013; Herissé, Hamel, Mahony & Russotto, 2012; Ho & de Croon, 2016; Ho, de Croon, van Kampen, Chu & Mulder, 2018; Izzo & de Croon, 2012; Pijnacker Hordijk, Scheper & de Croon, 2018).

Longuet-Higgins and Prazdny (1980) show that we can characterise the observer's ego-motion when the motion is either purely translational, or when the rotational component is small compared to the translational component (such as after derotation). To this end, the intersection of the observer's line of motion with the image plane is defined as:

$$x_0 = U/W \qquad\qquad y_0 = V/W \tag{2.6}$$

Assuming derotated flow and substituting Equation (2.6) into Equation (2.4), we may write the optical flow components as:

$$u = (x - x_0)\, W/Z \qquad\qquad v = (y - y_0)\, W/Z \tag{2.7}$$

From which follows that:

$$u/v = (y - y_0) \,/\, (x - x_0) \tag{2.8}$$

Looking at Equation (2.8), we can see that the point $(x_0,\, y_0)^\top$ on the image plane acts as a vanishing point of the optical flow field, having a null flow irrespective of the corresponding world point's depth. Because of the increasing magnitude of the flow vectors further away from this point, it is often referred to as the focus of expansion (FoE) in case of $+W$, or focus of contraction (FoC) otherwise. The location of this point on the image plane gives the observer a sense of motion direction. Furthermore, Equation (2.7) implies that the relative depth Z/W of all world points can be estimated with knowledge of the FoE's location. In case we have knowledge of the absolute depth of the FoE, we can compute the observer's *time-to-contact* (TTC) $\tau = Z/W$, which provides a measure of how fast the observer is approaching the FoE.

Planar flow

If, in addition to the assumption of a static scene, we consider it to be planar, the optical flow equations can be simplified even further. The derivation by de Croon et al. (2013), which is in turn based on the work by Longuet-Higgins and Prazdny (1980), results in expressions for the velocity components of this *planar flow field* in terms of the plane's slopes and the observer's normalised velocities. Let h define the distance to the planar surface along the observer's optical axis, and (α, β) the plane's slopes along the X and Y-axis of the observer, respectively. Along with the observer's depth-scaled velocities, $u_0 = U/h$, $v_0 = V/h$ and $w_0 = W/h$, the planar flow field's velocity components can be defined as:

$$\begin{aligned} u &= -u_0 + (\alpha u_0 + w_0)\, x + \beta u_0 y - \alpha w_0 x^2 - \beta w_0 xy \\ v &= -v_0 + \alpha v_0 x + (\beta v_0 + w_0)\, y - \beta w_0 y^2 - \alpha w_0 xy \end{aligned} \tag{2.9}$$

Which reduces to the following for negligible slopes of the planar surface (i.e., in case the surface is perpendicular to the optical axis):

$$\begin{aligned} u &= -u_0 + w_0 x \\ v &= -v_0 + w_0 y \end{aligned} \tag{2.10}$$

As is apparent from these equations, the normalised velocities (u_0, v_0, w_0) are essential visual cues in the perception of motion. Also known as *visual observables*, these cues are used for visual navigation and for the derivation of other observables, such as the already introduced TTC. Recalling the definition of TTC, we see that $\tau = Z/W = 1/w_0$. Next, following the mathematical definition of divergence:

$$\nabla \cdot (x, y) = \frac{\partial u}{\partial x}(x, y) + \frac{\partial v}{\partial y}(x, y) \tag{2.11}$$

Which, together with Equation (2.10), leads to the definition of *optical flow divergence* D as:

$$D = 2w_0 = \frac{2}{\tau} \tag{2.12}$$

Finally, the remaining visual observables u_0 and v_0 are the opposites of the *ventral flows* of the planar surface along its X and Y-axis, respectively: $\omega_x = -u_0$ and $\omega_y = -v_0$.

2.1.3 Estimation methods

As noted in the introduction to this chapter, optical flow is defined as the distribution of velocities of moving patterns of light hitting the eye's retina or a camera's sensor. In order to detect these velocities, brightness has to be measured at certain spatio-temporal intervals. In this section, we consider frame-based cameras that measure brightness periodically at all locations in their field-of-view.

The estimation of optical flow, according to Horn and Schunck (1981), is based on the so-called *brightness constancy constraint*, which assumes that the brightness of a local region in the image plane remains approximately constant under motion over short periods of time:

$$\nabla I \cdot \mathbf{u} + I_t = 0 \tag{2.13}$$

where $I(x, y, t)$ denotes the image intensity function, $\nabla I = (I_x, I_y)^\top$ and I_t its first-order derivatives, and $\mathbf{u} = (u, v)^\top$ the velocity of the image. As shown by Beauchemin and Barron (1995), this constraint leads to the *aperture problem* defined by Ullman (1979), which implies that only the motion component in the direction of the local image gradient may be estimated. This, in turn, may lead to ambiguities in the perceived flow if the gradient is not aligned with the direction of motion. Only at image locations with sufficient gradient structure, such as corners, can the flow be fully estimated. Figure 2.2 illustrates this phenomenon.

The various methods of optical flow estimation deal with the aperture problem in different ways. The remainder of this section will discuss the major classes to which these methods belong, following the distinction made by Beauchemin and Barron (1995). See Baker et al. (2011) for an extensive benchmark of many of these methods.

Gradient-based

Gradient-based estimation methods make use of the derivatives of the image intensity function (I_x, I_y, I_t) together with the brightness constancy constraint. The resulting aperture problem can be solved with a *local* approach, which uses a corner detector such as Harris or FAST (Harris & Stephens, 1988; Rosten & Drummond, 2006) to determine locations for which optical flow can be estimated reliably, or with a *global* approach, where dense optical flow is computed for the entire image and some other additional constraint, such as a smoothness constraint (e.g., Horn & Schunck, 1981), is imposed. The well-known local estimation method proposed by Lucas and Kanade (1981) is, despite its age, still the most-used approach for real-time optical flow estimation. Not surprisingly, its simplicity also makes it a very common choice for MAV-related applications (e.g., de Croon, 2016; de Croon et al., 2013; Ho & de Croon, 2016; Ho et al., 2018).

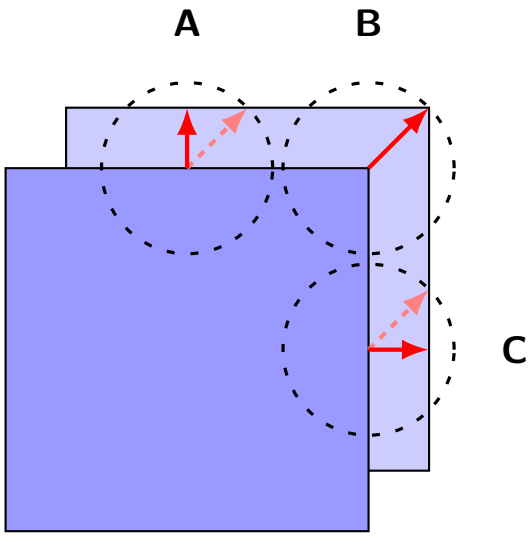


Figure 2.2: The aperture problem: a corner (aperture B) allows optical flow to be fully estimated, while edges (apertures A and C) only get you normal flows.

Correlation-based

As opposed to using sparse image features such as corners, correlation-based methods try to fit spatial shifts (d_x, d_y) such that the motion of contiguous image regions is best described. This approach has the advantage that, even in situations where good features are sparse, flow can be estimated. Camus (1997) found a way to decrease the rate of growth of the matching search space from quadratic to linear by searching in the time dimension instead of the spatial dimensions, which allowed for a real-time implementation. Kendoul, Fantoni and Nonami (2009) demonstrate that a similar improvement can be achieved through predicting pixel displacements with IMU data available on an MAV, allowing for vision-based navigation.

Frequency-based

Frequency-based methods employ velocity-tuned filters to estimate optical flow. By using orientation-sensitive filters in the Fourier domain, the motion of patterns seemingly random in the time domain may be extracted more readily from the frequency domain. Examples of frequency-based methods are the works by Adelson and Bergen (1985), Fleet and Jepson (1990). However, to the author's best knowledge, no frequency-based methods have been used in MAV-related applications due to their computational complexity.

2.2 Bio-inspired navigation with optical flow

Several authors have derived mathematical models from biological experiments that link the previously identified visual observables to vision-based manoeuvres performed by insects (e.g., Baird et al., 2013; Chahl, Srinivasan & Zhang, 2004; Srinivasan, Zhang, Lehrer & Collett, 1996). The remainder of this section introduces various navigation strategies implemented successfully in MAVs. In doing this, we follow the distinction made by Serres and Ruffier (2017) between navigation in the horizontal and vertical plane, and between the use of ventral flows, divergence and TTC for the latter.

2.2.1 Controlling flight speed and lateral position in corridors

Honeybees visually control their flight speed by means of optical flow (Baird, Srinivasan, Zhang & Cowling, 2005; Srinivasan et al., 1996). As they fly through a tapered tunnel, they decelerate when the tunnel narrows, and accelerate when it widens, in such a way that the bilateral optical flow (sum of the flows caused by the walls) stays constant. Keshavan, Gremillion, Escobar-Alvarez and Humbert (2014)

demonstrated this behaviour on a quadrotor MAV flying through various types of corridors.

In addition to this, bees have been observed to balance optical flows from the left and right part of their field-of-view, in order to control their lateral position in corridors. Known as the *optical flow balance hypothesis* (Srinivasan, Lehrer, Kirchner & Zhang, 1991), this way of navigating corridors has been implemented on a quadcopter MAV equipped with a 360-degree camera (Conroy, Gremillion, Ranganathan & Humbert, 2009) or a ring of optical flow sensors (Keshavan et al., 2014).

2.2.2 Terrain following and landing

Ventral flow

Honeybees were observed by Srinivasan et al. (1996) to perform grazing landings during which they keep the ventral flow, i.e., the velocity of the ground plane on the retina, constant. Chahl et al. (2004) have shown that results in a linear relation between forward and descent velocity, which causes the height above the surface to change over time as follows:

$$h\left(t\right) = h\left(t_0\right) \exp\left(-c\omega_x \left(t - t_0\right) \right) \tag{2.14}$$

While Chahl et al. (2004) implemented this solution using a fixed-wing MAV equipped with a downwards frame-based camera, Expert and Ruffier (2015), Ruffier and Franceschini (2005, 2015) employed a tethered rotorcraft MAV equipped with a neuromimetic EMD (see Section 3.1) to complete a variety of landing scenarios, including dealing with vertically oscillating and uneven terrain and landing on a moving platform.

However, as can be inferred from Equation (2.14), the ventral flow ω_x has to be nonzero to allow height control, which means that control of vertical dynamics and forward motion are coupled. MAVs that can control these two independently, such as quadcopters, cannot employ these strategies for hover and vertical landings, which are their main advantages over fixed-wing MAVs. To this end, the remainder of this section will deal with the visual observables that do allow this uncoupled vertical control.

Divergence

Baird et al. (2013) looked at the strategies used by honeybees when landing on vertical surfaces, and discovered that they keep divergence constant when approaching these surfaces. Recalling the definition of divergence, $D = 2w_0 = 2/\tau$, we can see that this strategy can indeed be used without any horizontal motion. Note, however, that a true constant-divergence landing will never lead to actually reaching the surface ($h \to 0$ would make $D \to \infty$), and some kind of landing procedure will have to be triggered in order to land.

Herissé et al. (2012) made a quadcopter MAV land on a moving platform using the constant-divergence strategy. Optical flow was estimated with a downward-looking, frame-based camera and the Lucas-Kanade algorithm (Lucas & Kanade, 1981). Meanwhile, de Croon (2016), Ho and de Croon (2016), Ho et al. (2018), Pijnacker Hordijk et al. (2018) look at the vertical oscillations that arise from using divergence for vertical control of MAVs. For a certain fixed control gain, performing a constant-divergence landing will lead to *self-induced oscillations* at a certain height above the surface. Thus by detecting these oscillations, the height above the surface can be inferred, and the control gain can be tuned during descent in order to ensure a smooth landing.

Time-to-contact

While the case of constant divergence is equivalent to keeping TTC τ constant, landing can also be achieved by constantly decreasing TTC. This strategy is actually applied by many animals, as has been demonstrated by Lee (1976), Lee, Davies, Green and van der Weel (1993): pigeons keep the *rate-of-change* of TTC, $\dot{\tau}$, constant during landing, and human drivers do the same when performing braking manoeuvres. This strategy results in a slightly different landing trajectory than the one described by

Equation (2.14) for constant divergence. Setting k as the desired $-\dot{\tau}$, the height above the surface evolves as follows (Izzo & de Croon, 2012):

$$h(t) = h(t_0)\left(k\frac{t}{\tau(t_0)} + 1\right)^{1/k} \tag{2.15}$$

Alkowatly, Becerra and Holderbaum (2014) used this relation to perform vertical landings using a quadrotor MAV equipped with a downwards frame-based camera. Furthermore, Armendariz, Becerra and Bausch (2019) performed vertical landings on moving platforms using a constantly decreasing TTC for vertical control. However, their experiments were only carried out in simulation.

3

Event-Based Vision Sensors & Optical Flow

So far, the discussed optical flow estimation methods and applications mainly involved conventional frame-based cameras. Therefore, this chapter will introduce event-based vision sensors and optical flow estimation methods. Section 3.1 will deal with the former, while Section 3.2 will discuss the latter, along with some applications. For a comprehensive overview of the entire field of event-based vision, refer to Gallego et al. (2019).

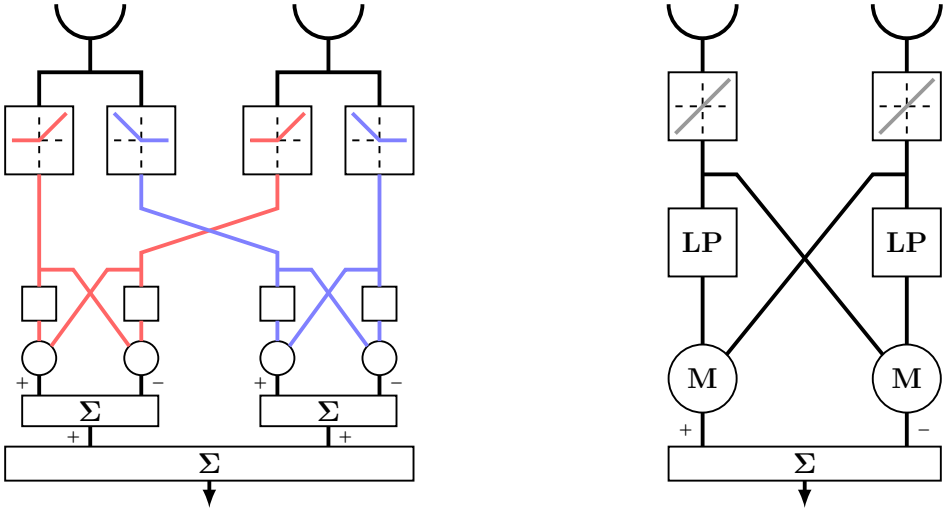
3.1 Event-based vision sensors

Conventional cameras operate in a frame-based manner: by periodically measuring brightness at all pixel locations, the world is perceived as a sequence of static images. Because the rate at which these frames are acquired is independent of the actual motion perceived, we will end up with a lot of redundant information in regions where motion is small, while we would prefer a higher frame rate in fast-changing regions.

Event-based vision sensors deal with these problems by reacting *asynchronously* to brightness changes and registering them as *events*, where asynchronous refers to the fact that these the time periods between consecutive events can be different across the field-of-view. The output of these sensors is then essentially a stream of events encoding image intensity variations at a particular time and location in the pixel array. This event-based manner of operation has actually been inspired by biological vision systems, which operate analogously by quantifying changes in the visual scene's brightness through spiking neurons in the retina (Posch et al., 2014).

Prior to understanding how event-based vision sensors work, we should have a look at arguably the simplest way of modelling the biological retina: the EMD. Conceived by Hassenstein and Reichardt (1956), these sensors work by multiplying input signals from two neighbouring photoreceptors after one of the incoming signals has been delayed by a low-pass filter. By performing this operation symmetrically and subtracting the outputs of both, a direction selectivity is achieved. A different model was proposed by Franceschini, Riehle and Le Nestour (1989), which prevents interactions between signals of opposite sign, so no cancellation can happen. This model has actually proven to be quite efficient at estimating optical flow with decent accuracy (Ruffier & Franceschini, 2005, 2015). See Figure 3.1 for a visual comparison of both models, and refer to the work by Eichner, Joesch, Schnell, Reiff and Borst (2011) for a more in-depth comparison.

Whereas a single EMD only has two photoreceptor cells, and can thus only quantify motion in a single direction, event-based vision sensors are comprised of much more of these photoreceptors, allowing, in principle, for a much more detailed motion estimate. The remainder of this section will further discuss the working principle of the event-based vision sensors, and compare some of their variants.



(a) Franceschini detector model. **(b)** Reichardt detector model.

Figure 3.1: Schematic overview of the EMD models covered by Eichner, Joesch, Schnell, Reiff and Borst (2011). Each having two photoreceptor cells on top, the signal goes through various low-pass filters (squares) and multiplications (circles) before it is summed to provide a notion of motion.

3.1.1 Working principle

The working principle of event-based vision sensors will be discussed on the basis of the Dynamic Vision Sensor (DVS), developed by iniVation[1] and described in the works by Lichtsteiner, Posch and Delbruck (2008), Posch et al. (2014).

Each DVS pixel produces a continuous-time spatio-temporal representation of the visual dynamics in its field-of-view by detecting *relative changes* in brightness. This is achieved through photoreceptors, which generate a voltage proportional to the logarithm of the perceived brightness. Whenever the brightness change in a pixel exceeds a predefined threshold C, a signed asynchronous event is generated, as is illustrated by the following equation:

$$|\,\Delta \log\left(I\left(x, y, t\right)\right)\,| > C_{ON/OFF} \tag{3.1}$$

Depending on whether the relative brightness change, with the reference set to the brightness level at the last event, is positive or negative, the *polarity* of the generated event is ON $(+1)$ or OFF (-1), respectively. Figure 3.2 shows how brightness changes over time lead to generated events. Following the address-event representation (AER) protocol (see, e.g., Lichtsteiner et al., 2008; Posch et al., 2014), each event consists of its location on the pixel array $(x,\, y)^\top$, its time stamp t, and its polarity P.

Apart from its asynchronous pixel readout, the DVS has several other advantages over conventional frame-based cameras. The nature of the photoreceptors acting as pixels means that brightness changes can be quantified at a very high temporal resolution (1 µs) and with a very low latency (12 µs). Also, the sparse activity of these pixels greatly reduces power consumption to an average of 23 mW, much less than comparable frame-based cameras (Lichtsteiner et al., 2008).

3.1.2 Variants and comparison

However, the DVS obviously also comes with its limitations. For instance, it lacks knowledge of absolute brightness levels, and its pixel array, at 128×128, is rather limited. To deal with this, several other event-based vision sensors have been developed in previous years. Following the overview by Posch et al. (2014):

[1] `https://inivation.com/`

- Asynchronous Time-based Image Sensor (ATIS): Described by Posch, Matolin and Wohlgenannt (2011), ATIS has a larger pixel array of 304×240 and the ability to measure absolute brightness, which allows for capturing conventional images with very high dynamic range.
- Dynamic and Active-pixel Vision Sensor (DAVIS): Combines an improved version of the DVS with a frame-based camera, which would be compatible with conventional computer-vision techniques, but also reintroduces the problem of data redundancy (Brandli, Berner, Yang, Liu & Delbruck, 2014; Posch et al., 2014).
- (Miniature) Embedded DVS (eDVS and meDVS): The eDVS and the miniature meDVS version were developed to allow use in cases where weight constraints are tight. As opposed to the 120-gram DVS, the 16-gram eDVS and 2.2-gram meDVS (Conradt, 2015) would be ideal for event-based navigation of lightweight MAVs, such as the DelFly (e.g., de Croon et al., 2009; Karásek et al., 2018).

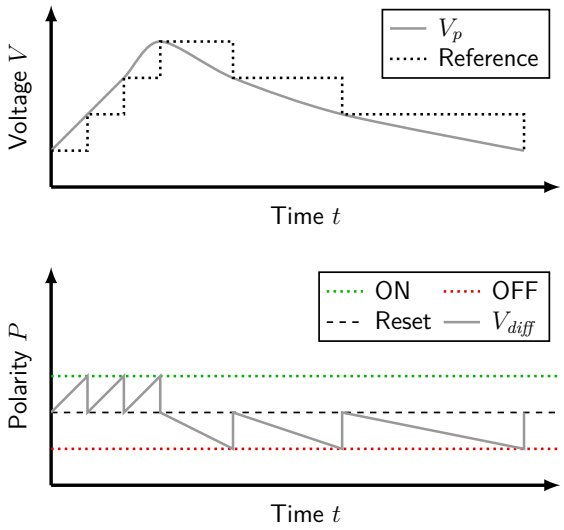


Figure 3.2: Working principle of a DVS pixel. If voltage V_p of a pixel changes more than a predefined threshold C relative to the previous reference, an event is generated, with its polarity dependent on the direction of the voltage change. Adapted from Lichtsteiner, Posch and Delbruck (2008).

3.2 Event-based optical flow

Analogous to frame-based cameras, the event-based cameras described in the previous section can be used to perceive ego-motion through optical flow. However, where frame-based approaches compare brightness levels in subsequent frames, the events registered by event-based cameras already include information on local brightness changes. The remainder of this section will introduce event-based optical flow estimation methods that can benefit from this, as well as some of their applications. The comparison of the various algorithms follows in part from the overview and benchmark by Rueckauer and Delbruck (2016).

3.2.1 Estimation methods

Event-based Lucas Kanade

Recall from Section 2.1.3 that gradient-based methods, such as the Lucas-Kanade algorithm (Lucas & Kanade, 1981), use the *spatio-temporal derivatives* of the image intensity function (I_x, I_y, I_t) together with the brightness constancy constraint (Horn & Schunck, 1981) for optical flow estimation. Benosman, Ieng, Clercq, Bartolozzi and Srinivasan (2012) accommodated this approach for event-based use

by computing the derivatives with a backward finite difference method, due to the fact that absolute brightness levels are absent.

A slightly different approach was taken in the more recent work by Brosch, Tschechne and Neumann (2015). Here, it is shown that the approach by Benosman et al. (2012) mixes first- and second-order derivatives due to the fact that generated events already represent the temporal derivative I_t. By rewriting the brightness constancy constraint in terms of the second-order partial derivatives (I_{tx}, I_{ty}, I_{tt}), and employing a central difference scheme for the spatial derivatives, a more consistent estimation of optical flow could be obtained.

However, after experiments carried out by Brosch et al. (2015), Rueckauer and Delbruck (2016), it seems that gradient-based approaches in general suffer severely from the fact that events generated in a single location are sparse. They conclude that these approaches are not suited for use with event-based cameras.

Spatio-temporal plane fitting

An alternative to the event-based Lucas-Kanade algorithm was proposed by Benosman, Clercq, Lagorce, Ieng and Bartolozzi (2014). Considering that a sequence of registered events makes up a 3D point cloud in spatio-temporal space, a surface could be fitted to any number of these points, the gradient of which can be used to compute optical flow. Thus, by fitting a plane to an incoming event's neighbours, its optical flow components can be estimated. Benosman et al. (2014) performs this fit using linear least-squares, and only considering events of equal polarity. Assuming constant local velocity makes the flow estimation robust against noise and compensates for absent events in the neighbourhood.

Experiments by Benosman et al. (2014) demonstrated decent accuracy when estimating optical flow. However, Brosch et al. (2015), Rueckauer and Delbruck (2016) argue that this approach can lead to inaccuracies for moving horizontal or vertical edges, because the threshold used to deal with the vanishing x- or y-gradient that is present in this case results in an angular quantisation. The adaptation proposed by Rueckauer and Delbruck (2016) produces more accurate velocity estimates, even in the case of horizontal or vertical edges.

Direction-selective filters

Similar to the frequency-based estimation methods introduced in Section 2.1.3, the approach presented by Tschechne, Sailer and Neumann (2014) employs spatio-temporal filters that are direction-selective and built from combinations of simpler spatial and temporal filters. By tuning the parameters of these filters using experimental data from cells in the human visual cortex, and building up an extensive *filter bank* that allows to detect different orientations and magnitudes, optical flow can be estimated. Brosch et al. (2015) reduced the ambiguity caused by the aperture problem by carrying out a response normalisation. Nevertheless, although these methods are able to robustly quantify the direction of motion, they have problems determining the optical flow magnitude (Brosch et al., 2015; Tschechne et al., 2014), since precision is very dependent on the extent of the filter bank.

ANN-based approaches

More recently, ANNs have been used to process event-based optical flow. Zhu, Yuan, Chaney and Daniilidis (2018) developed EV-FlowNet, an encoder-decoder architecture which allows learning of dense optical flow (meaning for each point in the field-of-view). It does this in a self-supervised manner, making use of a pair of greyscale images generated by the DAVIS camera immediately before and after an event to give a prediction of its local flow, allowing a loss function to be constructed. Zhu, Yuan, Chaney and Daniilidis (2019), on the other hand, learn event-based optical flow, ego-motion and depth from the event stream only. By processing a sequence of events and predicting their motion in time, they are able to remove blur, ending up with a robust set of events representing optical flow. Like Zhu et al. (2018), an encoder-decoder architecture is used, but here the encoder tries to create a meaningful representation of the set of input events, and the decoder uses this representation to reconstruct the deblurred set of events. The method presented by Ye, Mitrokhin, Fermüller, Yorke and Aloimonos

(2018) is very similar (also in performance), except for the fact that they make use of two separate networks for optical flow estimation, and that they assume a rigid scene.

3.2.2 Applications

Using the plane-fitting algorithm proposed by Benosman et al. (2014) and improved by Brosch et al. (2015) as a basis, Pijnacker Hordijk et al. (2018) presented two more improvements aimed at boosting the efficiency of the method, as well as its sensitivity to a larger range of optical flow magnitudes. This was achieved through 1) reducing the number of parameters of the fitted plane, and 2) achieve adaptive time windows by clustering recent events. Furthermore, the authors developed a procedure for determining flow divergence based on grouping optical flow components with similar orientations together, thus circumventing the aperture problem. They consequently used this estimation in a constant-divergence landing control loop implemented on board a quadcopter MAV equipped with a DVS.

Instead of grouping similarly oriented flow vectors, Akolkar, Ieng and Benosman (2018) deal with the aperture problem by performing *spatial pooling* of erroneous local flows, and mathematically prove that this approach can correctly estimate the true optical flow orientation. They demonstrate the performance of their algorithm using a car-mounted ATIS, and show that it can be used to capture flow direction more accurately compared to Benosman et al. (2014). Furthermore, the algorithm was applied to predict future poses of pedestrians, again showing improved performance.

Somewhat similar to this, the experiments conducted by Conradt (2015) include a simple optical flow estimation method that tries to fit patterns corresponding to three motions of an MAV (pan, tilt and yaw) to the first-order spatial derivatives of perceived events (computed similarly to Benosman et al. (2012)). The results obtained by implementing this algorithm on board a quadcopter MAV equipped with two meDVSs are, unfortunately, only evaluated qualitatively, and no control is performed whatsoever.

An ANN-based application can be found in the work by Sanket et al. (2019), which focuses on learning event-based optical flow for dynamic obstacle avoidance with MAVs. The authors present various encoder-decoder ANNs each with their own task in the processing pipeline, such as deblurring and denoising, estimating background motion and performing segmentation. Especially the deblurring/denoising network draws the attention, given that it allows relatively shallow networks trained in simulation to perform well in the real world, which is blurry and noisy.

4

Reinforcement Learning

The idea of learning by interacting with the environment seems to underlie most forms of learning and intelligence in the world around us. Exploring the connection between ourselves and world provides us with a wealth of information about achieving goals and the consequences of actions. *RL* is a computational approach to this kind of learning, which is aimed at accumulating some notion of *reward* by trying actions in a complex and uncertain environment, and discovering which ones are the most profitable. First, Section 4.1 will explore the biological foundation and application of RL, both from a psychological and a neuroscientific perspective. Next, Section 4.2 will largely follow the structure of the book by Sutton and Barto (2018), which can be regarded as the defining text for the field of RL, and highlight some of the most important aspects and distinctions of RL. After this, Section 4.3 will look at the implications of using RL in robotics, as well as some of the applications specifically relevant to this thesis.

4.1 Reinforcement learning in biology

Although RL, as described in this thesis, is an abstract computational framework that explores idealised situations, many of its fundamentals were inspired by biological theories of behaviour. Furthermore, as Sutton and Barto (2018) note, RL algorithms have contributed to the understanding of experimental data and even the development of new models of animal learning, both on the organism-level (psychology), as well as the neuron-level (neuroscience). The remainder of this section will explore each of these in turn.

4.1.1 Psychology

Throughout the 20th century, thousands of experiments on animal learning were conducted, and it is mainly here that the fields of RL and psychology touch. For instance, RL makes a distinction between algorithms for *prediction* and algorithms for *control*, where prediction is concerned with predicting quantities (such as reward) that depend on how an agent's environment is expected to change in the future, while control consists of actually adjusting behaviour as to produce reward (or avoid punishment) more frequently. These definitions show striking similarities with those of classical/Pavlovian conditioning, and instrumental/operant conditioning, respectively, of which prominent examples are the works by Pavlov (1927), Skinner (1938, 1963), Thorndike (1898). The temporal-difference (TD) algorithm for prediction (see Section 4.2.4 for an explanation of it) is actually a generalisation of the influential Rescorla-Wagner model (Rescorla & Wagner, 1972), and can account for some details of animal learning behaviour. As far as the control aspect is concerned, learning by trial-and-error is at the very base of both RL and Thorndike's *law of effect* (Thorndike, 1911), where we note that, for both animals and algorithms, this process does not have to be a merely blind one of just randomly trying out actions (exploration), but also consists of finding rewarding actions and connecting them to certain situations or states, and consequently exploiting these more often. Skinner (1958) explored a mechanism he called *shaping*, which involved progressively altering reward contingencies (making the problem

increasingly difficult) to approximate a certain desired behaviour. As we will see in Section 4.2.8, this approach is very effective for training RL agents.

The law of effect requires the ability to reinforce past actions based on current rewards, something which is very much related to the *credit-assignment problem* for learning systems (Minsky, 1961): how to distribute credit for success among the many decisions that may have been involved in producing it? The RL algorithms presented in Section 4.2 have two basic mechanisms for addressing this problem, namely the use of *eligibility traces* and the use of *TD methods* to learn value functions (see Section 4.2.4 for an explanation of both) that provide near-immediate evaluations of actions (in control) or immediate prediction targets (in prediction). Both these methods correspond to mechanisms of animal learning, namely *stimulus traces* (Hull, 1932, 1943; Pavlov, 1927) and *secondary reinforcement* (Hull, 1943), the latter of which may even have inspired the development of its artificial counterpart (Minsky, 1961).

While there are many more examples of the touching points between psychology, or animal learning in particular, and RL, these are of less relevance to this thesis. They can be found, however, in the book by Sutton and Barto (2018).

4.1.2 Neuroscience

As researchers are progressing in their understanding of the animal nervous system, they find more and more evidence that these systems implement algorithms that are remarkably similar to RL algorithms. One of these correspondences involves *dopamine*, one of the chemicals responsible for reward processing in animal brains, and which appears to convey *reinforcement signals* to parts of the brain responsible for learning and decision-making. These reinforcement signals are different from reward signals, as their function is to direct changes in an agent's valuations or decision-making (value function and policy, see Section 4.2.1), instead of simply classifying the desirability of some environmental state or action. This is very similar to the case of TD learning, where this reinforcement signal is called the *TD error* (discussed in Section 4.2.4). Based on experiments by neuroscientist Wolfram Schultz (e.g., Romo & Schultz, 1990; Schultz & Romo, 1990), Montague, Dayan and Sejnowski (1996) proposed the so-called *reward prediction error* (RPE) hypothesis of dopamine neuron activity, which states that one of the functions of dopamine is to encode the difference between old and new estimates of expected future reward, and to deliver this error to relevant areas of the brain, for example, when an animal is not expecting a certain rewarding event. See Schultz (1998, 2002), Schultz, Dayan and Montague (1997), Sutton and Barto (2018) for more details on dopamine and how it encodes the RPE. Dopamine and its role in learning are further discussed in Sections 5.2.1 and 5.2.4.

Experiments by O'Doherty et al. (2004), Takahashi, Schoenbaum and Niv (2008) demonstrate that parts of the animal brain implements something like an *actor-critic* structure, where the actor decides which actions to take based on valuations made by the critic. Both actor and critic would then use the TD error as a reinforcement signal, something that fits well with the above-mentioned function of dopamine. These actor-critic structures could also be implemented with ANNs, which are used to generalise in difficult learning problems. See Sections 4.2.6 and 4.2.7 for more about this.

Like in psychology, eligibility traces (or similar mechanisms) are also present in neuroscience. According to Klopf (1972), Klopf (1982), neurons that contribute to the firing of another neuron are temporarily marked, such that, in the case of a received reward, the neurons whose firing contributed to obtaining that reward can be made more likely to fire again in similar situations, in turn leading to more reward.

Again, more touching points between neuroscience and RL can be found in the book by Sutton and Barto (2018). A more detailed explanation of the various components of the nervous system and the terms used to describe them is given in Section 5.1.1.

4.2 Reinforcement learning basics

After initially introducing the elements underlying all of RL, this section will look at some important distinctions that can be made within the field of RL, and introduce many of its concepts and techniques in the process. In this, we heavily follow the book by Sutton and Barto (2018).

4.2.1 Elements

Problem definition

Finite Markov decision processes (MDPs) characterise the problem which RL is trying to solve: one of *sequential* decision making, where actions influence both immediate and future rewards. In this framework of learning from interaction to achieve a goal, the learner or decision maker is called the *agent*, while the thing it interacts with, encompassing everything that is not the agent, is called the *environment*. The continuous interaction between the two can be illustrated with the diagram given in Figure 4.1.

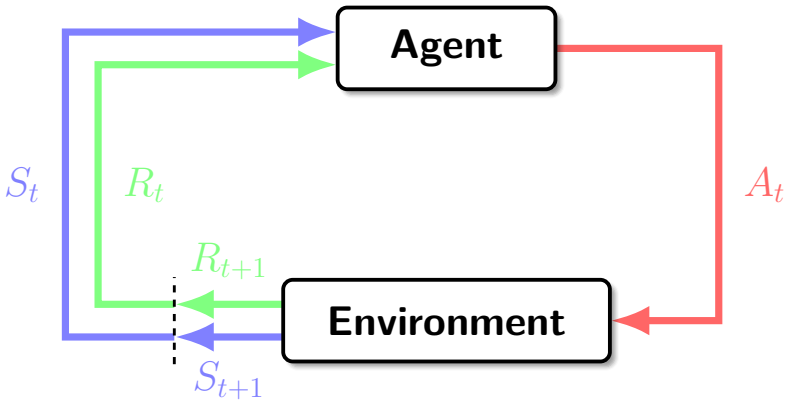


Figure 4.1: Interaction between agent and environment. Adapted from Sutton and Barto (2018).

At each discrete time step t, the agent perceives the environment's *state* $S_t \in \mathcal{S}$ and, based on this, takes an *action* $A_t \in \mathcal{A}(s)$. Consequentially, the next time step the agent receives a scalar *reward* $R_{t+1} \in \mathcal{R}$ from the environment, and finds itself in a new state S_{t+1}. Because of the finite nature of the MDP, the sets of states, actions and rewards ($\mathcal{S}$, $\mathcal{A}$, $\mathcal{R}$) all have a finite number of choices. Furthermore, the transition between subsequent states is probabilistic, such that:

$$p(s', r \mid s, a) \doteq \Pr\{S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a\} \quad \forall\, s', s \in \mathcal{S},\ r \in \mathcal{R},\ a \in \mathcal{A}(s) \tag{4.1}$$

In addition to this, MDPs have the so-called *Markov property*, which says that the transition to the next state S_{t+1} (and the received reward R_{t+1}) is completely determined by the preceding state and action, S_t and A_t, and not by any earlier states and actions.

Policy

A *policy* π defines the agent's behaviour at any given time step t. Put differently, it is a mapping from perceived states to probabilities of selecting each possible action to be taken in those states, denoted by $\pi(a \mid s)$, or, in case of deterministic choices, $\pi(s)$. This mapping can be represented in multiple ways, such as a lookup table or a search process.

Reward signal

The purpose or goal of an agent is formalised by means of the *reward signal*. At any time step t, the agent receives a scalar reward R_t from the environment. The sole objective of the agent is to maximise the total accumulated reward. And even though representing a complex goal by a scalar number

seems limiting, it has actually proved to be widely applicable and flexible (Sutton & Barto, 2018). For instance, in making an agent learn to escape a maze, we could simply let reward be -1 each time step, in order to stimulate fast escapes. The only thing we need to be concerned with is that the rewards we specify actually comply with the goal we want the agent to pursue.

The example described above uses a reward signal or function where reward depends only on time. Other forms are also possible: reward could be depending on the current state S_t, for example. Additionally, rewards can be dense or sparse, where the former gives back reward based on, e.g., the distance from a goal at each time step, while the latter only gives back reward when the goal or certain important milestones are reached. In general, sparse rewards lead to more difficult learning problems than dense rewards. This will be discussed in more detail in Section 4.2.8.

More formally, when speaking of maximising accumulated reward, we actually mean maximising *expected return*, which is some function of the reward sequence, such as the sum:

$$G_t \doteq R_{t+1} + R_{t+2} + \ldots + R_T \tag{4.2}$$

with T the final time step. While this makes sense in so-called *episodic* problems, where there is a natural final time step (such as solving the maze in the above example), this may be problematic in case there is no such terminal state. These continuing tasks need another definition of return G_t, since there is little sense in maximising something that could be infinite. Luckily, discounting comes to the rescue. By discounting future rewards, we can bound the return G_t for a discount rate $\gamma < 1$. Or, more formally:

$$G_t \doteq \sum_{k=0}^{T} \gamma^k R_{t+k+1} = R_{t+1} + \gamma G_{t+1} \tag{4.3}$$

where either $T = \infty$ or $\gamma = 1$, but not both. Note the recursive nature of G_t. Different discount rates result in agents with different characteristics: for $\gamma = 0$, we end up with an agent that is only concerned with maximising immediate rewards, while the agent becomes more farsighted as γ approaches 1.

From a biological perspective, reward may be seen as pleasure or pain. Analogous to a dog slowly changing its behaviour when receiving reward in the form of a treat, the reward signal forms the basis for changing the policy π: if a selected action is followed by a low or negative reward, the policy may be changed in order to select some other action in that state next time.

Value function

Whereas rewards give an indication of what is good in an immediate sense, the *value function* specifies what is good in the long run, and therefore presents a way to deal with the credit assignment problem mentioned in Section 4.1.1. The value of a state is the return an agent can expect to obtain over all future states, starting from that state. Thus, rewards can be seen as the immediate, intrinsic desirability of states, while values indicate their long-term desirability, taking into account states (and thus rewards) that are likely to follow. Maximising value will thus ultimately get you maximum return, and therefore we are most concerned with values when making and evaluating decisions.

More formally, the value function of a state s under a policy π is defined as the expected return when starting in s and following π afterwards:

$$v_\pi(s) \doteq \mathbb{E}_\pi\left[G_t \mid S_t = s\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right] \;\; \forall\, s \in \mathcal{S} \tag{4.4}$$

In a similar way, we can define the value of an action a in state s under policy π as the expected return when starting from s, taking a and following π afterwards:

$$q_\pi(s,a) \doteq \mathbb{E}_\pi\left[G_t \mid S_t = s,\, A_t = a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s,\, A_t = a\right] \;\; \forall\, s \in \mathcal{S},\; a \in \mathcal{A}(s) \tag{4.5}$$

From now on, we will refer to v_π as the *state-value* function, and to q_π as the *action-value* function. Note that both can be written in a recursive way, due to the recursive nature of the return G_t (see Equation (4.3). For v_π, we would end up with:

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \Big[r + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s'] \Big] \\ &= \sum_a \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a)[r + \gamma v_\pi(s')] \;\; \forall\, s', s \in \mathcal{S} \end{aligned} \tag{4.6}$$

which is known as the *Bellman equation* for v_π. Its variant for q_π can be derived similarly.

In solving an RL task, we are trying to maximise the agent's return, which implies choosing the actions that maximise value. The actions leading to this notion of maximum return are contained in the optimal policy π_*. There may be more than one π_*, but they all share the same optimal state-value and action-value functions:

$$v_*(s) \doteq \max_\pi v_\pi(s) \;\; \forall\, s \in \mathcal{S} \qquad\qquad q_*(s, a) \doteq \max_\pi q_\pi(s, a) \;\; \forall\, s \in \mathcal{S},\; a \in \mathcal{A}(s) \tag{4.7}$$

which must both satisfy their Bellman equation variants. Verifying this for v_* will result in its Bellman optimality equation:

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}(s)} q_*(s, a) \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s', r} p(s', r \mid s, a)[r + \gamma v_*(s')] \end{aligned} \tag{4.8}$$

where we have used that $v_*(s)$ is equivalent to $q_*(s, a)$ when taking the value-maximising action. Similarly, the Bellman optimality equation for q_* is:

$$\begin{aligned} q_*(s, a) &= \max_a \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \,\Big|\, S_t = s,\, A_t = a \Big] \\ &= \sum_{s', r} p(s', r \mid s, a) \Big[r + \gamma \max_{a'} q_*(s', a') \Big] \end{aligned} \tag{4.9}$$

4.2.2 Exploration versus exploitation

To accumulate as much reward as possible, an agent must prefer actions it has tried previously and found to be effective in producing reward. But to discover such actions, it has to try actions that were not selected before. In other words, the agent has to exploit what it already knows in order to accumulate reward, but it also has to explore in order to choose better actions next time. This trade-off, known as the *exploration versus exploitation dilemma*, is both central and unique to RL (as opposed to other kinds of learning) because of its trial-and-error nature. Going all-out on either exploration or exploitation will lead to failure for any RL problem, and thus a balance will have to be found.

While balancing exploration and exploitation in a sophisticated way is beyond the scope of this thesis, there is a simple approach to achieving a balance. First, consider a so-called *greedy policy*, where the chosen action always has the highest value, thus exploiting maximally:

$$\pi(s) \doteq \arg\max_a q_\pi(s, a) \tag{4.10}$$

Exploration is then equivalent to choosing non-greedy actions, as this improves the agent's estimate of the non-greedy action's value. We can balance the two by considering, at each time step, to take a random non-greedy action with probability ε. This is called an ε-greedy policy, and it comes with the nice theoretical guarantee that all q_π will converge to q_* for $T = \infty$. Obviously, ε needs to be selected by the user, its optimal value depending on the problem.

A slightly more sophisticated way of balancing exploration and exploitation is *upper confidence bound* (UCB) action selection, where the uncertainty of an action's currently estimated value, which may be high due to the fact that it has only been selected once, is also taken into account. A detailed explanation of the UCB approach can be found in the book by Sutton and Barto (2018).

4.2.3 Model-free versus model-based

Approximating optimal policies can be done in two ways: having a model of the environment and using this to evaluate certain policies and improving them, or generating experience within an environment and using it to estimate value functions and extract policies. The former is called a *model-based* approach, while the latter is known as *model-free*. We will discuss examples of each of them in turn below.

Model-based: dynamic programming

Dynamic programming (DP) refers to algorithms that can be employed to compute optimal policies given a perfect model of the environment as an MDP. This makes DP of limited utility for approaching actual RL problems, but it is still useful in the theoretical sense[1]. Given a finite MDP, whose dynamics are given through $p(s', r \mid s, a)$, and the Bellman optimality equations for v_* and q_* (given by Equations (4.8) and (4.9), respectively), DP is aimed at two things: evaluating policies (computing the value function) and, based on these evaluations, improving them. This approach is called *policy iteration*, and unfortunately it is an expensive computation, partly because policy evaluation is in itself an iterative computation. *Value iteration*, where one sweep of both policy evaluation and improvement are effectively combined in an iteration over the value function, often achieves faster convergence to an optimal policy.

Note that DP estimates the value of states based on estimates of the value of successor states. This updating of estimates based on other estimates is called *bootstrapping*, and it allows for very sample-efficient and reduced-variance learning at the cost of a slight bias (because state values are estimated based on partial instead of complete episodes). The full importance of bootstrapping will become clear in Section 4.2.4, but more background on DP can be found in Sutton and Barto (2018).

Model-free: Monte Carlo methods

As opposed to DP, *Monte Carlo* approaches can estimate value functions and discover optimal policies without perfect knowledge of the environment. It requires only experience in the form of sequences of states, actions and rewards from interactions (actual or simulated) with the environment. After each episode (we will only discuss Monte Carlo methods in terms of episodic tasks, i.e., those which have a natural final timestep or terminal state), action-value estimates are updated according to the experience just obtained. Estimating a state's value is done by averaging its returns over many episodes, and by keeping separate averages for each action taken in each state, we end up with an estimate of q_π instead of v_π, which is beneficial because we can extract the policy from it using Equation (4.10). The fact that Monte Carlo methods update action-value estimates only after an episode has finished means that no bootstrapping is going on, and that the estimates are unbiased. The downside of this is that these methods tend to experience high variance, which can only be countered by averaging over many episodes.

The process of trying to approximate optimal policies π_* by estimating q_* is called Monte Carlo *control*. One problem in these methods however, is that of maintaining sufficient exploration, since following

[1]The fact that DP is of limited use for solving real-world RL problems does not mean this holds for all model-based approaches. On the contrary, these methods are believed to be one of the key solutions to RL's sample inefficiency (meaning large amounts of experience are needed to achieve any kind of meaningful performance). See the work by Kaiser et al. (2019) for a comparison of model-free and model-based approaches for Atari games.

a deterministic policy will only get you experience for the states it visits. One solution is the use of exploring starts, or starting off with randomly sampled state-action pairs, but this might be difficult in problems where we work with real instead of simulated experience. Other solutions include using the ε-greedy approach from Section 4.2.2 in combination with on-policy learning, or off-policy learning. Both these concepts will be explained in Section 4.2.5.

4.2.4 Temporal-difference learning

TD learning is central to RL, and it forms a unifying framework that combines Monte Carlo ideas with DP ideas. TD methods can learn directly from experience, and have no need for a model of the environment's dynamics, just like Monte Carlo methods. TD methods update estimates based on other estimates, which is known as bootstrapping, just like DP does. The result of this is much lower variance compared to Monte Carlo methods, at the cost of a slight bias. As was already mentioned in Section 4.1, TD learning shows striking similarities with mechanisms found in the fields of psychology and neuroscience, and was partially inspired by those fields. This section will deal with TD prediction, or evaluating policies, only. TD control (approximating optimal policies) is introduced in Section 4.2.5.

Instead of waiting for an entire episode to finish and then updating the estimate of the value function, like Monte Carlo methods do, TD prediction can update this estimate every time step, while still converging to v_π in the mean if α is sufficiently small[2]. This method is called TD(0), or *one-step TD*, and comparing its update to the Monte Carlo update looks as follows:

$$\text{TD}(0): \quad V(S_t) \leftarrow V(S_t) + \alpha\left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right] \tag{4.11}$$

$$\text{Monte Carlo}: \quad V(S_t) \leftarrow V(S_t) + \alpha\left[G_t - V(S_t)\right] \tag{4.12}$$

with V the estimate of v_π and $\alpha \in (0, 1]$ the step size (sometimes called learning rate). Effectively, the target for the Monte Carlo update is the return G_t, while that for the TD update is $R_{t+1} + \gamma V(S_{t+1})$, which includes an estimate $V(S_{t+1})$ of $v_\pi(S_{t+1})$, and can thus be considered bootstrapping. The quantity within the square brackets of Equation (4.11) is called the *TD error*, denoted as δ_t, as it gives the error between the current and better estimate (target) of the value of S_t. Note that the TD error can be seen as a reinforcement signal, instead of a reward signal, as was discussed in Section 4.1.2.

In the spectrum between TD(0) and Monte Carlo methods there are those that make use of multiple steps to update value estimates. One such method is TD(λ), where λ refers to the use of an *eligibility trace*. Instead of basing the target on the next step only, as in TD(0), or on an entire episode, as with Monte Carlo methods, we take an exponentially weighted average of all steps, such that our target, now called the λ-return looks as follows:

$$G_t^\lambda \doteq (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n} \tag{4.13}$$

where $G_{t:t+n}$ is the expected return for the next n steps. According to this, the one-step return is given the largest weight, namely $1-\lambda$; the two-step return is given the next largest weight, $(1-\lambda)\lambda$, etc. For $\lambda = 1$ we would then end up with a Monte Carlo method, while $\lambda = 0$ would result in TD(0). It turns out that using multiple steps often achieves much faster convergence than TD(0), while being much less computationally complex than Monte Carlo methods. This is mainly due to the fact that bootstrapping works best if a significant state change has occurred. The way in which eligibility traces are formulated ensures that this is achieved at a minimum increase in computational complexity.

Figure 4.2 illustrates an eligibility trace, and suggests that it can be used in two ways. One way would be to use them to update current states based on future rewards and states. This is the so-called *forward view* employed by Equation (4.13) (and n-step TD, see Sutton and Barto (2018)). Another would be to consider them as a way to propagate the current TD error back in time and assign it to

[2]Or with probability 1 if α satisfies $\sum_{n=1}^{\infty} \alpha_n = \infty$ and $\sum_{n=1}^{\infty} \alpha_n^2 < \infty$, i.e., large enough to deal with initial conditions and noise and small enough to eventually converge.

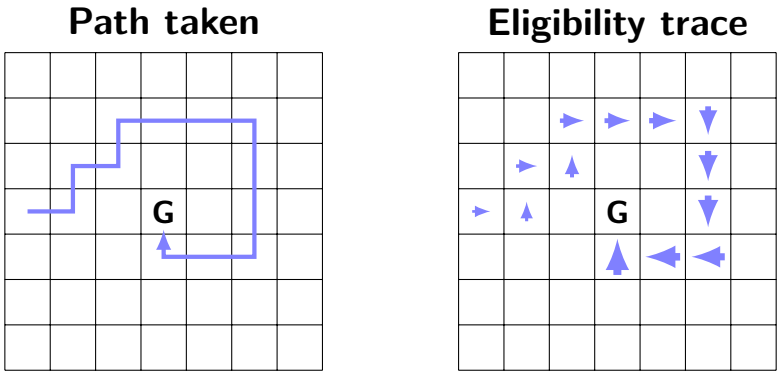


Figure 4.2: Illustration of an eligibility trace. The decreasing size of the arrows illustrates that the discovery of goal G affects values of recent states more, and those of earlier states less. Adapted from Sutton and Barto (2018).

prior states proportionally to how much they contributed to the current eligibility trace. Known as the *backward view*, this is the approach taken by TD(λ), and its way of using eligibility traces will be further discussed in Section 4.2.5. Like value functions, eligibility traces provide a way of dealing with the credit assignment problem that was mentioned in Section 4.1, with similar mechanisms found in both psychology and neuroscience.

4.2.5 On-policy versus off-policy control

As mentioned before, control can be defined as trying to approximate optimal policies based on estimated value functions. In this, we can make a distinction between on-policy and off-policy control. This distinction springs from the following dilemma: while control methods try to learn actions that can be seen as optimal behaviour, in order to discover these actions it has to behave non-optimally. The *on-policy* approach actually is a compromise to this dilemma: as discussed, it makes use of an ε-greedy policy where non-greedy actions are chosen with probability ε, and as such it actually approximates a near-optimal policy that still explores a bit. *Off-policy* learning, on the other hand, uses two policies: one that is learnt about and improved and which becomes the optimal policy, and one that is more exploratory and is used to generate experience. Respectively, these two are called the *target policy* and the *behaviour policy*. Because the experience from which off-policy methods learn is due to a different policy, learning is often characterised by greater variance and slower convergence than on-policy methods. However, off-policy methods are more general, and more powerful in the sense that they get stuck in local minima less often. As a matter of fact, on-policy methods are a special case of off-policy methods where the target and behaviour policy are the same (Sutton & Barto, 2018).

On-policy learning

An example of on-policy TD control is *Sarsa*[3]. It turns out that the convergence guarantees of TD(0) for the state-value function (see Section 4.2.4) carry over to the action-value function, and as such we can describe the Sarsa algorithm in Algorithm 1. By adding an eligibility trace to the Sarsa algorithm, we can increase convergence speed for many problems. This method, called *Sarsa(λ)*, is given in Algorithm 2.

Off-policy learning

On the other hand, *Q-learning* is an example of off-policy TD control, where the learnt action-value function Q directly approximates q_* independent of the followed policy. Its outline is given in Algorithm 3. Analogous to Sarsa(λ), this can be extended with eligibility traces to *Q(λ)*, which we will not show here. Another interesting extension that deserves a mention is the *double Q-learning* method by van Hasselt (2010), van Hasselt, Guez and Silver (2016), which aims to solve the overestimation

[3]Sarsa got its name from the quintuple of events (S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}) it uses to update the value function.

Algorithm 1 Sarsa: an on-policy TD control algorithm (Sutton & Barto, 2018).

Require:
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialise $Q(s, a)$ arbitrarily $\forall\ s \in \mathcal{S},\ a \in \mathcal{A}(s)$

loop for each episode
  Initialise S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  while S is not terminal **do**
    Take action A; observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma Q(S', A') - Q(S, A)\right]$
    $S \leftarrow S'$; $A \leftarrow A'$
  end while
end loop

Algorithm 2 Sarsa(λ) (Sutton & Barto, 1998).

Require:
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$, trace-decay rate $\lambda \in [0, 1]$
Initialise $Q(s, a)$ arbitrarily and $e(s, a) = 0\ \forall\ s \in \mathcal{S},\ a \in \mathcal{A}(s)$

loop for each episode
  Initialise S, A
  while S is not terminal **do**
    Take action A; observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$
    $e(S, A) \leftarrow e(S, A) + 1$
    for all S, A **do**:
      $Q(S, A) \leftarrow Q(S, A) + \alpha\delta e(S, A)$
      $e(S, A) \leftarrow \gamma\lambda e(S, A)$
    end for
    $S \leftarrow S'$; $A \leftarrow A'$
  end while
end loop

problem (or maximisation bias) of Q-learning that springs from it using the same action values for both selecting and evaluating an action. By learning two separate action-value functions, assigning experiences randomly to one of the two each time, we can decouple the selection from the evaluation, resulting in a more realistic estimate of the true action-value function and improved performance.

4.2.6 Tabular representation versus function approximation

So far, all introduced RL methods relied on tabular representations for their value function estimates. In this case, the methods can often find exact optimal value functions and policies, because all states can be visited enough times for learning. However, most of the real-world tasks for which we would like to apply RL involve such enormous state spaces, that these tabular representations become intractably large, not only in terms of memory needed to store them, but also in terms of filling them: almost every state encountered will never have been seen before. In cases like this we cannot expect to learn an optimal policy or value function even for infinite experience, and instead the goal becomes to find a good-enough approximate solution using limited computational expense.

Achieving this is a matter of generalisation: finding similarities between unvisited and visited states, and using this knowledge to improve our decision-making in these unvisited states. Combining existing

Algorithm 3 Q-learning: an off-policy TD control algorithm (Sutton & Barto, 2018).

Require:
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialise $Q(s, a)$ arbitrarily $\forall\ s \in \mathcal{S},\ a \in \mathcal{A}(s)$

loop for each episode
  Initialise S
  while S is not terminal **do**
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A; observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\,[\,R + \gamma \max_a Q(S', a) - Q(S, A)\,]$
    $S \leftarrow S'$
  end while
end loop

methods of generalisation, such as ANNs, with RL methods in order to achieve generalisation is called *function approximation*. Also in problems where only part of the state is observable, so-called *partially observable* problems, these function approximators provide a way to still attempt to solve them (by just leaving unobservable state variables out of the parametrisation).

To illustrate these methods, we will introduce an on-policy control method with function approximation. Furthermore, the dangers and limitations of function approximation are discussed.

On-policy control with function approximation

The goal of on-policy control is learning a parametric approximation of the optimal action-value function, such that $\hat{q}(s, a, \mathbf{w}) \approx q_*(s, a)$, where $\mathbf{w} \in \mathbb{R}^d$ is a finite-dimensional weight vector. Improving this approximation becomes a matter of updating $\mathbf{w}$ using gradient descent on the approximated value function (refer to Sutton and Barto (2018) for more information on gradient descent and its use here, or to Goodfellow, Bengio and Courville (2016) for the concept of gradient descent in general). As such, the weight update becomes:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\,[\,H_t - \hat{q}(S_t, A_t, \mathbf{w}_t)\,]\,\nabla \hat{q}(S_t, A_t, \mathbf{w}_t) \tag{4.14}$$

where ∇ is the gradient operator, $\alpha > 0$ the learning rate (called step size for tabular methods, but named learning rate here to stay consistent with gradient descent usage) and H_t the update target. In case of one-step Sarsa, this becomes $R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t)$, and we end up with the outline described by Algorithm 4.

The deadly triad

It turns out that the extension to function approximation is significantly harder for off-policy learning than it is for on-policy learning, leading to less robust convergence due to higher variance in many cases. According to Sutton and Barto (2018), off-policy learning and function approximation are part of the so-called *deadly triad* of RL, with the third component being bootstrapping. Combining all three, as is for instance done in Q-learning with function approximation, raises a significant challenge in terms of maintaining stability due to the loss of stability and convergence guarantees, and should be avoided if possible.

Despite this somewhat gloomy view, there is a glimmer of hope on the horizon. As is noted correctly by van Hasselt et al. (2018), the *deep Q-network* (DQN) by Mnih et al. (2013), Mnih et al. (2015) successfully learnt to play many Atari games while combining all three elements of the deadly triad. It is shown that improvements made to the Q-learning algorithm, such as bootstrapping on a separate network (Mnih et al., 2015) or correcting overestimation biases (van Hasselt et al., 2016), greatly reduce the chance of instability.

Algorithm 4 Sarsa with function approximation (Sutton & Barto, 2018).

Require:
Input: a differentiable action-value function parametrisation $\hat{q}(s, a, \mathbf{w})$
Algorithm parameters: learning rate $\alpha > 0$, small $\varepsilon > 0$
Initialise action-value weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

loop for each episode
  Initialise S, A
  loop for each step of episode
    Take action A; observe R, S'
    if S' is terminal **then**
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha [R - \hat{q}(S, A, \mathbf{w})] \nabla \hat{q}(S, A, \mathbf{w})$
      break
    end if
    Choose A' from S' using policy derived from $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε-greedy)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha [R + \gamma \hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})] \nabla \hat{q}(S, A, \mathbf{w})$
    $S \leftarrow S'$; $A \leftarrow A'$
  end loop
end loop

4.2.7 Direct policy search: policy gradient and actor-critic methods

So far, all discussed methods tried to learn the action-value function and selected actions based on their estimate of it. However, there are also methods that instead learn a parametrised policy that needs no value function estimate to select actions, although it might still be used in learning the policy's parameters. This approach of searching in the policy space directly is called *direct policy search*. If the parametrised policy is learnt using gradient ascent on some scalar performance measure $J(\boldsymbol{\theta})$, with $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ the policy's parameter vector, we speak of *policy gradient* methods[4]. Methods that learn approximations to both policy and value functions are often called *actor-critic* methods, where actor refers to the learnt policy, and critic to the learnt value function.

Policy gradient methods have various advantages over action-value methods. First, a parametrised policy can inherently approach a deterministic policy, whereas with ε-greedy selection over action values there is always a probability of selecting a random action. And while this could be mitigated by decreasing ε over time, choosing a suitable schedule for this decay requires quite some prior knowledge of the problem. Second, policy gradient methods have a natural way of approximating stochastic policies, where the optimal behaviour is to alternate multiple actions, because they allow learning probabilities with which to select certain actions. Action-value methods, which, in the case of ε-greedy, either select the action corresponding to the maximum value or a totally random one, lack this ability[5]. Third, in some problems the policy may simply be an easier function to approximate and parametrise than the action-value function. This ties in to findings by de Croon, Dartel and Postma (2005), who show that for a certain partially-observable MDP, the optimal policy may only be approximated by selecting certain combinations of state-action pairs, and that simply selecting the maximum value in each state (as action-value methods do) results in suboptimal performance. Fourth, policy gradient methods can naturally handle continuous action spaces by learning the statistics of the action probability distribution instead of probabilities for each action. And apart from all these practical advantages, there is also one important theoretical advantage: because the action values change smoothly as a function of the learnt policy parameter vector $\boldsymbol{\theta}$, there are much stronger convergence guarantees than there are for action-value methods.

Actor-critic methods are another way of extending TD approaches to control. In these methods, the value function is used as a bootstrapping critic, introducing bias but greatly reducing variance and accelerating learning. In other words: the actor decides on which actions to take based on the valuations

[4]Gradient-free policy search methods exist as well, and evolutionary algorithms could be considered an example of this.
[5]Example 13.1 in the book by Sutton and Barto (2018) perfectly illustrates this.

it receives from the critic. Like with TD(λ), adding eligibility traces allows us to determine the degree of bootstrapping. As was discussed in Section 4.1.2, actor-critic-like structures are also present in the animal brain. Algorithm 5 outlines an actor-critic method with eligibility traces. In addition to the value function parametrisation we already encountered in Algorithm 4, there is policy parametrisation $\pi(a \mid s, \boldsymbol{\theta})$ as well, whose output is the probability of selecting each possible action a in state s (in case of a discrete action space) or a continuous action (in case of a continuous action space whose statistics are represented by the policy parametrisation). Also note that two eligibility traces are involved, one for each parametrisation, and that these are updated with the gradient of the value function parametrisation and the gradient of the natural logarithm of the policy parametrisation[6]. The eligibility traces, in combination with the TD error, are then used to update the parameters in the value function and policy parametrisation.

Algorithm 5 Actor-critic with eligibility traces (Sutton & Barto, 2018).

Require:
Input: a differentiable policy parametrisation $\pi(a \mid s, \boldsymbol{\theta})$
Input: a differentiable state-value function parametrisation $\hat{v}(s, \mathbf{w})$
Algorithm parameters: learning rates $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$; trace-decay rates $\lambda^{\boldsymbol{\theta}} \in [0, 1]$, $\lambda^{\mathbf{w}} \in [0, 1]$
Initialise policy parameter vector $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ arbitrarily (e.g., to $\mathbf{0}$)

loop for each episode
  Initialise S
  $\mathbf{e}^{\boldsymbol{\theta}} \leftarrow \mathbf{0}$ (eligibility trace vector for $\boldsymbol{\theta}$)
  $\mathbf{e}^{\mathbf{w}} \leftarrow \mathbf{0}$ (eligibility trace vector for $\mathbf{w}$)
  $I \leftarrow 1$
  while S is not terminal **do**
    $A \sim \pi(\cdot \mid S, \boldsymbol{\theta})$ (sample action from stochastic policy)
    Take action A; observe R, S'
    if S' is terminal **then**
      $\delta \leftarrow R - \hat{v}(S, \mathbf{w})$
    else
      $\delta \leftarrow R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$
    end if
    $\mathbf{e}^{\boldsymbol{\theta}} \leftarrow \gamma \lambda^{\boldsymbol{\theta}} \mathbf{e}^{\boldsymbol{\theta}} + I \nabla \ln \pi(A \mid S, \boldsymbol{\theta})$
    $\mathbf{e}^{\mathbf{w}} \leftarrow \gamma \lambda^{\mathbf{w}} \mathbf{e}^{\mathbf{w}} + \nabla \hat{v}(S, \mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \delta \mathbf{e}^{\boldsymbol{\theta}}$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \mathbf{e}^{\mathbf{w}}$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$
  end while
end loop

4.2.8 Reward signal design

The fact that the goal of an RL system can be defined as a scalar reward signal is both a blessing and a curse. While, as opposed to supervised learning, RL does not rely on detailed knowledge of what an agent's correct actions should be, it does depend strongly on how well the goal of the system's designer can be represented by such a signal, and how well the signal assesses progress in reaching this goal. Designing a reward signal is thus a critical part of any RL application, and should not be overlooked.

While in some applications, such as playing a well-defined game, goals may be easily translated into reward signals, this can be much more difficult in other problems, for example designing a useful household robot. And even in cases where the goal is clearly identifiable, the problem of sparse rewards can

[6]The natural logarithm is used to condense the expression for the policy eligibility trace update: $\nabla \ln \pi(A \mid S, \boldsymbol{\theta}) = \frac{\nabla \pi(A|S,\boldsymbol{\theta})}{\pi(A|S,\boldsymbol{\theta})}$. This update stems from the REINFORCE algorithm, see Section 13.3 of Sutton and Barto (2018).

still arise. For in order to improve its estimated value function or policy effectively, the agent needs to frequently encounter nonzero rewards that steer it towards its goal.

In practice, this often leads to a trial-and-error search for a reward signal that produces acceptable results. One solution might be the administering of intermediate goal-directed rewards, as to make the reward function less sparse. This process of augmenting the reward structure, known as *reward shaping*, has been demonstrated to accelerate learning significantly (Kober, Bagnell & Peters, 2013; Laud & DeJong, 2002, 2003; Mataric, 1994; Ng, Harada & Russell, 1999). Ng et al. (1999) suggest shaping functions that involve a distance-based heuristic and a subgoal-based heuristic, and explain that reward shaping works because it provides a notion of closeness to the desired behaviour, instead of relying on a signal that only encodes success or failure.

Other solutions to the problem of sparse rewards include augmenting the approximated value function instead of the reward signal, for instance with guesses of what (parts of) the final function should be. This decreases the chance of the agent learning unintended behaviour (Sutton & Barto, 2018). Yet another approach would be to apply shaping in a gradual sense, as to make the problem increasingly harder (which is the shaping explored by Skinner (1958), as was explained in Section 4.1.1)[7].

In cases where an expert agent, such as a human, is present, observations of its behaviour may be used to augment the reward signal or value function (Smart & Kaelbling, 2002). This learning by observing an expert is called *apprenticeship learning* (Abbeel & Ng, 2004), and while it can be applied in the supervised sense as mentioned above, it can also be employed to extract a reward function from expert behaviour, a process called *inverse reinforcement learning* (Abbeel & Ng, 2004; Ng & Russell, 2000). This method turns out to be especially helpful in problems where manually constructing a reward function is too tedious and complex, such as autonomous driving.

Still other approaches include an automated trial-and-error search for acceptable reward signals, which involves defining a space of feasible candidates and applying an optimisation algorithm that evaluates the candidates according to some high-level scoring function that intends to encode the designer's true goal. Sorg, Lewis and Singh (2010) demonstrate that reward signals may even be improved in an online sense via gradient ascent, where the gradient is that of the high-level objective function.

4.2.9 Continuous time and space

Many interesting real-world tasks require smooth, continuous actions taken in response to high-dimensional observations. So far, the way to go has been to discretise time, state and action a priori, which allowed the application of a conventional RL algorithm[8]. However, this approach can have multiple undesirable consequences, such as rough control outputs due to coarse discretisation, or intractable learning due to fine discretisation. While function approximation can solve these problems for some applications, it might be insufficient for others. With this in mind, Doya (2000) formulated several RL algorithms suitable for continuous time and state problems based on the *Hamilton-Jacobi-Bellman* (HJB) equation, which is the continuous-time counterpart of the Bellman equation.

Learning the state-value function $v^{\pi}\left(\mathbf{s}\left(t\right)\right)$ of a certain policy π and with state $\mathbf{s}\left(t\right)$ in continuous time can be achieved by adjusting its estimate according to the continuous TD error:

$$\delta\left(t\right) \doteq R\left(t\right) - \frac{1}{\tau_{\delta}}\hat{v}\left(t\right) + \dot{\hat{v}}\left(t\right) \tag{4.15}$$

where $\hat{v}\left(t\right) = \hat{v}\left(\mathbf{s}\left(t\right), \mathbf{w}\right) \approx v^{\pi}\left(\mathbf{s}\left(t\right)\right)$ is the current parametrised estimate of the state-value function and τ_{δ} the TD error time constant. The corrections can be carried out using an exponential eligibility

[7] Another term for this approach to shaping could be *curriculum learning* (Bengio, Louradour, Collobert & Weston, 2009), where the learner is first presented easy examples of the task at hand, and the difficulty of the examples is increased gradually afterwards.

[8] RL algorithms solve *finite* MDP problems, meaning the state, action and reward space consist of a finite number of elements (Sutton & Barto, 2018). This implies that these spaces are discrete in nature, even though that may not be apparent at first sight.

trace, which will get us a continuous-time version of TD(λ). For an instantaneous TD error $\delta(t_0)$, this correction should be:

$$\Delta\hat{v}(t) = \begin{cases} \delta(t_0) \exp\left(\frac{t_0 - t}{\tau_\delta}\right) & \text{if } t \leq t_0 \\ 0 & \text{if } t > t_0 \end{cases} \tag{4.16}$$

The dynamics of the parameter and eligibility trace updates can then be given by:

$$\dot{w}_i = \alpha\delta(t) e_i(t) \tag{4.17}$$

$$\dot{e}_i(t) = -\frac{1}{\tau_e} e_i(t) + \frac{\partial v(\mathbf{s}(t), \mathbf{w})}{\partial w_i} \tag{4.18}$$

where α is the parameter learning rate and $0 < \tau_e \leq \tau_\delta$ the time constant of the eligibility trace. Now that we have a way to estimate the state-value function given a certain policy, we want to be able to improve that policy (control). One such way, described by Doya (2000), is a continuous actor-critic approach. Consider the policy implemented by the actor as:

$$\boldsymbol{\pi}(t) = s\left(A\left(\mathbf{s}(t), \mathbf{w}^A\right) + \sigma\mathbf{n}(t)\right) \tag{4.19}$$

where $A\left(\mathbf{s}(t), \mathbf{w}^A\right) \in \mathbb{R}^m$ is a function approximator with parameter vector $\mathbf{w}^A$, $\mathbf{n}(t) \in \mathbb{R}^m$ is noise, and $s(\cdot)$ is a monotonically increasing output function. We then update the parameters according to the *stochastic real-valued* (SRV) *unit* algorithm (Gullapalli, 1990):

$$\dot{w}_i^A = \alpha^A \delta(t) \mathbf{n}(t) \frac{\partial A\left(\mathbf{s}(t), \mathbf{w}^A\right)}{\partial w_i^A} \tag{4.20}$$

4.2.10 Game playing

As mentioned before, games represent the perfect problem for RL to solve, as their dynamics and goal are usually well-defined. Because of this, some of RL's largest breakthroughs have been achieved in game playing, some of the most interesting being:

- Learning to play many different Atari games at beyond-human level using a single architecture and configuration consisting of a deep *convolutional* ANN (CNN) acting as function approximator and a heavily modified Q-learning algorithm (together called DQN) to deal with the deadly triad from Section 4.2.6 (Mnih et al., 2013; Mnih et al., 2015).

- Mastering the game of Go using two different networks to evaluate board positions and moves, and combining these networks in a Monte Carlo search algortihm to select the best moves. By learning from either human experts and self-play (Silver et al., 2016) or self-play only (Silver et al., 2017), the algorithm was capable of beating the Go world champion, and could later be generalised to achieve a similar state-of-the-art performance in chess and shogi (Silver et al., 2018).

- Learning to play *Montezuma's Revenge*, an Atari game notorious for its sparse rewards and exploration and which proved too hard for DQN, by using *proximal policy optimisation* (PPO) (Schulman, Wolski, Dhariwal, Radford & Klimov, 2017) in combination with an exploration bonus. PPO's capability of handling two differently discounted reward streams (one for episodic and one for non-episodic[9] returns) together with a curiosity in the form of the exploration bonus made the agent capable of long-term decisions, achieving human-level performance (Burda et al., 2018).

[9] In this setting, an episode ends if the agent 'dies', meaning that non-episodic returns are those that carry over between game-overs. Burda, Edwards, Storkey and Klimov (2018) argue that providing these returns next to the episodic ones is a good way of stimulating exploratory behaviour, since encountering the end of an episode implies that future episodic rewards are zero, which might make agents overly risk-averse.

4.3 Reinforcement learning in robot control

At first sight, RL seems like a natural fit for robotics: it enables a robot to autonomously discover optimal behaviours by solely interacting with its environment. Instead of explicitly designing desired behaviours, engineers would only have to specify their goals in terms of a scalar reward signal. However, several aspects of robotics pose hard-to-overcome challenges to the application of RL in the field. This section will discuss some of these challenges (along with potential solutions), and highlight some relevant applications that were successful despite these difficulties.

4.3.1 Difficulties

Kober et al. (2013) have provided an excellent overview of the challenges of applying RL in robotics, and the remainder of this section will largely follow their distinctions.

Curse of dimensionality

While high-dimensional spaces are not confined to the field of robotics, the fact that robots operate in the real world makes it that these spaces are often continuous instead of discrete, posing an even greater challenge of generalisation. This curse of dimensionality (Bellman, 1957), where the number of dimensions leads to an exponential explosion of the number of states and actions, can be lifted in some problems through the use of function approximation, as was discussed in Section 4.2.6). Additional solutions include the use of preprogrammed actions that achieve some simple task (e.g., lift this leg) or structuring tasks into a hierarchical set of problems to be solved (Barto & Mahadevan, 2003).

Curse of real-world samples

Obtaining enough high-quality real-world experience to facilitate learning can be a difficult process because of the fact that operating in the physical world entails all kinds of external factors: wear and tear may change the dynamics of the robot over time (Sutton, Koop & Silver, 2007), environment settings such as light and temperature may not be constant, and so on. Ensuring safe exploration becomes a key issue because of hardware limitations (Moldovan & Abbeel, 2012), and operating in real-time means that accumulating experience cannot be sped up. Additionally, real-world sensors suffer from noise and delay, making the learning problem only partially observable. Solutions to difficulties like this include accurate simulated environments, function approximation, apprenticeship learning (see Section 4.2.8) and sample-efficient model-based learning methods.

Curse of under-modelling and model uncertainty

The previous part mentioned simulation as a possible solution to the difficult process of obtaining enough high-quality real-world data. Unfortunately, building an accurate simulator usually requires building an accurate model of the robot and its environment, which often needs very many real-world samples. Simplified or inaccurate models, as it turns out, often lead to policies that need heavy modification to be transferred to the real world successfully, unless the task is inherently stable (Kober & Peters, 2009). Still, approximate models may fulfil roles in, e.g., verification and unit testing.

Curse of goal specification

Perhaps the biggest challenge in applying RL to robotics is the specification of goals. While often dramatically simpler than specifying behaviour itself, translating poorly quantifiable goals to a scalar reward signal still requires a lot of work. Bing, Meschede, Röhrbein, Huang and Knoll (2018) make a distinction between three different approaches for specifying rewards: rewarding/punishing specific events (sparse rewards, see, e.g., Faghihi, Moustafa, Heinrich & Wörgötter, 2017), control error minimisation (e.g., error with another controller, see Clawson et al., 2016) or global metric minimisation (e.g., error with target angles, see Spüler, Nagel & Rosenstiel, 2015). Apart from this, the challenges and their solutions in terms of goal specification for robotics are very similar to those for general RL problems, which were already discussed in Section 4.2.8.

4.3.2 MAV control

To give an idea of the state-of-the-art of RL applied to MAV control, some recent work on this is collected here. Note that the focus is mainly on MAV landing, as this manoeuvre has proven to be the most tractable for RL to solve, and also seems to be the most relevant for this thesis.

Rodriguez-Ramos, Sampedro, Bavle, de la Puente and Campoy (2019) make use of a *deep deterministic policy gradient* (DDPG) algorithm, which is especially suited to the continuous spaces involved in robot control (Lillicrap et al., 2015), to smoothly land an MAV on a moving platform. Training was performed in the realistic robot simulator Gazebo[10] in order to allow smooth transfer to real-world testing. The authors also propose a Gazebo-based RL framework that allows the use of many algorithms, robots and environments, not unlike the work by Zamora, Lopez, Vilches and Cordero (2016). Unfortunately, MAV state is either provided directly by the simulation environment or OptiTrack[11], a precision motion capture and 3D tracking system, which poses a far smaller challenge than inferring altitude from visual sensors.

The work by Polvara et al. (2018), on the other hand, does make use of only visual sensors. Their controller, consisting of conventional and double DQNs, is capable of detecting a landing marker using a downwards-facing camera, navigating towards it, and landing on top of it. Training was performed in simulation, and the performance of the method carried over to real-world tests. However, flight control consisted of only seven discrete actions (such as forward, left, right, down), which greatly reduces the difficulty of the learning problem.

Rodriguez-Ramos, Sampedro, Bavle, Moreno and Campoy (2018) nevertheless demonstrate the ability to perform vision-based MAV landings on a marked moving platform with continuous control, allowed through the use of the DDPG algorithm. They also introduce a reward function that encourages smooth actions, and includes a shaping component, which informs the agent about its instantaneous progress (see Section 4.2.8).

[10] `http://gazebosim.org`
[11] `https://optitrack.com/`

5

Reward-Modulated Neuromorphic Computing

Coined by Carver Mead in the 1990s, the term *neuromorphic computing* refers to the use of very-large-scale integration (VLSI) of analog electronic circuits, which have the advantage of being orders of magnitudes more power-efficient than their digital counterparts (Mead, 1990), to mimic neurological architectures in biological nervous systems (Mead, 1989). More recently, neuromorphic computing has been used to describe the implementation of any system that aims to mimic the biological nervous system and its mechanisms for processing, learning and memorising.

The sections of this chapter will describe the components needed for such a system. SNNs are a type of ANN that more closely mimics networks of biological neurons, making them an ideal fit for neuromorphic computing. Section 5.1 will cover SNNs and their biological inspiration in detail. Learning in these networks can be achieved in multiple ways, some more biologically accurate than others. Given RL's biological foundation, the focus of Section 5.2 will be on reward-modulated learning, as inspired by the role of dopamine in the animal brain (see Section 4.1.2). Actual applications of neuromorphic computing, involving either hardware implementations or software simulations of SNNs, will be described in Section 5.3, with a focus on applications relevant to this thesis.

5.1 Spiking neural networks

Proposed by Maass (1997) as the third generation of ANNs, *SNNs* most closely mimic the biological networks of neurons found in the animal brain. Like their biological counterparts, data is sent through these networks as sparse sets of *spikes*, a characteristic which makes them computationally more powerful (Bouvier et al., 2019; Maass, 1997) as well as more energy efficient (Olshausen & Field, 2004; Pfeiffer & Pfeil, 2018; Rueckauer, Lungu, Hu, Pfeiffer & Liu, 2017; Tavanaei et al., 2019; Zambrano & Bohte, 2016) than the dense, non-spiking second generation (conventional ANNs, extensively described by Goodfellow et al. (2016)). These sparse sets of spikes allow (biological) spiking neurons to encode information temporally, which in turn gives them the ability to carry out high-speed computations that rapidly adapt to changes in input (Maass, 1997; Thorpe, Delorme & Van Rullen, 2001). The remainder of this section will look at the biological foundation of SNNs and the terms used to describe their architecture and functioning, as well as some of the neuron models that are in use.

5.1.1 Biological background

The book by Gerstner, Kistler, Naud and Paninski (2014) gives a selective description of the biological foundation of SNNs, perfectly fitting in the scope of this thesis. Therefore, we will largely follow their account here.

Neurons, the nervous system's specialised cells for processing and transmitting information, come in many different forms and shapes. Typically, however, they are made up of three functionally distinct parts: *dendrites*, *soma* (cell body) and *axon*, as illustrated in Figure 5.1. Dendrites branch from the soma and collect electric input signals, also called *action potentials* or spikes, coming from other neurons. The soma acts as a central processing unit and keeps track of the neuron's *membrane potential*: if the sum of input spikes coming from the dendrites exceeds some threshold, an output spike is generated. The neuron is then said to be *firing*. The axon delivers this output spike to other neurons.

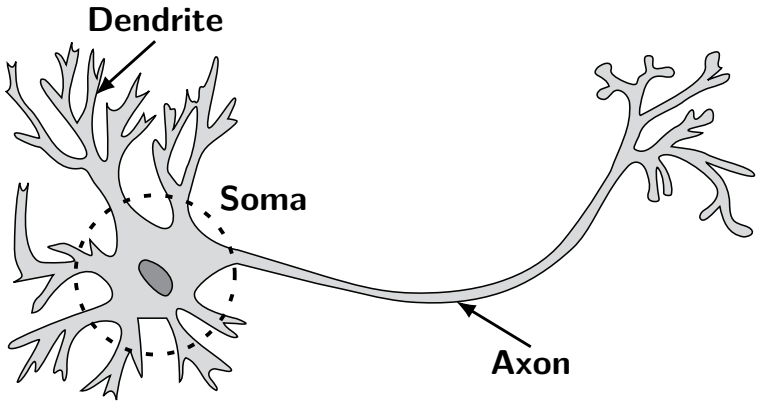


Figure 5.1: Schematic image of a biological neuron. Outline adapted from Wikimedia[1].

The junction between two neurons is called a *synapse*. In case a neuron sends an action potential across the synapse, we refer to the sending neuron as the *presynaptic* neuron, and to the receiving one as the *postsynaptic* neuron. Most synapses in the animal brain are of a chemical kind, where the electrical pulses on the presynaptic side are transferred to the postsynaptic side using a chemical called a *neurotransmitter*. The strength or effectiveness by which these pulses are transferred is the synapse's *efficacy*. This efficacy can be modified in certain ways, which is what allows learning (see Section 5.2.1). More details about the synapse and the transfer of spikes can be found in (Gerstner et al., 2014).

As said, an action potential is nothing more than a short electrical pulse. A sequence of these pulses emitted by a single neuron is called a *spike train*. Regulating the intervals between subsequent spikes is what allows these networks to encode information in the temporal domain. Little information in carried in the actual shape of the spike, since these are mostly constant. In case a neuron does not receive any incoming spikes, its membrane potential $u(t)$ remains at the rest potential u_{rest}. The arrival of a spike can have two effects on the membrane potential: a positive change (the synapse is said to be *excitatory*) or a negative change (the synapse is said to be *inhibitory*). Whether excitation or inhibition occurs depends on the neurotransmitter used. After this change, the membrane potential slowly decays back to its rest value. In case incoming spikes cause the neuron to fire, it enters into a *refractory period* of a few milliseconds, during which it is impossible to make the neuron fire again. This period of absolute refractoriness is followed by a period of relative refractoriness, during which it is difficult, but not impossible, to trigger another spike. Figure 5.2 shows the typical build-up of membrane potential as a result of presynaptic spikes by two input neurons.

5.1.2 Neuron models

The very nature of modelling implies that there are different possible models that can represent the same observed phenomenon, often with different levels of abstraction. The modelling of the biological neuron and its membrane potential in Figure 5.2 is no exception, and so various neuron models have been developed over the years.

The very first model of a biological neuron was developed by McCulloch and Pitts (1943) under the name threshold logic unit (TLU), although the term *perceptron* is more often used today. Non-spiking

[1] https://commons.wikimedia.org/wiki/File:Neuron_Hand-tuned.svg

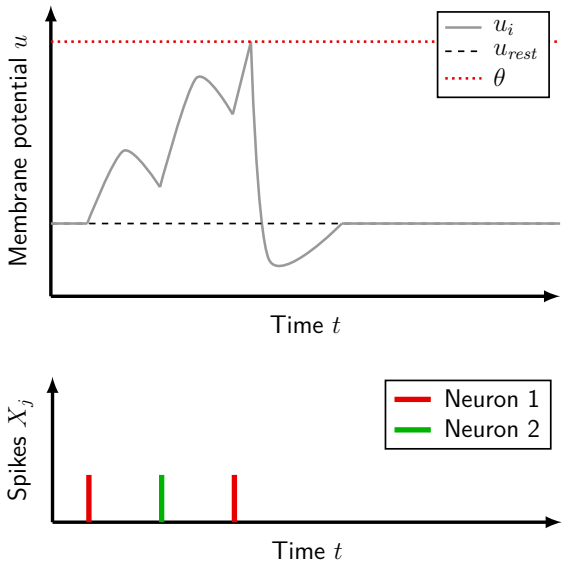


Figure 5.2: Build-up of postsynaptic membrane potential over time due to presynaptic spikes coming from two neurons. Adapted from Gerstner and Kistler (2002).

in nature and only taking in boolean values, the TLU could represent various logic gates such as AND or OR. And even though this is, obviously, not at all an accurate representation of a biological neuron, these neurons (in an adapted form) make up the ANNs that dominate much of machine learning today.

On the opposite end of the realism spectrum there is the spiking neuron model proposed by Hodgkin and Huxley (1952), who, after experimenting with axons of a squid, identified a set of differential equations that described the dynamics of a biological neuron. Furthermore, they put forward the idea that these differential equations could be represented as electrical circuits of a certain configuration. Notorious for its computational complexity, however, this model is not very usable in simulations of large SNNs, and we will not discuss it any further. Nevertheless, it served as inspiration for more tractable neuron models, of which the most prominent ones will be covered here, mostly following the descriptions by (Gerstner & Kistler, 2002; Gerstner et al., 2014).

LIF

Leaky integrate-and-fire (LIF) models, of which the first variants were proposed by Stein (1965, 1967), simplify the Hodgkin-Huxley model by assuming that action potentials always have the same shape, which implies that spike trains only carry information in the intervals between spikes, and can thus be reduced to sequences of events. Integrating these events over time then gives the membrane potential $u(t)$. This process can actually be modelled by a basic circuit consisting of a parallel capacitor C and resistor R driven by an input current $I(t)$. Defining $\tau_m = RC$ as the time constant of the membrane potential, representing the leaky integration that causes the potential to slowly decay, we end up with a first-order linear differential equation that looks like a *low-pass filter*:

$$\dot{u}(t) = -\frac{1}{\tau_m}[u(t) - u_{rest}] + \frac{R}{\tau_m} I(t) \tag{5.1}$$

where $I(t)$, the input current, represents incoming presynaptic spikes. In case of multiple presynaptic neurons, however, it would be more convenient to write this current as a sum over presynaptic spike trains. The membrane potential of postsynaptic neuron i would then be expressed as follows:

$$\dot{u}_i(t) = -\frac{1}{\tau_m}[u_i(t) - u_{rest}] + \frac{R}{\tau_m} \sum_j w_{ij} \sum_f \varepsilon\big(t - t_j^{(f)}\big) \tag{5.2}$$

where w_{ij} is the synaptic efficacy (strength of the connection, excitatory or inhibitory) between presynaptic neuron j and postsynaptic neuron i, $t_j^{(f)} < t$ are the presynaptic firing times with $f = 1, 2, \ldots$ and

$\varepsilon\big(t - t_j^{(f)}\big)$ the shape of these presynaptic spikes. For LIF models, this shape is usually neglected, i.e., $\varepsilon(s) = \delta(s)$, with δ the Dirac delta function, $\delta(s) = 0$ for $t \neq 0$ and $\int_{-\infty}^{\infty} \delta(s)\, ds = 1$. More realistically, however, the presynaptic spikes should have an effect of finite duration on the postsynaptic potential (PSP), i.e., they should decay over time. This can be achieved with, e.g., $\varepsilon(s) = \frac{1}{\tau_s} \exp(-s/\tau_s)$.

The membrane potential of a neuron not receiving any presynaptic spikes ($I(t) = 0$) decays to u_{rest}, which can be seen by solving Equation (5.1) with initial condition $u_i(t_0) = u_{rest} + \Delta u_i$ (meaning any potential above the resting state):

$$u_i(t) = u_{rest} + \Delta u_i \exp\left(-\frac{t - t_0}{\tau_m}\right) \quad \text{for } t > t_0 \tag{5.3}$$

However, if presynaptic spikes cause the potential threshold θ to be reached, i.e., $u_i\big(t_i^{(f)}\big) = \theta$, a postsynaptic spike is emitted at time $t_i^{(f)}$. Immediately after, the membrane potential is reset to u_{rest}, and the neuron may enter into a refractory period, interrupting the dynamics for a few milliseconds. Because the actual shape of the action potentials is neglected in the LIF model, a postsynaptic spike train $Y_i(t)$ of neuron i, firing at $t_i^{(f)} < t$ with $f = 1, 2, \ldots$, can be denoted as:

$$Y_i(t) = \sum_f \delta\big(t - t_i^{(f)}\big) \tag{5.4}$$

Due to its simplicity, the LIF model is frequently used in applications that involve large amounts of neurons (e.g., Bichler, Querlioz, Thorpe, Bourgoin & Gamrat, 2012; Diehl & Cook, 2015; Haessig, Berthelon, Ieng & Benosman, 2019).

SRM

Instead of describing the membrane potential with differential equations, the *spike response model* (SRM) by Kistler, Gerstner and Hemmen (1997) makes use of an integration over past presynaptic spikes. Suppose that postsynaptic neuron i last fired at time $\hat{t}_i$. Then, its membrane potential $u_i(t)$ evolves as follows, according to the SRM:

$$u_i(t) = u_{rest} + \eta\big(t - \hat{t}_i\big) + \sum_j w_{ij} \sum_f \varepsilon\big(t - \hat{t}_i,\, t - t_j^{(f)}\big) + \int_0^{\infty} \kappa\big(t - \hat{t}_i,\, s\big)\, I^{ext}(t - s)\, ds \tag{5.5}$$

where the behaviour of the neuron is mainly determined by three (usually exponential) kernels: $\eta(s)$, $\varepsilon(s)$ and $\kappa(s)$. $\eta(s)$ describes the time course of the membrane potential once it reaches the threshold θ (sometimes also called the *afterpotential*). Note that only the time of the last postsynaptic spike is important here. Other models, such as the cumulative SRM, make use of all past postsynaptic spikes (Gerstner, van Hemmen & Cowan, 1996), such that $\sum_f \eta\big(t - t_i^{(f)}\big)$ would represent the second term of Equation (5.5). $\varepsilon(s)$ corresponds to the response of the membrane potential to an incoming presynaptic spike. $\kappa(s)$ describes the variation in membrane potential due to an external input current I^{ext}. If there is no such current (which will mostly be the case in this thesis), this term vanishes.

An SRM neuron is said to fire when it crosses the threshold θ from below, since the afterpotential $\eta(s)$ causes it to cross the threshold value twice. So, firing occurs when $u_i(t) = \theta$ and $\dot{u}_i(t) > 0$.

A simpler version of the SRM, called SRM_0, removes the dependence of $\varepsilon(s)$ and $\kappa(s)$ upon the argument $t - \hat{t}_i$:

$$u_i(t) = u_{rest} + \eta\big(t - \hat{t}_i\big) + \sum_j w_{ij} \sum_f \varepsilon\big(t - t_j^{(f)}\big) + \int_0^{\infty} \kappa(s)\, I^{ext}(t - s)\, ds \tag{5.6}$$

Thus, each presynaptic spike evokes an identical PSP, independent of the presynaptic neuron or last postsynaptic firing time.

The generality of the SRM (and the SRM_0) allows it to represent the LIF model as well, given the right kernels. See Gerstner and Kistler (2002) for more on this.

Instead of having a sharp, deterministic threshold, as was the case with the LIF model, the firing threshold can be made stochastic. Also known as *escape noise*, this represents synaptic noise generated by presynaptic neurons in proximity of the neuron in consideration. An often-used solution is to let spikes be generated by an inhomogeneous Poisson point process with stochastic intensity (also called *instantaneous firing rate*):

$$\rho\left(u_i\left(t\right)\right) = \rho_0 \exp\left(\frac{u_i\left(t\right) - \theta}{\Delta\theta}\right) \tag{5.7}$$

with ρ_0 the stochastic intensity at the threshold, θ the formal threshold and $\Delta\theta$ the width of the threshold region. Integrating this expression, i.e., $\int_t^{t+\Delta t} \rho\left(u_i\left(t'\right)\right) dt' \approx \rho\left(u_i\left(t\right)\right) \Delta t$, will give the probability of firing.[2] However, this will lead to numerical instability for large Δt. To deal with this, we calculate the probability that the neuron does not fire, leading to an expression that remains bounded between zero and one:

$$\Pr\left\{\text{ spike in } \left[t, t+\Delta t\right] \mid u_i\left(t\right)\right\} = 1 - \exp\left(-\int_t^{t+\Delta t} \rho\left(u_i\left(t'\right)\right) dt'\right) \approx 1 - \exp\left(-\rho\left(u_i\left(t\right)\right) \Delta t\right) \tag{5.8}$$

Several learning rules discussed in the next section require this stochastic firing threshold to compute the probability density of a spike train (which would otherwise be non-differentiable) required for their derivation (e.g., Bohte et al., 2002; Florian, 2007; Frémaux et al., 2013; Pfister, Toyoizumi, Barber & Gerstner, 2006; Vasilaki et al., 2009).

5.2 Learning in spiking neural networks

Analogous to ANNs, learning in SNNs happens through modification of the connections between different neurons, in such a way as to improve some notion of performance or accuracy. Various approaches to learning exist for this, namely unsupervised, supervised and reinforcement learning, which we will all treat here. Unfortunately, the discrete spiking nature of SNNs severely limits the application of the highly-successful gradient-based optimisation algorithms that dominate learning in ANNs, such as backpropagation (Goodfellow et al., 2016). Instead, most learning in SNNs happens in an unsupervised way. Although this approach works (to a certain extent) for recognising patterns of spikes, it has no mechanism to specify desired behaviour or goals (Frémaux & Gerstner, 2016). Reward-modulated learning rules inspired by RL aim to solve this problem. Before going into more detail on the different types of learning in SNNs, however, we will first discuss the basics of learning in biological neurons.

5.2.1 Synaptic plasticity

Defined as the ability to modify synaptic efficacies, *synaptic plasticity* is one of the primary mechanisms underlying learning and memory in biological nervous systems (Martin, Grimwood & Morris, 2000; Sutton & Barto, 2018). Changing the efficacies, represented by the weights w_{ij} in the SRM and LIF model (see Equations (5.1) and (5.5), respectively), can be done through various mechanisms, portrayed by the different learning rules in the remainder of this section. For instance, one of the most prominent learning mechanisms in the nervous system, STDP (covered in Section 5.2.2), employs the relative timing of presynaptic and postsynaptic spikes for synaptic plasticity.

Another way would be through the presence of a *neuromodulator*, which is a neurotransmitter that has many other functions besides the direct, fast excitation or inhibition of targeted neurons. *Dopamine*, for example, is a neuromodulator associated with reward, and its presence can influence synaptic

[2] As Gerstner and Kistler (2002) explain in Section 5.3.1, $\rho\left(u_i\left(t\right)\right)$ can take on large values for $u_i\left(t\right) \gg \theta$, making it, strictly speaking, unsuitable for representing probability. However, because $u_i\left(t\right)$ is reset for each spike, and assuming the integration time is taken sufficiently small, such an overshoot is prevented.

plasticity (Sutton & Barto, 2018). The reward-modulated learning rules in Section 5.2.4 represent this phenomenon.

5.2.2 Unsupervised learning

In the context of SNNs, unsupervised learning is often referred to as *Hebbian* learning, due to the fact that the employed mechanisms stem from Hebb's postulate (Hebb, 1949):

> *'When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.'*

The postulate is often summarised as 'cells that fire together, wire together', although this neglects the causality implied by Hebb. More specifically, the synaptic efficacy of a connection is only increased in case cell A 'takes part in firing' cell B, i.e., if cell A fires just before cell B. It may be this temporal specificity of synaptic plasticity that is especially relevant to learning and memory (Bi & Poo, 2001). Several techniques implementing Hebbian learning have been developed over the years, and two will be mentioned here.

STDP

STDP is by far the most popular learning rule for SNNs, and it has a very intuitive interpretation. If a presynaptic neuron fires briefly before a postsynaptic neuron does so ($\Delta t < 0$), the connection between them is strengthened, in accordance with Hebb's postulate. However, if a presynaptic neuron fires shortly after a postsynaptic neuron ($\Delta t > 0$), the causal relationship between these events is spurious, and the efficacy of their connection is decreased. In case spikes are distant in time, the synaptic efficacy remains unchanged. The strengthening of a synapse is called *long-term potentiation* (LTP), whereas weakening is called *long-term depression* (LTD) (Bi & Poo, 1998).

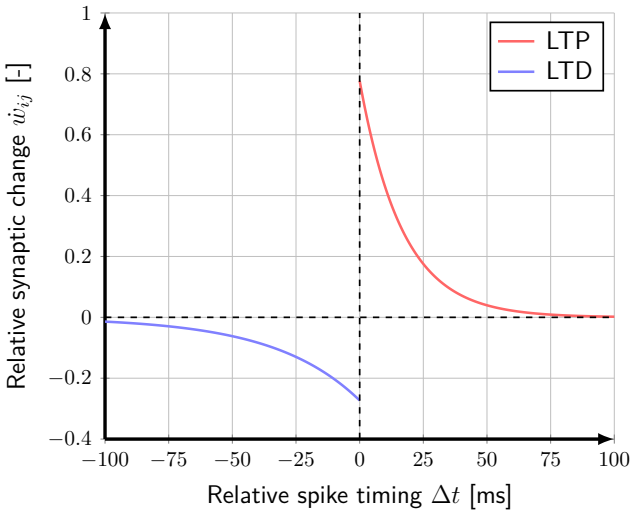


Figure 5.3: STDP windows based on experimental data. Adapted from Bi and Poo (2001). Note that they invert the definition of Δt, defining positive as pre-before-post firing.

In this form, STDP was observed in various types of neurons by Bi and Poo (1998), Markram, Lübke, Frotscher and Sakmann (1997), after which a formal model was developed by Song, Miller and Abbott (2000). The most common temporal windows, used to determine the amount of LTP/LTD, originate from the experimental data by Bi and Poo (1998, 2001) and the model by Song et al. (2000), and are shown in Figure 5.3. With $\Delta t = t_j^{(f)} - t_i^{(f)}$ the relative firing of presynaptic neuron j and postsynaptic neuron i, the synaptic plasticity induced by STDP can be mathematically formulated as (Song et al., 2000):

$$\dot{w}_{ij} = \begin{cases} A_+ \exp\left(\frac{\Delta t}{\tau_+}\right) & \text{if } \Delta t < 0 \\ A_- \exp\left(\frac{-\Delta t}{\tau_-}\right) & \text{if } \Delta t \geq 0 \end{cases} \tag{5.9}$$

with constants $A_+ > 0$, $A_- < 0$ determining the magnitudes of changes in synaptic efficacy, and time constants τ_+, τ_- setting the decay of the windows. However, this formulation allows synaptic efficacy to increase indefinitely under continued LTP, which is biologically not very accurate (and not very stable). Indeed, Rossum, Bi and Turrigiano (2000) observed that stronger synapses undergo relatively less potentiation than weak ones, whereas depression is more or less independent of synaptic strength. Gerstner and Kistler (2002) formulated the following adaptation to Equation (5.9) for this:

$$A_+ = a_+ \left(w^{max} - w_{ij} \right) \qquad A_- = a_- w_{ij} \tag{5.10}$$

with constants $a_+ > 0$, $a_- < 0$. STDP-variants that incorporate the current synaptic efficacy in updates are referred to as *multiplicative* rules, whereas Equation (5.9) would fall under *additive* STDP. Also note that, so far, we have only considered excitatory synapses in discussing STDP. In case of inhibitory synapses, whose weights w_{ij} are negative, the effects of STDP are inversed.

Despite the added stability offered by these multiplicative STDP rules, they nevertheless lead to *bimodal* weight distributions (see, e.g., Diehl & Cook, 2015; Kheradpisheh, Ganjtabesh, Thorpe & Masquelier, 2018), with weights either being equal to the lower or upper limit (w^{max} in Equation (5.10)). To enforce a more natural, i.e., balanced and unimodal, distribution of weights, Paredes-Vallés et al. (2019) proposed an inherently stable multiplicative STDP implementation based on the weight-dependent exponential rule by Shrestha, Ahmed, Wang and Qiu (2017) and the so-called *presynaptic trace* of which each synapse keeps track, containing the recent history of transmitted spikes (Morrison, Aertsen & Diesmann, 2007).

BCM

Predating STDP, the *Bienenstock-Cooper-Munro* (BCM) rule was modelled by Bienenstock, Cooper and Munro (1982) after their experimental results and those of others. Whereas STDP imposes LTP/LTD based on the relative timing of spikes, the BCM rule characterises synaptic changes as a product of presynaptic activity ν_j and a non-linear function $\phi(\nu_j, \nu_\theta)$ of postsynaptic activity $\nu_i = \sum_j w_{ij}\nu_j$ and a threshold $\nu_\theta = \mathbb{E}[\nu_i]$. By allowing this threshold to vary, different LTP/LTD windows can be achieved, with a typical one depicted in Figure 5.4. The BCM rule can be represented mathematically as (Bienenstock et al., 1982; Gerstner & Kistler, 2002):

$$\dot{w}_{ij} = -\frac{w_{ij}}{\tau_w} + \phi\left(\nu_i - \nu_\theta\right)\nu_j \tag{5.11}$$

with $-\frac{w_{ij}}{\tau_w}$ representing a uniform weight decay. Though different at first sight, the BCM rule and STDP actually appear to be closely related: Izhikevich and Desai (2003) showed that the BCM rule follows directly from STDP when the pre- and postsynaptic firing patterns comply with certain requirements, something which was confirmed by Baras and Meir (2007).

5.2.3 Supervised learning

Supervised learning makes use of labels of some kind, thus allowing the specification of desired behaviour. Almost all ANNs make use of a supervised training method called *backpropagation*, which uses chained derivatives to propagate each error backwards through the network, updating its weights in such a way as to not make that error again (Goodfellow et al., 2016). Unfortunately, the discrete spike trains that make up the signals within SNNs are not differentiable, meaning supervised methods will either have to come up with a completely different way to steer the network in the right direction, or find a substitute for the use of derivatives. Apart from this, considerable scepticism concerning the biological plausibility of backpropagation exists (Tavanaei et al., 2019).

Before going over two backpropagation methods for SNNs, we have to take note of another approach to achieving supervised learning in SNNs, namely by converting ANNs to SNNs, as is done by, e.g., Diehl

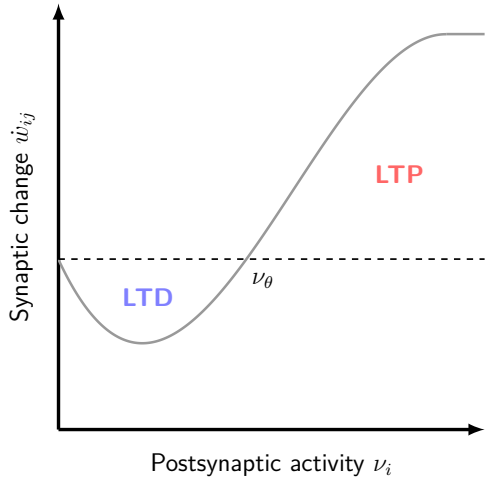


Figure 5.4: BCM windows for LTD and LTP. Adapted from Gerstner and Kistler (2002).

et al. (2015). However, as this method is even less biologically plausible than just using ANNs, it will not be considered further.

Bohte et al. (2002) appear to have been the first to introduce a method that allows training SNNs by backpropagating errors. The method, *SpikeProp*, considers spike timing in its cost function and was able to solve a temporally encoded XOR problem as well as some other simple classification problems. By using SRM neurons (discussed in Section 5.1.2) and linearising the relation between spike times and PSPs, the problem of non-differentiable spike trains was circumvented. However, SpikeProp only works with neurons that each fire a single spike to encode information temporally. This poses limitations on the temporal encodings that can be used, and excludes mechanisms such as rate-based coding.

Spike layer error reassignment (SLAYER), a more recent backpropagation method developed by Shrestha and Orchard (2018), does allow for information encoding using both precise spike times as well as firing rates by letting its cost function represent either of those. The method produced state-of-the-art results on the N-MNIST dataset, the event-based version of the famous MNIST dataset, developed by Orchard, Jayawant, Cohen and Thakor (2015). Furthermore, SLAYER was able to correctly classify human gestures recorded by a DVS, outperforming a previous implementation using IBM's TrueNorth neuromorphic chip (Amir et al., 2017), while using significantly less neurons.

Apart from these, many more supervised learning methods for SNNs exist. Refer to Tavanaei et al. (2019) for a comprehensive overview.

5.2.4 Reinforcement learning

Classical Hebbian learning protocols, such as STDP, implement synaptic plasticity (and thus learning) based on joint pre- and postsynaptic activity. And while heavily supported by experimental data (e.g., Bi & Poo, 1998; Markram et al., 1997), these methods neglect the influence of neuromodulators, such as dopamine, on plasticity, something for which also extensive experimental evidence exists (see Frémaux & Gerstner, 2016; Gerstner, Lehmann, Liakoni, Corneil & Brea, 2018, for an overview).

As was first discussed in Section 4.1.2, the role of dopamine involves signalling some notion of reward, most likely in the form of a reinforcement signal that directs learning (Sutton & Barto, 2018). Gerstner et al. (2018) argue that neuromodulators encoding reward or novelty, in combination with eligibility traces, are necessary to bridge the gap in time scale between elementary behaviours (seconds) and neuronal action potentials (milliseconds). And although the effect of neuromodulators could also be explained in terms of supervised learning (encoding a high-dimensional, neuron-specific error signal) instead of RL (encoding a scalar, global error signal), the latter seems more intuitive and simple, and

has been implemented much more often (e.g., Florian, 2007; Frémaux, Sprekeler & Gerstner, 2010, 2013; Vasilaki et al., 2009).

Frémaux and Gerstner (2016) made a distinction between different types of these *reward-modulated* learning rules (sometimes called *neo-Hebbian* or *three-factor* rules, with the first two factors being pre- and postsynaptic activity, and the third a neuromodulator). We will now go over each category in turn. Figure 5.5 already shows a schematic overview of the differences between the shapes of the learning rules in those categories.

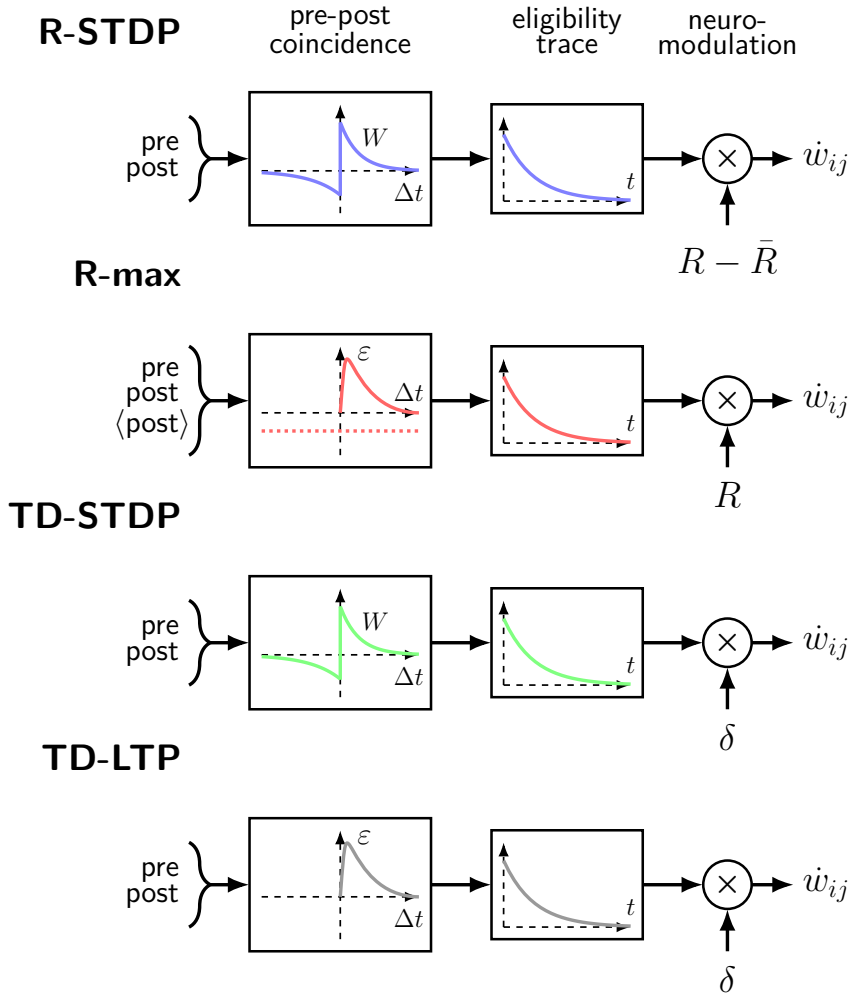


Figure 5.5: Overview of reward-modulated learning rules. Each rule can be broken down into three steps: correlating pre- and postsynaptic activity, updating the eligibility trace and performing neuromodulation. R and $\bar{R}$ represent reward and mean reward (e.g., based on a certain number of past episodes or steps), respectively, and δ indicates the TD error (as covered in Section 4.2.4). Adapted from Frémaux and Gerstner (2016).

R-STDP

Probably the most straightforward way of introducing a neuromodulatory effect would be to extend STDP with a third factor, as is done for *R-STDP*. With STDP being a *phenomenological* model (based on experimental data), there are no theoretical guarantees on whether adding a third, reward-related factor will actually lead to maximisation of accumulated reward, something which is the case for classical RL methods (Sutton & Barto, 2018). However, Legenstein, Pecevski and Maass (2008) develop analytical tools which can be used to analyse for which problems R-STDP does display reward-maximising behaviour, while Frémaux et al. (2010) derive theoretical conditions for R-STDP to demonstrate successful learning. Together with the fact that, for many typical RL problems, significant learning is

demonstrated (e.g. Farries & Fairhall, 2007; Florian, 2007; Frémaux et al., 2010; Legenstein et al., 2008; Vasilaki et al., 2009), this makes it seem worthwhile to cover R-STDP in detail here.

Combining the notations by Florian (2007), Frémaux et al. (2010), R-STDP can be represented mathematically as:

$$\dot{w}_{ij} = \alpha S\left(R\left(t\right)\right) e_{ij}\left(t\right) \tag{5.12}$$

$$\dot{e}_{ij}\left(t\right) = -\frac{e_{ij}\left(t\right)}{\tau_e} + U_{ij}\left(t\right) \tag{5.13}$$

$$U_{ij}\left(t\right) = Y_i\left(t\right) A_+ \sum_f \exp\left(-\frac{t - t_j^{(f)}}{\tau_+}\right) + X_j\left(t\right) A_- \sum_f \exp\left(-\frac{t - t_i^{(f)}}{\tau_-}\right) \tag{5.14}$$

with α the learning rate, $S\left(R\left(t\right)\right)$ the *success signal*, which is a monotonic function of reward $R\left(t\right)$, $e_{ij}\left(t\right)$ the eligibility trace as low-pass filter, $U_{ij}\left(t\right)$ the term representing the unsupervised part of the learning rule, the various τs representing time constants and $X_j\left(t\right)$ and $Y_i\left(t\right)$ the pre- and postsynaptic spike trains with spikes at $t_j^{(f)}$ and $t_i^{(f)}$ up to t, respectively. Just as with regular STDP, $A_\pm$ can be taken as constant to achieve additive STDP, or be made dependent upon the weight to get multiplicative STDP (see Equation (5.10)). Note that the sum in the terms of $U_{ij}\left(t\right)$ is nothing more than a convolution, i.e., the integral (running from $t = 0$ to $t = \infty$) of the product between the pre- or postsynaptic spike train (sum of Dirac pulses) and the corresponding LTP/LTD window. Furthermore, the learning rate α can also be put in front of the unsupervised term $U_{ij}\left(t\right)$, however we put it in the weight update for generality purposes. The same goes for the eligibility trace time constant τ_e, which we put in the first term on the right-hand side of Equation (5.13) (following the convention by Frémaux and Gerstner (2016), Vasilaki et al. (2009)) instead of on the left-hand side (as Florian (2007), Frémaux et al. (2010) do), making the spikes contribute more to the weight updates.

An interesting note by Frémaux and Gerstner (2016), Frémaux et al. (2010) is the fact that the *post-before-pre* (LTD) part of the STDP window, represented by the second term of Equation (5.14), does not contribute significantly to reward-maximising behaviour, and can even hamper performance. By setting $A_- = 0$, the LTD portion of the window can be removed.

Averaging the weight updates implemented by Equation (5.12) over multiple episodes and splitting it into two parts, we end up with:

$$\langle\, \Delta w_{ij} \,\rangle = \langle\, S\left(R\left(t\right)\right) e_{ij}\left(t\right) \,\rangle = \mathrm{Cov}\left[\, S\left(R\left(t\right)\right),\, e_{ij}\left(t\right) \,\right] + \langle\, S\left(R\left(t\right)\right) \,\rangle \langle\, e_{ij}\left(t\right) \,\rangle \tag{5.15}$$

where the first term reflects the reward-sensitive component of learning that can potentially detect rewarding behaviours, and the second term the mean behaviour of the unsupervised part of the learning, $U_{ij}\left(t\right)$, introducing an unsupervised bias in the weight updates (Frémaux et al., 2010). To cancel this bias, the average success signal $\langle S\left(R\left(t\right)\right)\rangle$ has to be made zero, which can be done by writing success as a sort of RPE:

$$S\left(R\left(t\right)\right) = R\left(t\right) - \bar{R} \tag{5.16}$$

where $\bar{R}$ is, for example, an exponentially weighted moving average over past rewards, updated every so many learning episodes. So, analogous to the role of dopamine as discussed in Section 4.1.2, the success signal is positive if the received reward is above the mean/expected reward, and negative when below. Note that this expected reward has to be task-specific, as both Frémaux and Gerstner (2016), Frémaux et al. (2010) point out. Because the other term in Equation (5.15) correlates candidate weight changes with the success signal through the covariance, R-STDP is a so-called *covariance-based* learning rule.

Izhikevich (2007) instead uses sparse, positive rewards to 'gate' STDP, strengthening a specific connection and depressing all others. However, given the limited use of sparse rewards in the robotic problem considered in this thesis, we will not look into this method further.

R-max

Another covariance-based learning rule is *R-max*, with 'max' indicating this rule was not based on experimental data, but was derived directly from reward maximisation principles (see, e.g., Florian, 2007; Pfister et al., 2006; Vasilaki et al., 2009, for a derivation). This derivation relies on using SRM neurons with stochastic thresholds (escape noise) to deal with the non-differentiable spike trains, which causes the unsupervised term of the learning rule to look as follows (with equations for weight update and eligibility trace dynamics the same as Equations (5.12) and (5.13), respectively):

$$U_{ij}(t) = \left(Y_i(t) - \frac{\rho_i(t)}{1 + \tau_c \rho_i(t)} \right) \sum_f \varepsilon\big(t - t_j^{(f)}\big) \tag{5.17}$$

where $\rho_i(t)$ the instantaneous firing rate (see Equation (5.7)) and $\varepsilon(s)$ the shape of the PSP. τ_c is a parameter introduced by Vasilaki et al. (2009), which allows a family of rules: for $\tau_c = 0$ it becomes a *strict policy gradient* method (purely reward maximisation), while a *naive Hebbian* model can be achieved with $\tau_c \to \infty$. The authors go on to show that pure policy gradient methods fail to learn for several problems and architectures, implying that some Hebbian learning is useful for learning in those problems. Frémaux et al. (2010) claim that this is the case because the coding scheme used (higher neuron activity $\to$ higher probability of choosing an action) is in agreement with the unsupervised bias.

Like the success signal $S(R(t))$ for R-STDP, τ_c can seen as a trade-off parameter that determines the balance between the reward-sensitive component of learning, and the unsupervised/Hebbian bias. With $\tau_c = 0$, the rule becomes unbiased (LTP and LTD in balance), and there is no need for a success signal encoding RPE. Nevertheless, R-max with $\tau_c = 0$ can benefit from using RPE, because this reduces the trial-to-trial variability of the weight changes (as it uses the average reward over multiple episodes), allowing the weights to stay closer to the optimum (Frémaux et al., 2010).

So far, the suggested way of implementing RPE as the success signal has been a moving average over several episodes. However, there are more sophisticated ways of doing this, as the next learning rule will show.

TD-LTP and TD-STDP

Building upon the continuous TD learning approach by Doya (2000) (discussed in Section 4.2.9), Frémaux et al. (2013) developed two learning rules that implement actor-critic learning for SNNs, and which show much more learning potential than any of the covariance-based rules. As explained in Section 4.2.7, these consist of two networks, actor and critic, where the critic learns to predict expected future reward, and the actor decides on which actions to take based on these predictions. The TD error from the critic acts as a reinforcement signal for both actor and critic, encoding RPE, and perfectly fitting in with the role of dopamine as mentioned in Section 4.1.2. With the critic now serving as a predictor of expected future reward, there is no need for a modified success signal as was the case for R-STDP.

Starting from the weight update and TD error for continuous time derived by Doya (2000), Frémaux et al. (2013) arrive at *TD-LTP*, whose name comes from the fact that its shape is similar to the LTP window of STDP (see Figure 5.5). Adapting the mathematical formulation to fit with the previously mentioned learning rules, TD-LTP's weight update (for both actor and critic) is defined as follows:

$$\dot{w}_{ij} = \alpha \delta(t)\, e_{ij}(t) \tag{5.18}$$

$$\dot{e}_{ij}(t) = -\frac{e_{ij}(t)}{\tau_e} + Y_i(t) \sum_f \varepsilon\big(t - t_j^{(f)}\big) \tag{5.19}$$

$$\delta(t) = \frac{V_{scale}}{N_{critic}} \sum_{i=1}^{N_{critic}} \sum_f \left(\dot{\kappa}\big(t - t_i^{(f)}\big) - \frac{\kappa\big(t - t_i^{(f)}\big)}{\tau_r} \right) - \frac{V_0}{\tau_r} + R(t) \tag{5.20}$$

where $\delta(t)$ is the critic's TD error, N_{critic} the number of critic neurons, $\kappa(s)$ a causal smoothing kernel to determine the firing rate of neurons and τ_r a time constant. V_{scale} and V_0 specify the value scaling factor and the value for no spiking activity, respectively, and seem to be very dependent on the problem the learning rule is used for. Frémaux et al. (2013) do not explicitly write out the differential equation for the eligibility trace $e_{ij}(t)$, and instead perform a convolution between the second term of Equation (5.19) and the kernel $\frac{\kappa(s)}{\tau_r}$, which has the shape $\kappa(s) = \frac{\exp(-s/\tau_k)-\exp(-s/\tau_q)}{\tau_k-\tau_q}$. Furthermore, they limit the sum over presynaptic spikes in the second term of Equation (5.19) to those spikes arriving after the last postsynaptic spike. Even though this conflicts with the use of presynaptic spike trains in similar papers (e.g., Florian, 2007; Frémaux et al., 2010; Vasilaki et al., 2009), the impact of this is most likely limited given the exponentially decaying nature of $\varepsilon(s)$.

Instead of deriving a TD-based learning rule from reward-maximisation principles, as is the case with TD-LTP, the TD error $\delta(t)$ can also be used as critic to cancel the unsupervised bias in R-STDP, taking over this role from the modified success signal $S(R(t)) = R(t) - \bar{R}$. The resulting learning rule, called *TD-STDP*, is identical to the R-STDP rule defined by Equations (5.12) to (5.14), except for $S(R(t))$ being replaced by $\delta(t)$ from Equation (5.20). As with R-STDP, the LTD portion of the STDP window does not contribute significantly to reward-maximising behaviour (Frémaux & Gerstner, 2016; Frémaux et al., 2013). While experimental evidence has been found for the bi-phasic learning window of regular, unsupervised STDP (Bi & Poo, 1998; Markram et al., 1997), this is not the case for the reward-modulated version of STDP.

Other variants of TD-based learning rules were developed by di Castro, Volkinshtein and Meir (2009), Potjans, Morrison and Diesmann (2009), and while the latter looks similar to the definition by Frémaux et al. (2013), the former did not seem make use of eligibility traces, which allow for faster learning (Frémaux & Gerstner, 2016).

Comparing rules

The overview papers by Frémaux and Gerstner (2016), Frémaux et al. (2010) allow to make a comparison between the different types of reward-modulated learning rules on different aspects, namely:

- Is a reward-predicting critic or RPE crucial for learning reward-maximising behaviour, and how do the implementations vary per rule?
- Are there differences in convergence speed between the various rules?
- Does the rule require a compact representation of state/action spaces?

An overview of the answers to these questions is given in Table 5.1. From this comparison, it seems that TD-based learning rules show the most potential for learning, even though they might require a complexity reduction (finding a low-dimensional manifold in the state/action space) for learning to be tractable.

Table 5.1: Comparison of reward-modulated learning rules for SNNs. Based on Frémaux and Gerstner (2016), Frémaux, Sprekeler and Gerstner (2010).

	Covariance-based		**TD-based**
	R-STDP	**R-max**	**TD-STDP + TD-LTP**
Critic / RPE	Task-specific moving average	Improves performance	Actor-critic architecture
Convergence	Slow: many trials	Slow: many trials	Fast: few trials
Representation	Can be high-dimensional	Can be high-dimensional	Requires compactness

5.3 Neuromorphic applications

Most advantages of SNNs, such as their increased computational power and power efficiency, stem from hardware implementations using neuromorphic chips, as conventional computer architectures currently require large amounts of computational expense to simulate them (Bouvier et al., 2019). Nevertheless, simulation of SNNs on conventional hardware is crucial in the fast development of improved learning techniques and the search for potential applications, as neuromorphic chips are not yet mainstream. This section will therefore explore several of the most prominent neuromorphic hardware implementations, as well as some commonly used simulation frameworks. Refer to Bouvier et al. (2019), Pfeiffer and Pfeil (2018) for a more complete overview of the various hardware implementations. Finally, applications (software or hardware) of SNNs in problems relevant to this thesis will be covered.

5.3.1 Hardware implementations

Bouvier et al. (2019) perform an excellent up-to-date comparison of neuromorphic chips capable of emulating large-scale SNNs, with SpiNNaker (Furber, Galluppi, Temple & Plana, 2014), TrueNorth (Merolla et al., 2014) and Loihi (Davies et al., 2018) being the most prominent ones. We will succinctly review their characteristics here.

SpiNNaker makes use of a single chip consisting of a network of central processing units (CPUs) tightly connected to local memory. While this probably makes it the most configurable chip (allowing, e.g., on-chip learning and many different neuron models), it leaves the chip lacking in energy efficiency and processing speed compared to others. Implementations of STDP (Diehl & Cook, 2014) and R-STDP (Mikaitis, Pineda García, Knight & Furber, 2018) have already been demonstrated on SpiNNaker. IBM's TrueNorth, on the other hand, is aimed at ultra-low power consumption at the cost of flexibility, allowing no on-chip learning and only a single neuron model (LIF). The most recently developed chip, Intel's Loihi, aims to achieve a balance between power efficiency and flexibility, with a slightly higher power consumption than TrueNorth, but all the flexibility of the SpiNNaker platform. It seems the most promising chip for applications related to this thesis. Researchers at Intel, having developed a toolkit (with Python application programming interface (API)) for programming SNNs on Loihi, show that it allows for the implementation of STDP (Lin et al., 2018). So far, however, no implementations involving neuromodulated learning rules have been developed.

5.3.2 Simulation frameworks

Flexible, easy-to-use simulation frameworks are crucial to advancing the state-of-the-art in terms of learning in SNNs, but also in terms of potential applications. For quite some time, simulation packages with varying characteristics have been available, some more capable than others. Here, we will review several of them, with the goal of finding candidates for carrying out experiments as part of this thesis.

Some frameworks, such as NEST[3] (Eppler, Helias, Muller, Diesmann & Gewaltig, 2009; Gewaltig & Diesmann, 2007), Brian[4] (Goodman & Brette, 2008; Stimberg, Goodman, Benichoux & Brette, 2013) and ANNarchy[5] (Vitay, Dinkelbach & Hamker, 2015), focus on biologically realistic simulations of SNNs, and allow the user to specify the details of different parts of each neuron. A major benefit of these packages is that, besides the built-in modules for neurons and connections, the dynamics of these parts can be specified using ordinary differential equations (ODEs) that can be taken straight from neuroscientific literature. However, the fact that these frameworks are written in a combination of low- and high-level programming languages (e.g., C with an interface in Python) limits flexibility in terms of modifying existing objects or adding new ones, such as reward-modulated learning rules. Apart from this, networks are required to be homogeneous, meaning that only one type of neuron/synapse can be used. These limitations make the mentioned frameworks unlikely candidates for this thesis.

[3] https://www.nest-simulator.org/
[4] https://github.com/brian-team/brian2
[5] https://github.com/ANNarchy/ANNarchy

More recently developed packages, like SpykeTorch[6] (Mozafari, Ganjtabesh, Nowzari-Dalini & Masquelier, 2019) or BindsNET[7] (Hazan et al., 2018), aim to be flexible at the cost of biological accuracy. For example, BindsNET is built on the popular deep-learning framework PyTorch[8] (which can harness the compute power offered by GPUs) and includes many different types of neurons, connections and learning rules, including reward-modulated rules such as R-STDP. Even though BindsNET does not explicitly support ODEs, we can convert the ODEs representing the dynamics of neurons and learning rules into difference equations ourselves and let BindsNET solve for them at regular time steps (which is of course what the above-mentioned packages that can deal with ODEs do under the hood). Furthermore, the way BindsNET is structured allows users to easily add their own objects to the framework, as well as existing RL environments through its interface with OpenAI Gym[9]. SpykeTorch, on the other hand, seems more limited in the number of neuron models and learning rules available, despite it also being built on top of PyTorch. Apart from this, SpykeTorch can only simulate single-spike SNNs (at most one spike per neuron), which could potentially limit its applications.

Whereas all the above-mentioned frameworks have an interface in the popular programming language Python[10], cuSNN, developed by Paredes-Vallés et al. (2019) is written completely in C++ to enable GPU acceleration with CUDA. While the package includes various neuron models and synapses, the learning rules are limited to variants of unsupervised STDP. With C++ being much less flexible than Python, resulting in slower prototyping, and with BindsNET also offering GPU acceleration, this package does not seem particularly suited for this thesis.

5.3.3 Applications in optical flow estimation

Instead of applying the event-based methods covered in Section 3.2.1 to estimate optical flow, SNNs can be used to perform this task. Several examples will be given here.

Most work using SNNs for optical flow estimation tries to mimic some of the filter-based estimation methods mentioned in Sections 2.1.3 and 3.2.1 by configuring the SNN architecture in smart ways. For example, Orchard, Benosman, Etienne-Cummings and Thakor (2013) implements spatio-temporally-oriented filters consisting of multiple non-plastic synapses with different delays to capture motion, however the large number of neurons needed prevents this method from running in real time. To deal with this, Brosch and Neumann (2016), Haessig, Cassidy, Alvarez, Benosman and Orchard (2018) implement these filters on IBM's TrueNorth chip (Merolla et al., 2014), but still no actual learning is going on.

Paredes-Vallés et al. (2019), on the other hand, present a convolutional SNN which, in combination with a novel STDP rule, allows for motion selectivity to emerge in an unsupervised manner from the events generated by an event-based camera. The hierarchical feature extraction that this SNN learns to perform, from capturing geometric features to identifying their local motion and integrating this into a global ego-motion estimate, results in strong correlations between the activity of neurons in the output layer and the magnitudes of visual optical flow observables (ventral flows and divergence, see Section 2.1.2). To be suitable for vision-based navigation of MAVs, a method for 1) identifying which neurons encode which visual observables, and 2) creating a mapping to control inputs would need to be developed, which is where an SNN trained with reward-modulated learning rules would come in. Implementation on a neuromorphic chip would increase the potential of this approach even further.

5.3.4 Applications in vision-based navigation

Several authors have so far employed SNNs for vision-based navigation of MAVs and other vehicles. Clawson et al. (2016) train an SNN in flight using an R-STDP-variant (Foderaro, Henriquez & Ferrari, 2010) to perform lateral control of a simulated RoboBee (Ma et al., 2013). One of the sensory inputs to the SNN comes from a rudimentary visual sensor that can estimate pitch and roll rates by measur-

[6] https://github.com/miladmozafari/SpykeTorch
[7] https://github.com/Hananel-Hazan/bindsnet
[8] https://pytorch.org/
[9] https://gym.openai.com/
[10] https://www.python.org/

ing changes in intensity, making navigation partially dependent on vision. The authors demonstrate that R-STDP is capable of learning MAV control, however instead of letting reward be defined as a global metric (e.g., hover at a certain altitude), it follows from the control error with respect to a linear quadratic regulator (LQR), implying learning is more supervised than actual RL. Still, this application shows promise for other MAV controllers based on SNNs.

Zhao et al. (2018), on the other hand, approach the problem of performing vision-based control of MAVs from a more bio-inspired perspective, trying to mimic the actor-critic structure found in the human brain. Instead of performing flight control directly, the SNN decides on which action to take (up, down, left right) in order to avoid obstacles and fly through windows. Visual inputs from a conventional, frame-based camera are preprocessed to locate the obstacle in the field-of-view, after which the corresponding input neuron is stimulated. This means that no actual optical flow control is going on, making the learning problem significantly simpler for the SNN. Nevertheless, successful learning of real-world control is demonstrated with a variant of neuromodulated STDP.

Evidently, SNNs can also be applied to vision-based control of ground-based vehicles, as was shown by Bing, Meschede, Huang et al. (2018), Bing et al. (2019), Nichols, McDaid and Siddique (2013). While these implementations did make use of R-STDP and TD-based methods, respectively, to train their networks successfully in following lanes or walls, something related to actual optical flow control was learnt only by the network of Bing, Meschede, Huang et al. (2018), which takes aggregated events from a DVS to decide on the motion direction. The authors made reward dependent on the distance from the lane's centre, which makes for a true RL problem, but did not implement RPE yet, leaving room for improvement. Furthermore, these SNN controllers were not tested in the real world.

6

Synthesis of Literature

This chapter aims to provide a synthesis of the literature study conducted in this thesis. The goal of this study has been to collect relevant literature on reward-modulated neuromorphic computing for vision-based navigation of MAVs, and three areas were covered in particular. First, bio-inspired control of MAVs using visual observables was explored. Valuable insights regarding the advantages and feasibility of event-based optical flow approaches were obtained. Second, RL was introduced as a biologically plausible way of learning, with foundations in both psychology and neuroscience. Relevant implementations of RL in both simulated and real-world environments, along with some of the challenges encountered in robot control, were listed. Third, SNNs were presented as efficient computational frameworks derived from biological neurons, that allow to exploit the benefits of event-based data. Reward-modulated learning rules, inspired by RL and biological approaches to modifying neurons, were found to be able to train these networks in performing various navigational tasks. A plethora of implementations in both soft- and hardware demonstrated the activity and progression in this field of computing.

6.1 Vision-based navigation for MAVs

Optical flow (Gibson, 1950) was presented as a source of information about an observer's ego-motion and the 3D structure of the visual scene. Various methods for the estimation of this optical flow were discussed, with (adaptations of) the gradient-based method by Lucas and Kanade (1981) still the most-used due to its simplicity, especially for MAV-related purposes (e.g., de Croon, 2016; Ho et al., 2018). In an effort to summarise the optical flow field and, through this, further relaxing the demands on computational power, *visual observables* were introduced (de Croon et al., 2013). Ways in which flying insects make use of these observables, such as balancing *lateral flows* for horizontal control (e.g., Baird et al., 2005; Srinivasan et al., 1991) and *divergence* for landing (Baird et al., 2013), were successfully translated to bio-inspired navigation methods for MAVs (e.g., Expert & Ruffier, 2015; Herissé et al., 2012; Ruffier & Franceschini, 2015). *Self-induced oscillations* emerging from divergence-based vertical control were shown by, e.g., de Croon (2016), Ho and de Croon (2016), Ho et al. (2018) to be both a blessing and a curse, imposing limitations on control accuracy, but also allowing the inference of absolute height.

Inspired by the sparse, asynchronous nature of biological vision systems when quantifying motion, *event-based cameras* were introduced as a high-temporal-resolution and low-latency alternative to their frame-based counterparts (Gallego et al., 2019; Posch et al., 2014). Registering brightness changes as they occur, instead of at fixed intervals, allows these cameras to deal with variant visual dynamics in different parts of their field-of-view. The sparsity of these registered *events* ensures that energy consumption is greatly decreased (Lichtsteiner et al., 2008), whereas the inclusion of information on local brightness changes makes them ideal for optical flow estimation. Various event-based estimation methods were developed to exploit this feat (Rueckauer & Delbruck, 2016), and implementations of event-based cameras for vertical MAV control showed their ability to guide high-speed vision-based manoeuvres (Pijnacker Hordijk et al., 2018).

6.2 Reinforcement learning

RL was introduced as a computational framework for learning from interacting with the environment, i.e., an agent must discover rewarding actions through trial-and-error, with the goal of maximising cumulative *reward* (Sutton & Barto, 2018). Contrary to unsupervised learning, which is mainly useful for learning patterns in data, or supervised learning, where an agent is being told which actions to take, RL provides a biologically plausible way of learning desirable behaviours (Neftci & Averbeck, 2019), and its fundamentals were found to have many touching points with both psychology (e.g., Pavlov, 1927; Skinner, 1938, 1963; Thorndike, 1898) and neuroscience (e.g., Schultz & Romo, 1990; Schultz, 2002). For instance, Thorndike's *law of effect* (Thorndike, 1911) shows great overlap with the trial-and-error learning in RL, which is not a blind process of just trying out actions, but actually consists of finding rewarding actions and connecting them to certain states. The *credit assignment problem* described by Minsky (1961) was introduced as one of the core problems in RL, requiring learning systems to have methods of reinforcing past actions based on current rewards. *Eligibility traces* and *value functions* learnt through TD methods demonstrated the ability to cope with this difficulty, again both having their foundations in biology (e.g., Hull, 1943; Klopf, 1982). *Dopamine*, one of the chemicals responsible for reward processing in animal brains, was shown by Montague et al. (1996) to encode *RPE*, which acts as a reinforcement signal for directing changes in an agent's valuations and decision-making (Schultz, 1998; Schultz et al., 1997).

The *TD learning* mentioned previously was discussed in depth and found to be one of the most influential methods in RL, forming a unifying framework that combines the best of Monte Carlo and DP ideas, namely learning from pure experience and *bootstrapping*, respectively. Eligibility traces and *function approximation* were put forward as extensions that could enhance the learning potential of TD learning even further, while the TD error acting as the *reinforcement signal* in these methods was shown to closely correspond to the afore-mentioned RPE and dopamine (Sutton & Barto, 2018). Methods for directly learning *policies* (actions) instead of value functions (desirability of states), such as *actor-critic methods* (where an actor decides on actions based on valuations by a critic), were also covered, not in the least because these also have biological counterparts in animal brains (e.g., O'Doherty et al., 2004).

Multiple well-known successes of RL in game playing (e.g., Mnih et al., 2015; Silver et al., 2016) were discussed to illustrate the generality of the framework, but the hardships faced in applying RL to robot control were touched upon (Kober et al., 2013), since these are especially relevant to this thesis. For instance, real-world environments bring about uncertainties and noise, which could lead to difficulties in obtaining enough high-quality training experience (e.g., Sutton et al., 2007). Furthermore, the fact that the agent's goals can only be specified in terms of a scalar signal might seem appealing at first sight, but often turns out to be immensely difficult for poorly quantifiable behaviours. Luckily, deviations of an MAV from a desired flying path do not suffer from this, and smart reward functions can be created based on this deviation (e.g., Rodriguez-Ramos et al., 2018; Rodriguez-Ramos et al., 2019). Together with extensions of RL methods to continuous time and actions (Doya, 2000), this allowed the successful application to vision-based MAV landings (Rodriguez-Ramos et al., 2018).

6.3 Reward-modulated neuromorphic computing

Neuromorphic computing refers to the use of VLSI of analog electronics, which have the advantage of orders of magnitude more power-efficient than their digital counterparts, to mimic neurological architectures (Mead, 1990). *SNNs*, proposed by Maass (1997) as a way of implementing these architectures, were found to be computationally more powerful than conventional ANNs (Bouvier et al., 2019), while being more energy efficient (e.g., Pfeiffer & Pfeil, 2018; Tavanaei et al., 2019). These characteristics stem from the fact that, analogous to event-based cameras and biological neurons, these networks process data in a sparse and asynchronous manner using sets of *spikes*, making them a natural fit for working with event-based vision inputs (Orchard & Etienne-Cummings, 2014). The temporal aspect of these spikes furthermore allows SNNs to cope with rapidly changing inputs (Thorpe et al., 2001). Several neuromorphic chips demonstrating these advantages, such as Intel's Loihi (Davies et al., 2018), were covered, as well as some of the applications for which these have been used (e.g., Brosch & Neumann,

2016; Haessig et al., 2018). Software simulators for SNNs, of which BindsNET (Hazan et al., 2018) seemed the most promising, were also covered. While these do not benefit from any of the energy-related advantages mentioned above, they do allow fast prototyping and the comparison of many architectures.

Unfortunately, implementing SNNs also comes with a price: the discrete spiking nature of these networks imposes severe restrictions on the learning methods that can be used, and almost none of the successful gradient-based optimisation algorithms translate well from conventional ANNs to SNNs. Instead, most learning was found to be *unsupervised* and based on the relative timing of spikes between two neurons (STDP) (e.g., Markram et al., 1997; Song et al., 2000). As mentioned before, this works well for extracting patterns from data, and Paredes-Vallés et al. (2019) demonstrated an SNN capable of correlating event-based camera inputs and visual observables. Reasonably, this method could be extended to neuromorphic vision-based navigation of an MAV using another SNN to correlate these visual observables with control inputs.

Reward-modulated learning rules, taking inspiration from RL and neuroscience (Gerstner et al., 2018), allow for the learning of desired behaviours (Frémaux & Gerstner, 2016), and were deemed to be most suitable for this task. To this end, several neuron models and reward-modulated rules were introduced, each with their own strengths and weaknesses. For instance, *covariance-based* rules such as R-STDP (e.g., Florian, 2007) and R-max (e.g., Vasilaki et al., 2009) can deal with high-dimensional state spaces (Frémaux et al., 2010), but their learning potential stays behind *TD-based* methods (Frémaux et al., 2013) like TD-STDP and TD-LTP, which employ an actor-critic structure to optimally exploit experience. RPE (linked to dopamine) was shown to be of crucial importance to successful learning in some of these rules, while improving performance in others (Frémaux & Gerstner, 2016). Several applications of SNNs to robot control were discussed, however most were found to be lacking in some aspects. For instance, Clawson et al. (2016) successfully learnt vision-assisted flight control of a flapping MAV in flight using R-STDP, but this was performed in simulation and the learning signal originated from an LQR, making it more supervised learning than RL. Zhao et al. (2018) do perform real-world visual control of an MAV learnt using SNNs and a reward-modulated rule, but still rely on algorithmic methods for detecting obstacles and motion, instead of optical flow. Bing, Meschede, Huang et al. (2018), however, do make use of something resembling event-based optical flow, using aggregated events to decide on the motion direction, and to subsequently learn to follow a lane using R-STDP. With reward dependent on the deviation from the lane's centre, this makes for a true RL problem, and a great example of a benchmark for the various reward-modulated learning rules. Part III will carry out such a benchmark, with the goal of analysing the potential of the various rules for a simple MAV-related control problem.

III

Preliminary Evaluation of Reward-Modulated Neuromorphic Computing for Vertical Control

7

Methodology

These preliminary experiments aim to give an indication of the learning performance of reward-modulated SNNs for optical flow control of MAVs, a task for which they have not been employed before. Different reward-modulated learning rules will almost certainly perform differently, and it is crucial to the success of this thesis to find out each rule's strengths and weaknesses, and to gain a better understanding of what composes a well-performing rule. To this end, a simple simulator has been devised that can serve as a benchmark for these rules.

This chapter will first present an outline of the performed analysis in Section 7.1, after which the SNN simulation framework of choice and the simulation environment for benchmarking the learning rules are discussed in turn in Sections 7.2 and 7.3, respectively. The following chapters will provide an in-depth account of the procedures and the outcome of the learning rule comparison. Chapter 8 covers the implementation of the different reward-modulated learning rules and the configuration of the SNN, and lists the results for each tested variant. These results and their implications are subsequently discussed in Chapter 9, along with the simulation set-up.

7.1 Outline of the analysis

As a heuristic for real-world optical flow control of an MAV, these preliminary experiments aim to benchmark reward-modulated learning rules for SNNs in their ability to learn vertical control of an MAV[1] in a simple simulated environment, in order to make it hover. The idea is that the performance of various learning rules could serve as an indicator as to what would work when training an SNN to perform vertical optical flow control in a real-world environment (possibly using an event-based camera and a neuromorphic chip), a task for which these networks have not been applied so far.

Starting from the highest level of abstraction, vertical control of an MAV is nothing more than control of a second-order integrator under influence of gravity, where lifting forces of varying magnitude can be applied to either decrease or increase its altitude. This is the main idea behind the vertical simulator used in these preliminary experiments, and the task of the SNN controller will be to select the appropriate force in order to keep the MAV at a constant altitude, based on an observation of either altitude and vertical speed or divergence (see Section 2.1.2). A capable reward-modulated learning rule, in combination with a well-thought-out reward function, will be crucial in learning these desirable behaviours.

Different configurations in terms of learning rule, neural architecture and hyperparameters will be tested. A comparison based on these results will then give an indication as to which configuration shows the most potential for learning real-world optical flow control.

[1] From here on, we define 'MAV' as a quadrotor MAV that can ascend/descend in a vertical line, and 'vertical control' as control of this ascent/descent.

7.2 Spiking neural network simulator

The SNN simulator of choice for this preliminary evaluation is BindsNET (Hazan et al., 2018), which was already discussed in Section 5.3.2. The flexibility of BindsNET in terms of adding and tweaking neuron models and learning rules, along with the fact that it has been written in Python, allows for fast prototyping and the testing of many different configurations, which is what these experiments are all about. Apart from this, BindsNET is being actively developed[2], and its developers are keen to help with any problems and open to suggestions. To demonstrate BindsNET's functioning, Listing 7.1 shows a minimal working example of an SNN of LIF neurons that receives spike trains generated by a Poisson process as input. Note that the default simulation time step (at which the difference equations describing, e.g., neuron dynamics, are calculated) is set to $\Delta t = 1$ ms.

Another advantage of BindsNET is its seamless integration with OpenAI Gym[3] environments, which, apart from covering many classic RL problems, allow users to easily create their own environments. The next section will illustrate this further.

Listing 7.1: A minimal working example of an SNN simulated with BindsNET.

```
import torch

from binsnet.encoding import poisson
from bindsnet.network import Network
from bindsnet.network.nodes import Input, LIFNodes
from bindsnet.network.topology import Connection


# Instantiate network.
network = Network(dt=1.0)

# Create layers.
X = Input(100)  # Input layer.
Y = LIFNodes(100)  # Layer of LIF neurons.
C = Connection(source=X, target=Y)  # Connection from X to Y.

# Add everything to the network.
network.add_layer(layer=X, name='X')
network.add_layer(layer=Y, name='Y')
network.add_connection(connection=C, source='X', target='Y')

# Create Poisson-distributed spike trains.
rates = 15 * torch.rand(100)  # Random Poisson firing rates for 100 input neurons.
spikes = poisson(datum=rates, time=500.0)  # Create 500 ms Poisson spike trains.

# Simulate network on the generated spike trains.
network.run(inpts={'X' : spikes}, time=500.0)
```

7.3 Vertical control simulation environment

To get an idea of their ability to train an SNN to perform optical flow control, different configurations of each learning rule will be tested on a problem that is similar in some regards, but heavily simplified in others, namely vertical control of a second-order integrator (which is to represent an MAV) with the goal of keeping it at a constant altitude. The remainder of this section will illustrate the various aspects of the simulator implementing this problem.

[2] https://github.com/Hananel-Hazan/bindsnet
[3] https://gym.openai.com/

7.3.1 Environment characteristics

Figure 7.1 shows the forces acting on the 'MAV' of mass m. In order to hover, the thrust T provided by the propellers, which is the action chosen by the SNN controller, has to balance gravity (or weight) $W = mg$. There are two goal settings for which this task can be carried out: 1) achieve hover at any altitude h, such that we are only concerned with $\dot{h} = 0$ ms^{-1}, or 2) achieve hover at a certain goal altitude h_{goal}. Learning rules will be tested in both settings, but for the explanation of the environment's characteristics we will focus on the latter. The former will be treated in Section 7.3.2.

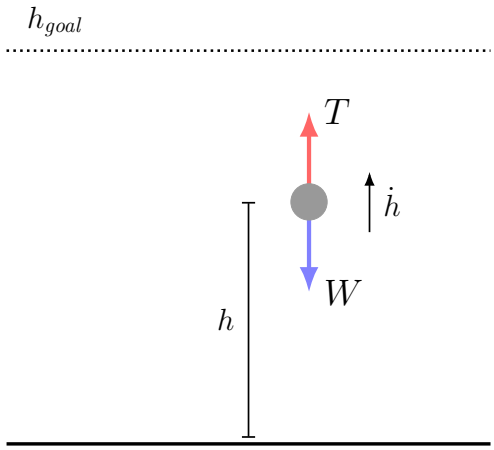


Figure 7.1: Schematic overview of the vertical simulation environment.

The goal altitude h_{goal} is set beforehand, and the starting position h_0 and vertical speed $\dot{h}_0$ are chosen randomly within a certain range. By default, $(h_0, \dot{h}_0) = (10, 0)$ with variations of 1 m and 0.1 ms^{-1}, respectively. Therefore, the SNN will have to figure out that, in case it starts off below the goal altitude, it first has to increase T to ascend, and decrease it to match W as soon as h_{goal} is reached. Each time the MAV flies goes out of bounds, which we define as $h > 20$ m, $h \leq 0$ m or $|\dot{h}| > 10$ ms^{-1}, the problem restarts, giving it an episodic setting (recall from Section 4.2.1). The environment will also reset if t reaches $T = 10.0$ s, with steps of $\Delta t = 0.01$ s.

7.3.2 State observation

The MAV can perceive its state $S_t \in \mathcal{S}$ in one of two predefined ways: 1) as a scalar value of divergence[4] $D = -2\dot{h}/h$, which only allows the learning of hover ($D = 0$ s^{-1}) in general, or 2) as a tuple $(h, \dot{h})$ of altitude and vertical speed, allowing the SNN controller to learn to hover at a goal altitude h_{goal}. Even though both make use of the same information, the former, while seemingly simpler, may be more similar to actual optical flow control. From now on, we will refer to both goal settings as the *zero-divergence problem* and the *goal altitude problem*. As mentioned, we will test the performance of learning rules for both. The way in which the SNN encodes the state into spike trains will be covered in Section 8.2.2.

7.3.3 Action selection

The action $A_t \in \mathcal{A}$ (indicating available actions are independent of state) selected by the SNN controller determines the lifting force T. The selected force is bounded to prevent it from becoming unrealistically large or negative (which of course is impossible in the case of a lifting force provided by propellers). Recall that the action space $\mathcal{A}$ can be discrete or continuous, meaning that the SNN can either choose from a distinct number of different forces, or can select any force within the action bounds. The

[4] To prevent $D \to \infty$ s^{-1} for $h \to 0$ m, we shift the altitude bounds mentioned previously upwards by 0.1 m, resulting in allowed altitudes $0.1 < h \leq 20.1$ m.

implications of these different spaces, as well as the mechanism used by the SNN to decode its spike trains into actions, is further discussed in Section 8.2.3.

7.3.4 Reward function

A well-designed reward function is critical for learning desired behaviour. Not only does the reward function have to capture the goal that has to be reached within a problem, it also has to provide the agent (SNN controller in our case) with a sense of what actions are desirable. The most effective way to do this is through a dense reward function, as was mentioned in Section 4.2.8.

For the problem of vertical control presented here, the goal for each of the two settings mentioned in Section 7.3.2 is clear:

- **Zero-divergence problem:** Achieve and keep $D = 0$ s^{-1} within the defined state bounds and allotted time.
- **Goal altitude problem:** Achieve and keep $h = h_{goal}$ and $\dot{h} = 0$ ms^{-1} within the defined state bounds and allotted time.

Obviously, these problems each require different reward functions. Before suggesting some for both of them in turn, we look at some general considerations. For instance, apart from the dense reward functions specific to each setting, there can be rewards/punishments for specific events (see Section 4.3.1), such as punishment for crashing into the ground or flying out of bounds. However, the use of eligibility traces in reward-modulated learning rules implies that the actions taken slightly before this punishment will be affected most (see Section 5.2.4), whereas it is very likely that things went south quite some time before that. Because of this, and to keep everything as simple as possible, event-specific rewards will not be considered here. By doing so, we follow Rodriguez-Ramos et al. (2018), Rodriguez-Ramos et al. (2019), who successfully learnt vision-based control of an MAV using a reward function based on the minimisation of some global metric, such as the magnitude of divergence or the difference in altitude with h_{goal}. The globality and simplicity of this approach make it biologically more plausible, which is appealing.

Zero-divergence problem

With the goal of $D = 0$ s^{-1}, several reward functions can be defined based on the magnitude of the current divergence and some relationship, e.g., linear or quadratic. Figure 7.2 proposes some of the general shape:

$$R(D) = R_{goal} - \beta_1 \left[\sqrt{D^2} \right]^{\beta_2} \tag{7.1}$$

with the βs defining constants to allow different variants. Higher βs give functions that punish deviations from the goal more severely, leading to a reward space where most states correspond to punishment. For instance, setting $\beta_2 = 2$ makes R decrease quickly, resulting in almost a 'hard' bound on the values of divergence that are preferred. Note that, when taking the reward for $D = 0$ s^{-1} (the maximum reward in the goal state) as $R_{goal} = 10$, the maximum accumulated reward obtained at the end of an episode is $R_{sum} = \sum_{t=1}^{N} R_t = 1000 \cdot 10 = 10000$. Another possibility would be to set it $R_{goal} = 0$, such that there is only punishment, and the reward signal would corresponding to the error with the goal.

Goal altitude problem

Similarly, we can define rewards for the goal altitude problem, with the function consisting of separate terms for altitude errors and vertical speed errors. Figure 7.3 illustrates several candidates, which follow the general shape:

$$R\left(h, \dot{h}\right) = R_{goal} - \beta_1 \left[\sqrt{\left(h - h_{goal}\right)^2 + \beta_3 \dot{h}^2} \right]^{\beta_2} \tag{7.2}$$

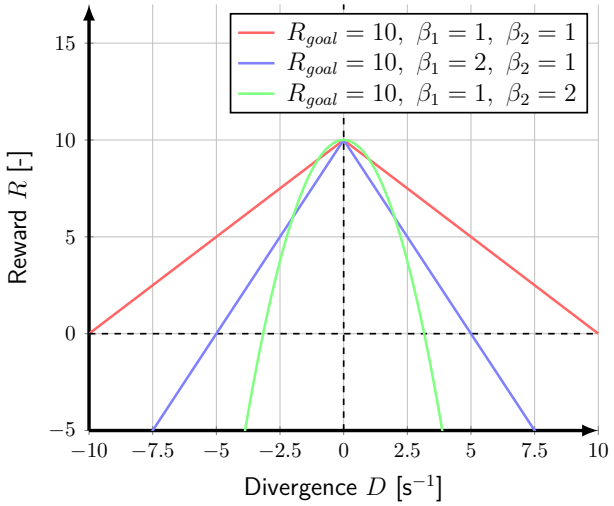


Figure 7.2: Possible reward functions for the zero-divergence problem.

The same things mentioned about the βs in Equation (7.1) apply here: increasing β_1 and β_2 makes that deviations are punished more heavily. Additionally, however, we can balance the importance of altitude and vertical speed errors through β_3. Setting $\beta_3 = 0$ means that reward is not dependent on $\dot{h}$, which could result in a more difficult learning problem, given that less information is incorporated in the reward function.

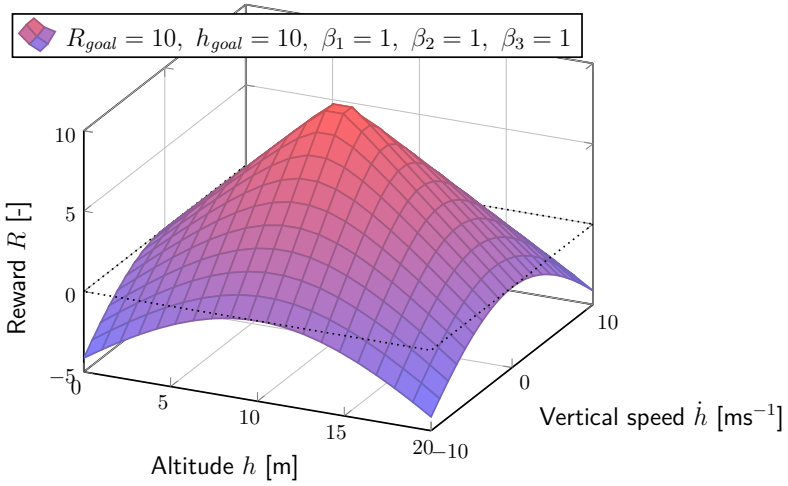


Figure 7.3: Possible reward function for the goal altitude problem.

8

Vertical control with reward-modulated neuromorphic computing

With the simulation environment, serving as a proxy for real-world optical flow control, well-defined, we can focus on the actual evaluation of the various reward-modulated learning rules and their ability to train an SNN to solve the simulated problem of vertical control. To avoid any ambiguity, it is imperative that all components of the SNN controller or agent (the used learning rule, the SNN itself and the way it interacts with the environment) are also defined properly. This chapter will see to that.

First, the learning rules in question and their implementations are treated in Section 8.1. Second, Section 8.2 goes over the configuration of the SNN, including neuron models and its interface with the environment. Finally, Section 8.3 covers the settings and details involved in the performed simulations, of which the results are presented in Section 8.4.

8.1 Reward-modulated learning

Reward-modulated learning rules were already discussed in their continuous form in Section 5.2.4, however they need to be discretised in order to be used in BindsNET. This discretisation, as well as possible variations of each rule, is described here. Mechanisms for reward prediction, which is needed to update weights based on RPE (see Section 5.2.4), are also given. Note that for this preliminary analysis, only covariance-based learning rules will be considered. Because these learn slower than the TD-based rules (Frémaux & Gerstner, 2016; Frémaux et al., 2013), they are well-suited for getting a sense of the tractability of the problem. If we can demonstrate learning for the covariance-based rules, their TD-based counterparts will certainly be capable of learning. Another reason for leaving out the TD-based rules is their critic network, which severely complicates implementation.

8.1.1 R-STDP

A well-explained discretisation of R-STDP is given by Florian (2007). He remarks that, in simulation, it is useful to introduce variables P_j^{pre} and P_i^{post} to keep track of the influence of pre- and postsynaptic spikes, respectively, instead of keeping sums of these spikes in memory (as is implied by Equation (5.14)). This simplification, along with writing the eligibility trace decay as an exponential function, leads to:

$$w_{ij}(t+\Delta t) = w_{ij}(t) + \alpha S(R(t+\Delta t))\, e_{ij}(t+\Delta t) \tag{8.1}$$

$$e_{ij}(t+\Delta t) = e_{ij}(t)\exp\left(\frac{-\Delta t}{\tau_e}\right) + U_{ij}(t+\Delta t) \tag{8.2}$$

$$U_{ij}(t+\Delta t) = P_j^{pre}(t+\Delta t)\, y_i(t+\Delta t) + P_i^{post}(t+\Delta t)\, x_j(t+\Delta t) \tag{8.3}$$

$$P_j^{pre}(t+\Delta t) = P_j^{pre}(t)\exp\left(\frac{-\Delta t}{\tau_{pre}}\right) + A_{pre}x_j(t+\Delta t) \tag{8.4}$$

$$P_i^{post}(t+\Delta t) = P_i^{post}(t)\exp\left(\frac{-\Delta t}{\tau_{post}}\right) + A_{post}y_i(t+\Delta t) \tag{8.5}$$

where Δt is the simulation time step, and x_j and y_i are binary variables indicating pre- and postsynaptic spikes, respectively. Note that Florian (2007) did not strictly follow any discretisation method, such as forward Euler, in particular in his approach. The result is, however, very similar. He does put Δt as a factor in the weight update, but we let this be absorbed by the learning rate α. The variables A_{pre} and A_{post} determine the contribution of pre- and postsynaptic spike traces, respectively (analogous to A_+ and A_- in Equation (5.14)), and can again be set to differentiate between additive and multiplicative STDP (see Section 5.2.2).

8.1.2 R-max

Vasilaki et al. (2009) present a discretisation of their R-max rule, which makes use of the SRM neuron (whose discretisation will be covered in Section 8.2.1). Again, discretisation is not performed according to any one method. Note that we introduce, as with R-STDP, a variable to keep track of spikes:

$$w_{ij}(t+\Delta t) = w_{ij}(t) + \alpha S(R(t+\Delta t))\, e_{ij}(t+\Delta t) \tag{8.6}$$

$$e_{ij}(t+\Delta t) = \left(1 - \frac{\Delta t}{\tau_e}\right) e_{ij}(t) + U_{ij}(t+\Delta t) \tag{8.7}$$

$$U_{ij}(t+\Delta t) = \left(y_i(t+\Delta t) - \frac{\hat{\rho}_i(t+\Delta t)}{1+\frac{\tau_c}{\Delta t}\hat{\rho}_i(t+\Delta t)}\right) P_j^{pre}(t+\Delta t) \tag{8.8}$$

$$P_j^{pre}(t+\Delta t) = P_j^{pre}(t)\exp\left(\frac{-\Delta t}{\tau_{pre}}\right) + A_{pre}x_j(t) \tag{8.9}$$

where P_j^{pre} takes the role of the sum of presynaptic spikes in Equation (5.17) (identical to Equation (8.4)), with $A_{pre}\exp(-\Delta t/\tau_{pre})$ functioning as the kernel $\varepsilon(s)$ of the SRM (see Equation (5.5)). Furthermore, $\hat{\rho}_i$ is the discrete version of the escape noise $\rho(u_i(t))$, which was already given by Equation (5.8). Also note that the decay term of the eligibility trace, $(1-\Delta t/\tau_e)$, is almost identical in value to $\exp(-\Delta t/\tau_e)$, and that τ_c can be used to balance reward-sensitive learning and Hebbian learning. Vasilaki et al. (2009) remarked that both have their use, and so we will test different values of τ_c.

8.1.3 Reward prediction

The success signal $S(R(t+\Delta t))$ in Equations (8.1) and (8.6) needs to encode RPE in order to achieve the best learning performance (as discussed in Section 5.2.4). Equation (5.16) suggested that this could be done with a sparsely updated moving average of past rewards, which Vasilaki et al. (2009) defines as:

$$S(R(t+\Delta t)) = R(t+\Delta t) + \bar{R}(n) \tag{8.10}$$

$$\bar{R}(n) = \left(1 - \frac{1}{\sigma_r}\right)\bar{R}(n-1) + \frac{1}{\sigma_r}R_{step}(n) \tag{8.11}$$

where $\bar{R}(n)$ indicates that the moving average is updated after every episode n, σ_r is the width of the averaging window and $R_{step}(n) = \frac{R_{sum}(n)}{N_{steps}}$ is the average reward per step in episode n.

As mentioned, other methods of doing reward prediction, such as a second SNN serving as critic network, are possible, but these will not be implemented as of now.

8.2 Network configuration

With synaptic plasticity out of the way, the neuron and synapse models from Section 5.1.2 have to be adapted for use in discrete-time simulation. Furthermore, the interface between the SNN and the simulation environment has to be defined. State S_t has to be encoded in spike trains that are fed to the input layer of the SNN, and spike trains coming from the output layer have to be decoded to an action A_t. This section looks into these issues.

8.2.1 Neuron models and synapses

Before going over the discretisation of some of the most-used neuron models, we have to discuss the synaptic efficacy or weight w_{ij} of the connections between these neurons. Judging from any of the above-mentioned learning rules, weights will continue to increase indefinitely if positive rewards keep coming in. The simplest solution to this would be to bound weights within an interval $[w_{min}, w_{max}]$, however this would lead to a bimodal weight distribution where most weights end up near the bounds. Multiplicative learning rules (see Section 5.2.2) can slow this process somewhat, but ultimately also lead to bimodal distributions (e.g., Diehl & Cook, 2015; Kheradpisheh et al., 2018). The homeostasis mechanism devised by Paredes-Vallés et al. (2019) does lead to a unimodal and balanced weight distribution, but is more elaborate to implement. For this preliminary evaluation, we will therefore stick with simply bounding weights within an interval. This still leaves the choice between allowing only excitatory synapses (positive weights) or also inhibitory synapses (negative weights).

LIF

The LIF neuron represented by Equation (5.2) is discretised by Florian (2007), where he again makes use of an exponential decay instead of following the forward Euler method:

$$u_i(t) = u_{rest} + [\, u_i(t - \Delta t) - u_{rest} \,] \exp\left(\frac{-\Delta t}{\tau_m} \right) + A_{pre} \sum_j w_{ij} x_j(t) \tag{8.12}$$

where, compared to Equation (5.2), we replace R/τ_m with A_{pre} to set the magnitude of the contribution by presynaptic spikes. As opposed to Florian (2007), we define the contribution of spikes to be immediate, such that the binary variable indicating a spike is $x_j(t)$ instead of $x_j(t - \Delta t)$, and in this follow the definition by Vasilaki et al. (2009) more closely.

When a neuron's membrane potential $u_i(t)$ reaches the threshold θ, it emits a spike, and $u_i(t)$ is reset to u_{rest}. A refractory period can be implemented by a simple counter, which prevents the neuron from firing/building up membrane potential for a certain number of time steps.

SRM$_0$

The derivation of the R-max learning rule depends on the stochastic firing threshold of the SRM neuron, which means that a discretised form of its dynamics is needed for simulation. As was shown by Gerstner and Kistler (2002), the SRM is more general than the LIF model, and can represent it given the right kernels. To make sure learning rules are compared on an even playing field, we let the SRM$_0$ (the simpler version which removes the dependence of the kernels on the last postsynaptic spike, see Section 5.1.2) model the LIF neuron using exponential kernels of the form $\exp(-\Delta t/\tau)$. This results in the same membrane potential equation as the discretised LIF model (defined by Equation (8.12)). The remaining difference lies in the spiking, which is governed by a stochastic threshold that can be formulated by combining Equations (5.7) and (5.8):

$$\hat{\rho}_i(t) \doteq 1 - \exp(-\rho(u_i(t))\,\Delta t) = 1 - \exp\left(-\rho_0 \exp\left(\frac{u_i(t) - \theta}{\Delta\theta} \right) \Delta t \right) \tag{8.13}$$

In case the neuron fires, its membrane potential is reset to u_{rest} and a possible refractory period can be started based on a counter of time steps.

8.2.2 Encoding state

As Section 7.3.2 explained, the state of the environment S_t is represented by either a scalar value of divergence D, or a tuple $\left(h, \dot{h}\right)$ of altitude and vertical speed. Frémaux et al. (2013), Vasilaki et al. (2009) come up with a solution in the form of neurons whose firing rate is dependent on the current state, taking inspiration from *place cells* and *grid cells* in the animal brain. A short explanation of both types of cells will illustrate that neither exactly corresponds to the implementation by these authors. See Moser, Kropff and Moser (2008) for more on the difference between place and grid cells.

Hippocampal place cells (HPCs) were first described by O'Keefe and Dostrovsky (1971) as neurons that encode a spatial map in their firing. Different locations in the environment correspond to a different neuron firing, implying that each place cell corresponds to a specific location. On the other hand, grid cells, present in the entorhinal cortex and discovered by Hafting, Fyhn, Molden, Moser and Moser (2005), are laid out as a grid over the spatial environment, and all cells fire each time the current location coincides with one of them. This means that individual locations are encoded in the firing patters of all grid cells combined, resulting in a sort of 'coordinate system' that also allows for measuring distances. Other works indicate that grid cells encode many more types of information that are spatial in some way (e.g., Aronov, Nevers & Tank, 2017; Constantinescu, O'Reilly & Behrens, 2016; Killian, Jutras & Buffalo, 2012).

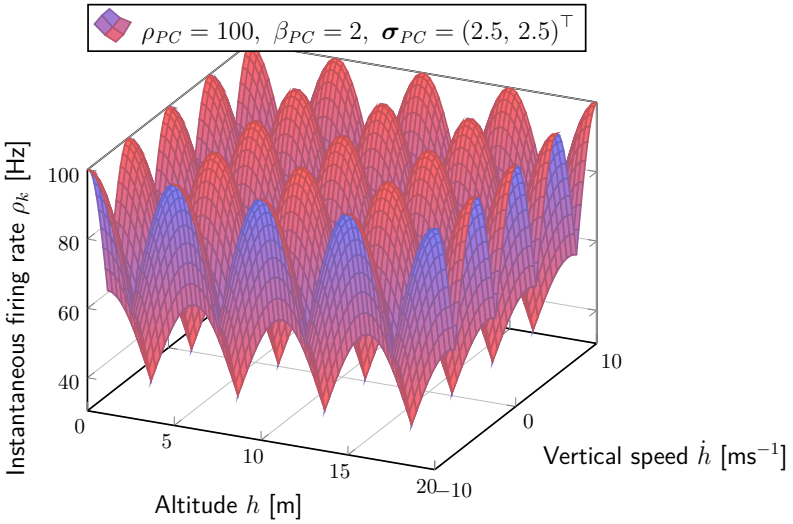


Figure 8.1: Instantaneous firing rates ρ_k of place cells as a function of state $\left(h, \dot{h}\right)$ for a possible configuration. $\mathbf{s}_k$ is spaced in steps of 5 for both h and $\dot{h}$.

The place cells[1] implemented Frémaux et al. (2013), Vasilaki et al. (2009) are distributed uniformly over the state space $\mathcal{S}$. The activity of a place cell k is reflected by its instantaneous firing rate ρ_k, which is modelled as an *inhomogeneous Poisson process* dependent on the distance between the current state and the centre of the place cell (Frémaux et al., 2013):

$$\rho_k\left(\mathbf{s}\left(t\right)\right) = \rho_{PC} \exp\left(-\frac{\| \mathbf{s}\left(t\right) - \mathbf{s}_k \|^2}{\beta_{PC} \boldsymbol{\sigma}_{PC}^2}\right) \tag{8.14}$$

where ρ_{PC} is the maximum firing rate, $\mathbf{s}\left(t\right)$ the state vector at time t, $\mathbf{s}_k$ the centre of place cell k and $\boldsymbol{\sigma}_{PC}$ its width vector (for different states variables) with a scaling constant β_{PC}. Figure 8.1 illustrates the way place cells are laid out over the state space for $\left(h, \dot{h}\right)$. Analogous to the instantaneous firing

[1] We stick to the naming by Frémaux et al. (2013), Vasilaki et al. (2009) for consistency reasons.

probability or firing rate $\rho(u_i(t))$ of SRM neurons, the probability that a place cell spikes in $[t, t+\Delta t]$ can be obtained by integration, i.e., $\int_t^{t+\Delta t} \rho_k(\mathbf{s}(t'))\,dt' \approx \rho_k(\mathbf{s}(t))\,\Delta t$.

8.2.3 Decoding actions

Recall from Section 7.3.3 that the action A_t represents the thrust T exerted on the MAV. We prevent this force from becoming unrealistically large by bounding it within an interval $[T_{min}, T_{max}]$. Apart from this, there are multiple approaches to decoding spike trains coming from the output layer of the SNN, in order to end up with a scalar force. Here we will present two approaches that are dependent on the nature of the action space $\mathcal{A}$ being discrete or continuous.

Discrete action space

In case of a discrete action space, the agent (SNN) has a discrete number of forces to choose from at each time step. We model this in the simplest way possible, namely by giving the SNN a single actor neuron, $N_{actor} = 1$, as output layer, and choosing an action A_t based on whether it spikes:

$$A_t = \begin{cases} \frac{T_{min} + \frac{T_{min}+T_{max}}{2}}{2} & \text{if } y_h(t) = 0 \\ \frac{T_{max} + \frac{T_{min}+T_{max}}{2}}{2} & \text{if } y_h(t) = 1 \end{cases} \tag{8.15}$$

where $y_h(t)$ is a binary variable indicating an output spike by actor neuron h, and the actions are chosen in such a way that they are balanced around the centre of the action bounds. This means that the SNN will have to figure out which combinations of actions make the MAV ascend, descend or hover. Note that different values for the bounds T_{min} and T_{max} can lead to behaviours that are easier or harder to learn for the agent: say the force needed for hover is roughly in the middle of the action interval, then alternating the two actions will lead to something close to hover. If this is not the case, more elaborate action patterns may have to be learnt.

Continuous action space

Continuous action selection with any amount of output neurons is possible with a method called *population vector coding* (Georgopoulos, Schwartz & Kettner, 1986), where each of the actor neurons that make up the SNN output layer 'vote' for their preferred action A_t by spiking. Frémaux et al. (2013), Vasilaki et al. (2009) implemented this by keeping track of the postsynaptic spike traces of each actor neuron $Y_h(t)$, and deciding on an action by taking a weighted average of a vector of possible actions:

$$A_t = \frac{\sum_h P_h^{act}(t)\, T_h}{\sum_h P_h^{act}(t)} \tag{8.16}$$

$$P_h^{act}(t) = P_h^{act}(t - \Delta t) \exp\left(\frac{-\Delta t}{\tau_{act}}\right) + A_{act} y_h(t) \tag{8.17}$$

where P_h^{act} is the postsynaptic spike trace of each actor neuron (identical to Equation (8.5), but with *act* indicating it belongs only to actor neurons). The action vector is defined based on the action bounds, i.e., $\mathbf{T} = (T_1, \ldots, T_{N_{actor}})^\top$ with $T_h = T_{min} + h\frac{T_{max}-T_{min}}{N_{actor}}$.

8.3 Simulation settings

In order to gain insight into the strengths and weaknesses of the various learning rules, and to compare the difficulty of the zero-divergence and goal altitude problem, we investigate the effect of changing certain hyperparameters on the learning curve. This is done for the following four cases:

1. **Discrete action space + goal altitude problem:** Varying T_{min} and T_{max} to see whether the rule can learn different action patterns.

2. **Discrete action space + zero-divergence problem:** Varying τ_e to determine the importance of eligibility traces in accelerating learning.

3. **Continuous action space + goal altitude problem:** Varying h_{goal} to force the agent to first ascend/descend and then achieve hover, increasing the difficulty of the learning problem.

4. **Continuous action space + zero-divergence problem:** Varying h_0 to investigate whether the rule allows learning of different gains needed for optical flow control at different heights (de Croon, 2016).

Both R-STDP and R-max will be compared in each of these cases, and the default configurations are given in Appendix A. Note that, since the neurons used in each rule differ in spiking nature (deterministic versus stochastic), there might be slight differences in the configuration of each rule for a certain case, in order to ensure at least minimum spiking and thus learning. However, these differences will be kept to a minimum to allow fair comparison. Table 8.1 gives the precise hyperparameter settings that will be covered for each case. In the interest of time, no repetitions using different weight initialisations will be performed, as the nature of these experiments is more exploratory. Any follow-up experiments must therefore look at the effect of this.

Two pieces of hardware will be used for running all simulations. These are:

- Laptop running Ubuntu 18.04 LTS, equipped with Intel i7-7700HQ quad-core CPU (eight virtual CPUs) and 16 GB of memory.
- Google Compute Engine[2] server running Ubuntu 18.04 LTS, equipped with Intel Xeon scalable processors (96 virtual CPUs) and 86 GB of memory.

As far as software versions are concerned: Python 3.6 (included by default on Ubuntu 18.04 LTS) is used as programming language. The to-be-used variants of BindsNET and the vertical simulator can be found in their respective repositories[3].

Table 8.1: Hyperparameter variations for each simulation case, in addition to the default parameters specified in Appendix A.

Case	Hyperparameters
Discrete + goal altitude	$[T_{min}, T_{max}] = [0, 30]$, $[0, 40]$, $[0, 50]$ N
Discrete + zero divergence	$\tau_e = 100$, 500, 800 ms
Continuous + goal altitude	$h_{goal} = 5$, 10, 15 m
Continuous + zero divergence[4]	$h_0 = 1$, 5, 10, 15 m

8.4 Results

Here, we presents the simulation results for the learning rules and cases described in the previous sections of this chapter. The specific settings for each case can be found in Section 8.3, with the default hyperparameters coming from the tables in Appendix A. Learning curves based on a moving average (100 episodes) of accumulated rewards are compared in order to study the learning capacity, or reward-maximisation capability, of the various settings. The smoothing of these curves was performed to make differences in learning more clear. Note that the maximum reward that could be obtained in each case is $R_{goal} \cdot N_{steps} = 10 \cdot 1000 = 10\,000$.

[2] `https://cloud.google.com/compute/`

[3] BindsNET: `https://github.com/Huizerd/bindsnet/tree/prelim`, vertical simulator: `https://github.com/Huizerd/vertical`

[4] For $h_0 = 1$ m, the usual stochasticity of the starting position is decreased from 1 m to 0.5 m, to prevent starting on the ground.

8.4.1 Discrete action space + goal altitude problem

Figure 8.2 shows the learning curves belonging R-STDP and R-max for the discrete action space + goal altitude case. Immediately, it becomes obvious that action bounds of $[0, 30]$ N, resulting in discrete actions of 7.5 N and 22.5 N (see Equation (8.15)), allow the most learning by either rule. Given an MAV weight of $W = mg = 2 \cdot 9.81 = 19.62$ N, it seems that the problem is easiest when the force for a spiking output neuron (22.5 N) is closest to the force needed for hover. This might indicate that the default hyperparameters do not allow the network to decrease the number of spikes enough, leading to a very active actor neuron. Allowing inhibitory connections ($w < 0$) or a higher spiking threshold θ could solve this. Looking at the final distribution of the synaptic weights in Figure 8.3, we see that indeed quite some weights are set to 0, with a further decrease being prevented by the weight bounds $[0, 2]$. However, at the other side of the spectrum ($w = 2$) there is also a peak, seemingly indicating plenty of room left to decrease spiking activity. These weights could nonetheless also be the ones connected to place cells for lower altitudes, in which case they need to cause actor spikes. What can be said for sure, though, is that a bimodal distribution of weights (as we have, to some extent, here) is less preferable than a balanced, unimodal one (Paredes-Vallés et al., 2019), due to the fact that bimodality can be considered unnatural and most likely non-optimal, since otherwise one would only have to be bothered by setting weights to either of the two weight bounds.

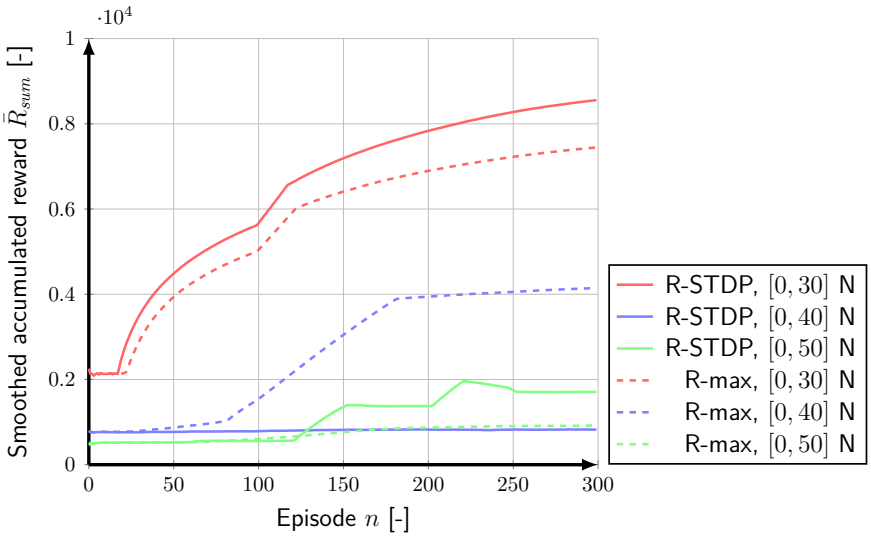


Figure 8.2: Accumulated reward moving average (100 episodes) for the discrete action space + goal altitude problem. $[T_{min}, T_{max}]$ is varied for both R-STDP and R-max.

Furthermore, it seems odd that that R-STDP slightly outperforms R-max for the edge cases of $[0, 30]$ N and $[0, 50]$ N, while performing clearly inferior for for $[0, 40]$ N. The answer may lie in the different nature of the two rules. As discussed by Frémaux et al. (2010), R-STDP is a phenomenological rule, based on experimental data, while R-max is directly derived from reward-maximising principles related to RL. In other words, R-STDP might outperform R-max in specific cases, but the generality of R-max ensures that it will always learn to some extent. Experiments performed by Frémaux et al. (2010) show that, for R-STDP without a post-before-pre window ($A_{post} = 0$, as we have here) and in the presence of a critic (here implemented as a moving average), can outperform R-max in some cases.

When observing the MAV's altitude over time in Figure 8.4 for a reward of approximately 8000 (R-STDP, $[0, 30]$), it appears that control is still rudimentary, with some drift going on. The fact that this drift has mostly been eliminated around 8 s with only 300 episodes of training is promising, however, and more training could very well lead to an elimination of drift for this case.

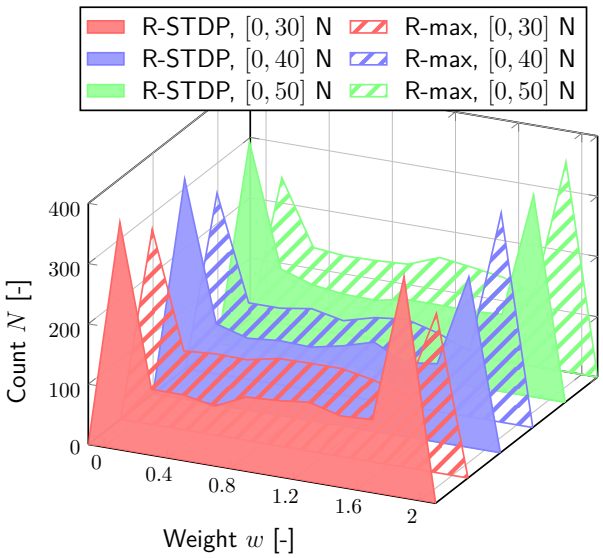


Figure 8.3: Final distribution of the synaptic weights for the discrete action space + goal altitude problem. $[T_{min}, T_{max}]$ is varied for both R-STDP and R-max. Note that the data points are the centres of the histogram bins, i.e. point $(0.1,\ 377)$ indicates 377 weights in the interval $[0.0, 0.2]$.

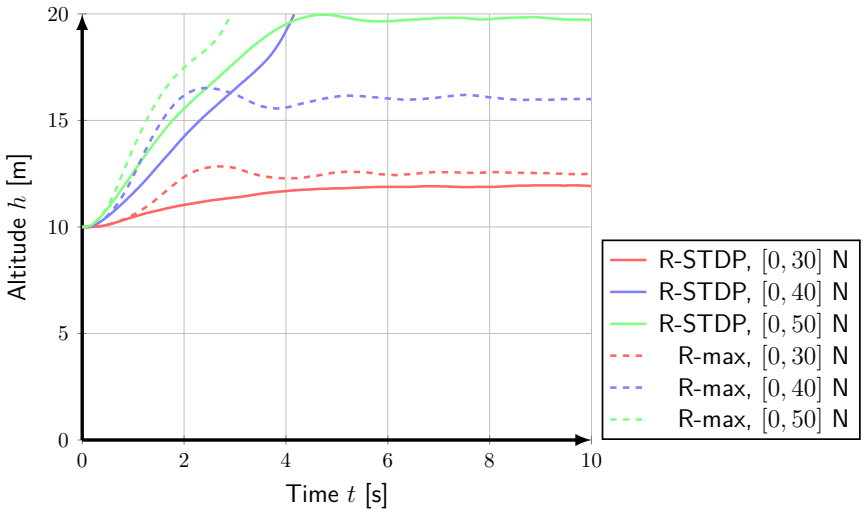


Figure 8.4: Altitude over time for the discrete action space + goal altitude problem. $[T_{min}, T_{max}]$ is varied for both R-STDP and R-max. Note that initial state is not randomised for these tests.

8.4.2 Discrete action space + zero-divergence problem

We now move to the case of learning general hover at any altitude, corresponding to a divergence of $D = 0\ \mathrm{s}^{-1}$, with the controller able to choose either an action of 10 N or 30 N. Figure 8.5 shows the learning curves belonging R-STDP and R-max for this case. While R-STDP learns to accumulate much more reward than R-max at first, it comes crashing down out of nowhere for all three values of τ_e[5]. This implicates that there seem to be some very suboptimal weight distributions close to very optimal ones, and that changing even a single weight slightly could lead to a severe decrease in performance. R-max, however, does not seem to suffer from this instability, but then again, learning is weak compared to R-STDP. As with the discrete action space + goal altitude problem, R-max seems more robust, but inferior to R-STDP for the hyperparameter configurations studied here. The stochastic nature of firing

[5]We extended training from 300 to 600 episodes to show the 'crashes' for $\tau_e = 500$ ms and 800 ms. The one for 100 ms happens after 600 episodes.

in the SRM neurons used together with R-max might very well contribute to this, being both a curse and a blessing: while it prevents crashes, it also impedes learning to some extent, because there is always some randomness going on.

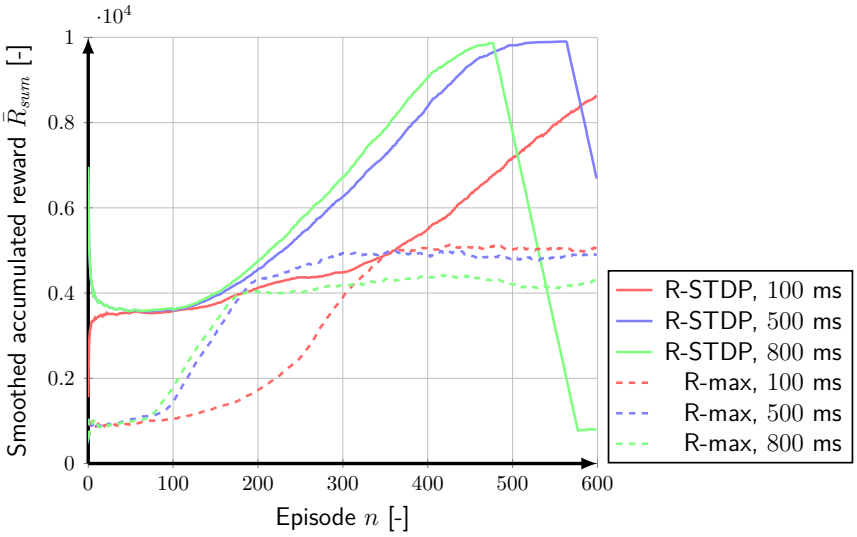


Figure 8.5: Accumulated reward moving average (100 episodes) for the discrete action space + zero-divergence problem. τ_e is varied for both R-STDP and R-max.

A conclusion that can be drawn with certainty is that longer eligibility traces (higher τ_e) lead to faster learning up to some point: rules with $\tau_e = 500$ ms learn much more quickly than those with $\tau_e = 100$ ms, while the difference between 800 ms and 500 ms is already much smaller.

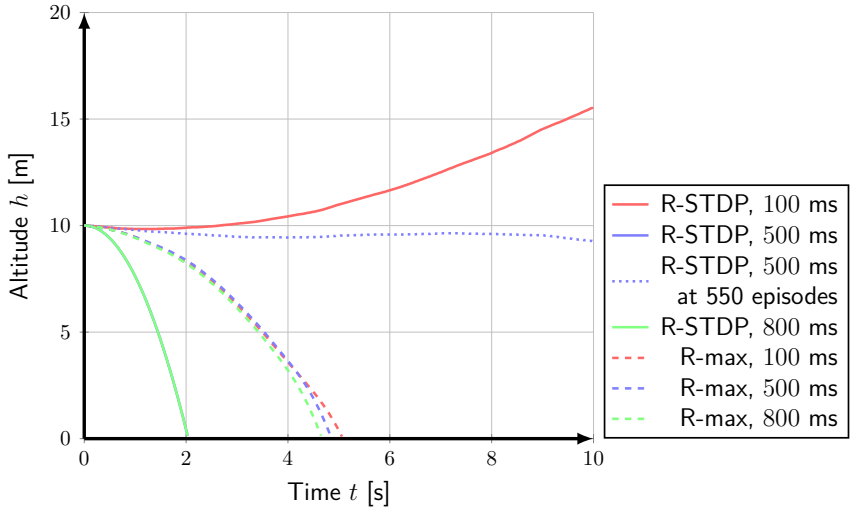


Figure 8.6: Altitude over time for the discrete action space + zero-divergence problem. τ_e is varied for both R-STDP and R-max. Note that initial state is not randomised for these tests. The dotted line indicates the performance of R-STDP with $\tau_e = 500$ ms at 550 episodes, while the plot indicating the performance at 600 episodes is hidden below the line belonging to R-STDP with $\tau_e = 800$ ms.

Looking at the altitude over time plots in Figure 8.6, it appears that the sudden drops in performance for R-STDP in the previous figure translate to quick drops in altitude here. R-max does slightly better, but seems to achieve no significant learning whatsoever. The fact that altitude decreases so quickly indicates that there is far too less spiking going on. More precisely, the fact that the combination of a discrete action space (only one actor neuron) and the perception of state as divergence (only a single state variable) leads to relatively few connections (41 compared to $41^2 = 1681$ for the discrete action

space + goal altitude problem) means that setting one of those to zero could result in a significant decrease in spiking activity. This might partly explain the sudden drops in performance in Figure 8.5. A solution to this could be the implementation of multiplicative STDP instead of additive, where the magnitude of a weight change is dependent on the magnitude of the weight (see Section 5.2.2), preventing weights from drifting too much towards the bounds, possibly keeping spiking activity up.

The performance of R-STDP with $\tau_e = 500$ ms at the peak of its learning curve (roughly 550 episodes) is very promising however, as is demonstrated by the dotted line in Figure 8.6. With almost no drift going on, this configuration so far constitutes the thing closest to hover.

8.4.3 Continuous action space + goal altitude problem

The learning rules' performance for the continuous action space + goal altitude problem is given in Figure 8.7. Given that we vary h_{goal} and that $h_0 = 10$ m, it seems that cases where the goal altitude is different from the initial altitude are more difficult to learn, as could be expected. Furthermore, it appears that ascending to 15 m poses less of a challenge than descending to 5 m, even though the bounds of the environment are equally close to both goals. While there is no explanation for this at first sight, the fact that a continuous action is selected based on a decaying weighted average makes it inherently slow to respond (but also more robust), likely contributes to the difficulty of the $h_{goal} = 5$ m and 15 m settings. Still, as was the case with the discrete action space problems, the learning by R-max is more robust, always learning something for each hyperparameter setting. Note that some of the plots in Figure 8.7 are still ascending at the 300-episode mark, indicating that more episodes could increase performance further.

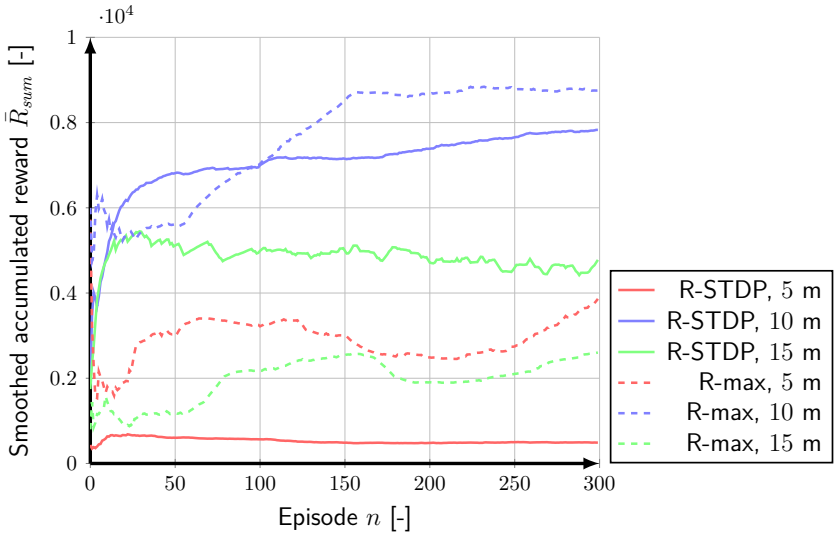


Figure 8.7: Accumulated reward moving average (100 episodes) for the continuous action space + goal altitude problem. h_{goal} is varied for both R-STDP and R-max.

The altitude over time plots in Figure 8.8 displays a multitude of behaviours, some of which auspicious. For instance, R-max with $h_{goal} = 15$ m first ascends quickly and then oscillates around some final altitude, which, although being too high, more or less constitutes the behaviour we would like our controller to demonstrate. The same goes for the dotted line, belonging to R-max with $h_{goal} = 5$ m and a slightly different initial state (caused by the randomisation of initial state also present in training, and which most likely caused a slight upwards vertical speed here, helping the controller): a quick descent is followed by an oscillation around a final incorrect altitude. The fact that this behaviour was learnt in the first place makes it reasonable to assume that more learning episodes would lead to the refinement of this behaviour, correcting the final altitude to h_{goal} and decreasing the magnitude of the oscillations. Unfortunately, R-STDP with $h_{goal} = 5$ m does not seem to have learnt anything at all. This could be caused by the setting of hyperparameters severely impeding learning for some reason, but only extensive

optimisation of these parameters could give an answer to this. Nevertheless, it would be very odd if R-STDP could not learn in combination with optimised parameters, given that it does in many of the previous cases.

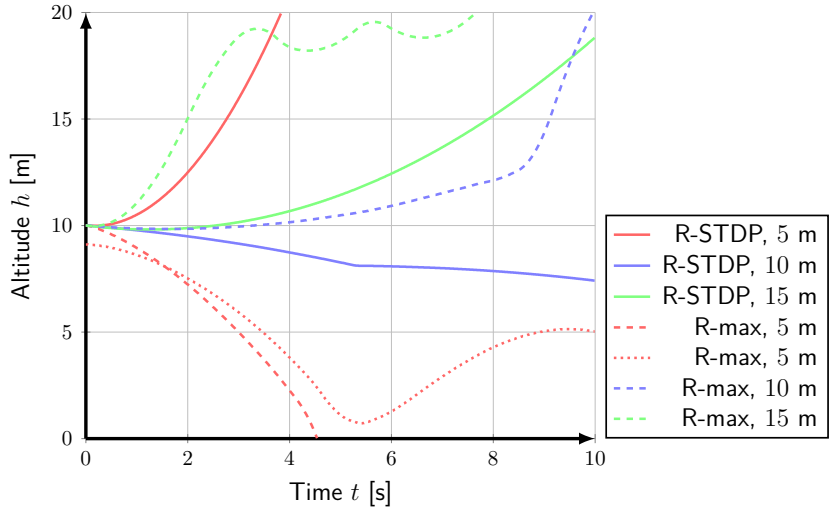


Figure 8.8: Altitude over time for the continuous action space + goal altitude problem. h_{goal} is varied for both R-STDP and R-max. Note that initial state is not randomised for these tests. The dotted line, however, indicates a test from a slightly different initial state.

8.4.4 Continuous action space + zero-divergence problem

A continuous action space in combination with perceiving state as divergence probably resembles real-world optical flow control the most, which makes this the most interesting benchmark for the various learning rules. Amazingly, it is also the setting where overall performance is best. Inspecting Figure 8.9, we see that the almost all combinations of learning rule and h_0 achieve final accumulated rewards over 9000, except for R-STDP with $h_0 = 1$ m, which suffers from a sudden drop in performance (as we have seen before). R-max, on the other hand, is able to perform well for this starting altitude, cementing its reputation as a robust learning rule.

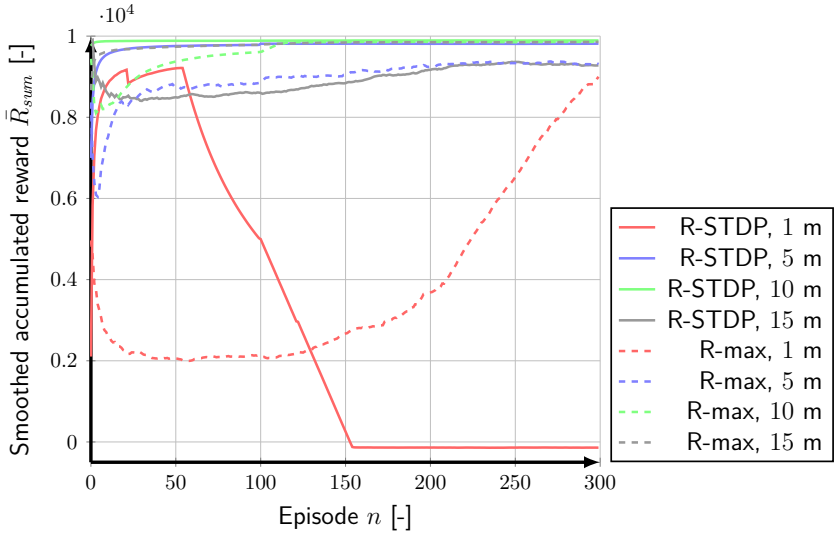


Figure 8.9: Accumulated reward moving average (100 episodes) for the continuous action space + zero-divergence problem. h_0 is varied for both R-STDP and R-max.

The final weight distributions in Figure 8.10 give a clue as to why R-STDP performs so badly for a starting altitude of 1 m: most weights have been set to zero, leading to a loss in spiking activity and most likely a loss in the ability to select actions corresponding to higher thrust. This is confirmed by the weight distribution over episodes for this setting in Figure 8.11, showing that the jump in the weights is just as sudden. As was suggested for the discrete action space + zero-divergence problem in Section 8.4.2, where R-STDP suffered from similar drops in performance, changing the STDP part of the learning rule from additive to multiplicative might help in preventing a large portion of the weights from going to zero.

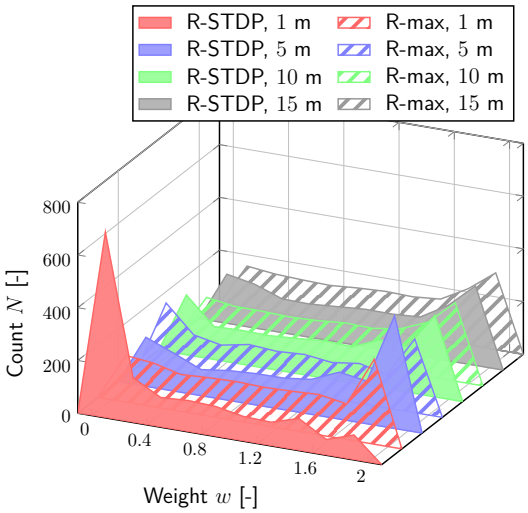


Figure 8.10: Final distribution of the synaptic weights for the continuous action space + zero-divergence problem. h_0 is varied for both R-STDP and R-max. Note that the data points are the centres of the histogram bins, i.e. point $(0.1, 704)$ indicates 704 weights in the interval $[0.0, 0.2]$.

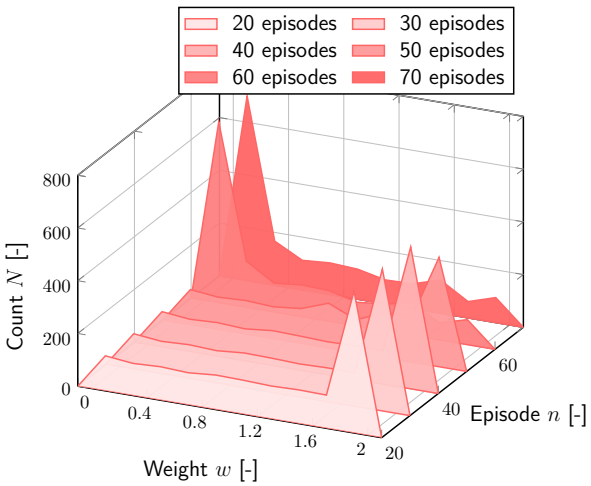


Figure 8.11: Distribution of the synaptic weights over time for R-STDP with $h_0 = 1$ m in combination with the continuous action space + zero-divergence problem. Note that the data points are the centres of the histogram bins, i.e. point $(0.1, 133)$ indicates 133 weights in the interval $[0.0, 0.2]$.

The nature of divergence-based control of altitude implies that, given a constant but nonzero value of divergence, altitude will continue to change, meaning that drift is almost inherent to this type of control, whereas direct control of altitude obviously does not suffer from this. Knowing this, the altitude over time plots in Figure 8.12 make the performance of the learning rules look worse than it actually is. Although the amount of altitude drift can most likely be decreased through more learning episodes (and possibly a more punishing reward function, such as a quadratic one), it is to be expected that some drift will always be there. Something else interesting about the plots is the apparent tendency of R-max to suffer from downwards drift (apart from $h_0 = 1$ m, where this is clearly a worse idea than upwards drift), whereas R-STDP consistently drifts upwards. The most probable cause of this is the fact that the default $|u_{rest} - \theta| = 10$ mV in the case of R-max, and 5 mV in the case of R-STDP. These values were chosen as to ensure enough spiking activity to facilitate learning, but in the case of no significant difference in accumulated reward (and thus no preference) between upwards and downwards drift, the initial increased spiking activity of R-STDP compared to R-max would lead to ascent instead of descent.

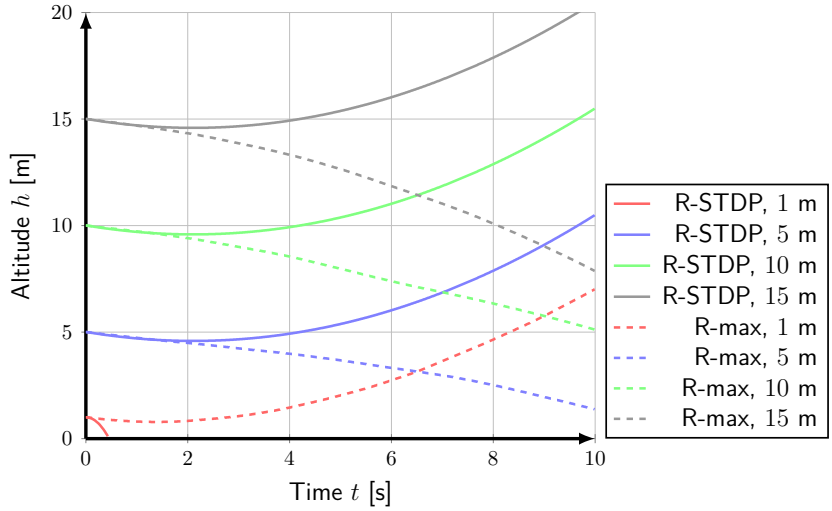


Figure 8.12: Altitude over time for the continuous action space + zero-divergence problem. h_0 is varied for both R-STDP and R-max. Note that initial state is not randomised for these tests.

9

Discussion of Preliminary Experiments

The presented preliminary experiments have given us an idea of the characteristics and learning capacity of SNNs when combined with two covariance-based learning rules, R-STDP and R-max, and applied to a simple vertical control problem. Various problem settings, differing in perception of state and action space, were observed to pose different challenges to the controller, providing us with pointers concerning the feasibility of this thesis's goal.

First, Section 9.1 will discuss the simulation set-up, and suggest possible improvements. Next, Section 9.2 looks at the learning performance for each problem setting, trying to draw lessons from the various distinctions that can be made between these settings, and assesses the feasibility of performing optical flow control with reward-modulated SNNs. Finally, Section 9.3 discusses the relation between the preliminary results and the paper presented in Part I.

9.1 Simulation set-up

The simulation set-up used in the experiments consisted of the SNN simulation framework by BindsNET (Hazan et al., 2018) and the vertical control environment discussed in Section 7.3, and implemented according to the OpenAI Gym environment conventions[1]. This allowed seamless integration of both components, and saved a tremendous amount of time compared to developing a bespoke SNN simulator and environment. BindsNET proved to be very flexible in terms of implementing new learning rules and neuron models, while any difficulties that did arise were quickly resolved with the help of its developers. The ongoing development of the package means that BindsNET will only become better in the future, with more people within the computational neuroscience community using it. This makes it very worthwhile to continue to use BindsNET, as long as constraints related to, e.g., real-time or embedded implementation do not prevent it. That being said, BindsNET's performance turned out to be more than sufficient for experimenting with different settings concurrently, and will therefore most likely be suitable for future hyperparameter optimisation.

The vertical control simulation environment specifically developed for this thesis suited the performed experiments well, and served the purpose of getting an idea of the feasibility of the learning problem and the learning capacity of the various rules. In light of making the simulator more versatile and the learning problems more realistic, however, some improvements are conceivable:

- Although stochasticity has been introduced in the initial state of the environment, as well as the firing of certain neuron models (SRM, see Section 8.2.1), state perception is currently nominal. Introduction of noise here would make the problem more realistic.
- Although the constants in the environment's reward functions allow for quite some optimisation, there might be need for a function that is more informative to the agent. For example, the reward

[1] `https://gym.openai.com/`

function devised by Rodriguez-Ramos et al. (2018), Rodriguez-Ramos et al. (2019) includes a shaping component, which informs the agent about its instantaneous progress.

- As of now, the actions selected by the SNN determine the thrust T exerted by the MAV. However, control could be made more high-level (and possibly easier to learn) by instead letting the SNN tune the gain values of a PID controller which in turn controls the thrust. Combining this approach with the perception of state as divergence would give us a control loop very similar to the ones used in many works on divergence-based control for MAVs (e.g., de Croon, 2016; Ho et al., 2018).

9.2 Performance & feasibility of reward-modulated learning for MAV control

The performed experiments allow us to make multiple important distinctions which could help us determine the feasibility of the learning problem presented in this thesis. This section goes over these distinctions, and summarises the lessons that can be drawn from them.

9.2.1 R-STDP versus R-max

R-STDP and R-max, first introduced in Section 5.2.4, are both covariance-based learning rules, meaning that they correlate candidate weight changes with obtained rewards through the covariance. They differ, however, in their foundation: whereas R-STDP springs from STDP, an unsupervised learning method derived from experimental data, R-max has been derived from RL-related reward-maximisation principles, meaning that there are theoretical guarantees to the way it maximises the accumulation of reward. While this, at first sight, seems to indicate that R-max is universally superior to R-STDP, the learning curves in Section 8.4 prove this is not true: for some of the tested hyperparameter values, neural architectures and problem settings, R-STDP (significantly) outperforms R-max in terms of learning. Nevertheless, R-max does seem to benefit from its foundation in another way, namely robustness. Whereas R-STDP suffers from severe learning instability (sudden drops in performance) for some settings of the zero-divergence problem, and shows practically no learning for two settings of the goal altitude problem, R-max manages to always learn to some extent, seemingly not suffering from any sudden drops or instabilities. It is likely that the stochasticity of the SRM neuron contributed to this at least partially, being both a curse and a blessing: while it might prevent learning instabilities and sudden crashes in performance, it could also impede learning to some extent, because there is always some randomness going on. That being said, both learning rules showed significant learning capacity over a diverse range of problems and configurations, which was one of the primary goals of these preliminary experiments. The variation of the eligibility trace length showed that this capacity could be greatly expanded by making the trace longer (up to some length).

9.2.2 Altitude + vertical speed versus divergence perception

The vertical simulation environment implemented two ways of perceiving the MAV's state: a tuple of altitude and vertical speed, and a scalar value of divergence. Given that this thesis is aimed at optical flow control, the second case is the most interesting one. Fortunately, the learning curves for the various experiments showed that, overall, learning was more successful when state was perceived as divergence. On the other hand, R-STDP did suffer from learning instability for some hyperparameter settings, and the final weight distributions belonging to these suggested that this was due to too many weights approaching zero. This indicates that a solution to the sudden performance drops could be found in adapting R-STDP to implement multiplicative weight changes, which have their magnitude dependent on the weight's magnitude, instead of additive ones, where this is not the case (see Section 5.2.2 for more details). If more rigorous measures to ensure balanced weight distributions and spiking activity are needed, the stable STDP rule by Paredes-Vallés et al. (2019), might prove valuable.

9.2.3 Continuous versus discrete actions

The action space from which the SNN selects an action was made to be either discrete, meaning it has to choose one of two possible actions by letting the single output (actor) neuron spike or not, or continuous, where the desired action followed from a weighted average of a certain action vector based on the spiking activity of the actor neurons in the output layer. Based on the number of actions to choose from, one would expect problems with the discrete action space to be learnt more easily. However, comparing the learning curves and the final accumulated rewards for both cases gives no definitive verdict on this. From the perspective of realism this is promising, since a real-world scenario would involve continuous action selection, and the results indicate that the SNN and learning rule do not experience significantly more difficulties when dealing with continuous control problems.

9.3 Implications of the analysis

The preliminary analysis conducted in this part has had an effect on the remaining course of this thesis, and has served as a basis for some of its contributions. We will go over these in this section.

First of all, consider the learning task at hand. Whereas we focused on controlling hover during the preliminary experiments, this was altered to the task of landing afterwards. There are multiple reasons for this. The main one is the inherent 'drift' that occurs when performing optical-flow-based hover control in the real world, as demonstrated by Pijnacker Hordijk et al. (2018), where an altitude difference of 0.5 m has already emerged after 30 s. The likely cause of this is a small initial velocities that, in the presence of sensor noise, does not lead to significant values of divergence, and thus hardly triggers any control. Landing, on the other hand, lets the controller experience a much higher divergence, both through larger velocities as well as the approach of the ground. It was therefore believed that the task of landing would be a more suitable, and also more relevant, demonstration of performance.

The introduction of noise in the simulation environment has been another implication of the preliminary analysis. As was done in the work of Scheper and de Croon (2020), we augmented the divergence observed by the SNN with constant and proportional noise, as well as delay, following the model by Ho and de Croon (2016). Computational jitter was added to simulate the case of a missing divergence estimate, and vertical wind was used as an external disturbance. Scheper and de Croon (2020) furthermore modelled rotor spin-up and spin-down time through a time constant, leading to non-instantaneous control. By varying all the above-mentioned quantities (noise, delay, wind, jitter, dynamics) between evaluations, transferability between simulation and the real world should be improved. For this reason, we adopted this same approach.

Given the absence of any kind of noise in our preliminary experiments, the lacking performance of the SNN controllers is likely to be at least partially the result of the employed reward-modulated learning rules. Judging from these experiments, one could say that these learning methods are currently not mature enough to solve a complex problem like optical flow control. The fact that applications of these learning rules either restrict themselves to simulated problems (Bing et al., 2020; Clawson et al., 2016) or simple discrete mappings (Zhao et al., 2018) seems to confirm this observation. Therefore, the switch was made to a neuroevolutionary framework similar to that of Scheper and de Croon (2020). The generality of this approach made for a lot of flexibility in the configuration of the spiking controllers, and also allowed us to introduce additional objectives like the minimisation of energy.

On the other hand, neuroevolution was not considered earlier because our focus was primarily on methods that were suited for fast online learning. Though there are online/real-time adaptations of, for instance, NEAT (rtNEAT, Stanley, Bryant & Miikkulainen, 2005), these have no known implementations at all for SNNs. Furthermore, some of the adaptations applied to ANNs still include an offline phase (Agogino, Stanley & Miikkulainen, 2000).

IV

Appendices

A

Default Simulation Configurations

Here, the default simulation configurations are listed for both R-STDP and R-max for the following cases (see Sections 7.3 and 8.3 for more details):

1. Discrete action space + goal altitude problem
2. Discrete action space + zero-divergence problem
3. Continuous action space + goal altitude problem
4. Continuous action space + zero-divergence problem

Tables A.1 and A.2 give the default hyperparameter values for each case.

Table A.1: Default hyperparameter values for the discrete simulation cases. Hyphens indicate no units, blanks indicate an irrelevant parameter for that case.

		Discrete + goal altitude		**Discrete + zero divergence**	
Hyperparameter	**Unit**	**R-STDP**	**R-max**	**R-STDP**	**R-max**
Δt	ms	10	10	10	10
m	kg	2	2	2	2
g	ms^{-2}	9.81	9.81	9.81	9.81
h_0, $\dot{h}_0$	m, ms^{-1}	10, 0	10, 0	10, 0	10, 0
h_{goal}	m	10	10		
R_{goal}, β_1, β_2, β_3	-	10, 1, 1, 1	10, 1, 1, 1	10, 1, 1, 1	10, 1, 1, 1
N_{steps}	-	1000	1000	1000	1000
$N_{episodes}$	-	300	300	600	600
α	-	5e−5	5e−5	1e−5	1e−5
τ_e	ms	500	500	500	500
A_{pre}, A_{post}	-	1, 0	1, -	1, 0	1, -
τ_{pre}, τ_{post}	ms	20, 20	20, -	20, 20	20, -
τ_c	ms		5		5
σ_r	-	10	10	10	10
Neuron model	-	LIF	SRM_0	LIF	SRM_0
Δt_{refrac}	ms	0	0	0	0
u_{rest}	mV	−70	−70	−70	−70
θ	mV	−50	−50	−65	−60
$\Delta\theta$	mV		1		1
ρ_0	$(\mathrm{ms})^{-1}$		1		1
τ_m	ms	20	20	20	20
N_{PC}	-	41	41	41	41
$\boldsymbol{\sigma}_{PC}$	m, ms^{-1}	1, 1	1, 1	1, 1	1, 1
ρ_{PC}	Hz	100	100	100	100
β_{PC}	-	2	2	2	2
N_{actor}	-	1	1	1	1
A_{act}	ms				
τ_{act}	ms				
T_{min}, T_{max}	N	0, 40	0, 40	0, 40	0, 40
w_{min}, w_{max}	-	0, 2	0, 2	0, 2	0, 2

Table A.2: Default hyperparameter values for the continuous simulation cases. Hyphens indicate no units, blanks indicate an irrelevant parameter for that case.

		Continuous + goal altitude		**Continuous + zero divergence**	
Hyperparameter	**Unit**	**R-STDP**	**R-max**	**R-STDP**	**R-max**
Δt	ms	10	10	10	10
m	kg	2	2	2	2
g	ms^{-2}	9.81	9.81	9.81	9.81
h_0, $\dot{h}_0$	m, ms^{-1}	10, 0	10, 0	10, 0	10, 0
h_{goal}	m	10	10		
R_{goal}, β_1, β_2, β_3	-	10, 1, 1, 1	10, 1, 1, 1	10, 1, 1, 1	10, 1, 1, 1
N_{steps}	-	1000	1000	1000	1000
$N_{episodes}$	-	300	300	300	300
α	-	5e−5	5e−5	5e−5	5e−5
τ_e	ms	500	500	500	500
A_{pre}, A_{post}	-	1, 0	1, -	1, 0	1, -
τ_{pre}, τ_{post}	ms	20, 20	20, -	20, 20	20, -
τ_c	ms		5		5
σ_r	-	10	10	10	10
Neuron model	-	LIF	SRM_0	LIF	SRM_0
Δt_{refrac}	ms	10	10	10	10
u_{rest}	mV	−70	−70	−70	−70
θ	mV	−60	−50	−65	−60
$\Delta\theta$	mV		1		1
ρ_0	$(\mathrm{ms})^{-1}$		1		1
τ_m	ms	20	20	20	20
N_{PC}	-	41	41	41	41
$\boldsymbol{\sigma}_{PC}$	m, ms^{-1}	1, 1	1, 1	1, 1	1, 1
ρ_{PC}	Hz	100	100	100	100
β_{PC}	-	2	2	2	2
N_{actor}	-	41	41	41	41
A_{act}	ms	1	1	1	1
τ_{act}	ms	20	20	20	20
T_{min}, T_{max}	N	0, 40	0, 40	0, 40	0, 40
w_{min}, w_{max}	-	−1, 1	−1, 1	0, 2	0, 2

Bibliography

Abbeel, P. & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-first International Conference on Machine Learning* (pp. 1–). ICML '04. doi:10.1145/1015330.1015430

Adelson, E. H. & Bergen, J. R. (1985). Spatiotemporal energy models for the perception of motion. *JOSA A*, *2*(2), 284–299. doi:10.1364/JOSAA.2.000284

Agogino, A., Stanley, K. & Miikkulainen, R. (2000). Online interactive neuro-evolution. *Neural Processing Letters*, *11*(1), 29–38. doi:10.1023/A:1009615730125

Akolkar, H., Ieng, S. & Benosman, R. (2018). See before you see: real-time high speed motion prediction using fast aperture-robust event-driven visual flow. *arXiv:1811.11135 [cs]*. Retrieved April 25, 2019, from http://arxiv.org/abs/1811.11135

Alkowatly, M. T., Becerra, V. M. & Holderbaum, W. (2014). Bioinspired autonomous visual vertical control of a quadrotor unmanned aerial vehicle. *Journal of Guidance, Control, and Dynamics*, *38*(2), 249–262. doi:10.2514/1.G000634

Amir, A., Taba, B., Berg, D., Melano, T., McKinstry, J., Di Nolfo, C., … Modha, D. (2017). A low power, fully event-based gesture recognition system. (pp. 7243–7252). Retrieved May 8, 2019, from http://openaccess.thecvf.com/content_cvpr_2017/html/Amir_A_Low_Power_CVPR_2017_paper.html

Armendariz, S., Becerra, V. & Bausch, N. (2019). Bio-inspired autonomous visual vertical and horizontal control of a quadrotor unmanned aerial vehicle. *Electronics*, *8*(2), 184. doi:10.3390/electronics8020184

Aronov, D., Nevers, R. & Tank, D. W. (2017). Mapping of a non-spatial dimension by the hippocampal–entorhinal circuit. *Nature*, *543*(7647), 719–722. doi:10.1038/nature21692

Bäck, T., Fogel, D. B. & Michalewicz, Z. (1997). *Handbook of evolutionary computation*. doi:10.1201/9780367802486

Baird, E., Boeddeker, N., Ibbotson, M. R. & Srinivasan, M. V. (2013). A universal strategy for visually guided landing. *Proceedings of the National Academy of Sciences*, *110*(46), 18686–18691. doi:10.1073/pnas.1314311110

Baird, E., Srinivasan, M. V., Zhang, S. & Cowling, A. (2005). Visual control of flight speed in honeybees. *Journal of Experimental Biology*, *208*(20), 3895–3905. doi:10.1242/jeb.01818

Baker, S., Scharstein, D., Lewis, J. P., Roth, S., Black, M. J. & Szeliski, R. (2011). A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, *92*(1), 1–31. doi:10.1007/s11263-010-0390-2

Baras, D. & Meir, R. (2007). Reinforcement learning, spike-time-dependent plasticity, and the BCM rule. *Neural Computation*, *19*(8), 2245–2279. doi:10.1162/neco.2007.19.8.2245

Barto, A. G. & Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, *13*(1), 41–77. doi:10.1023/A:1022140919877

Beauchemin, S. S. & Barron, J. L. (1995). The computation of optical flow. *ACM Comput. Surv. 27*(3), 433–466. doi:10.1145/212094.212141

Bellman, R. E. (1957). *Dynamic programming*. Princeton University Press.

Bengio, Y., Louradour, J., Collobert, R. & Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning* (pp. 41–48). ICML '09. doi:10.1145/1553374.1553380

Benosman, R., Clercq, C., Lagorce, X., Ieng, S. & Bartolozzi, C. (2014). Event-based visual flow. *IEEE Transactions on Neural Networks and Learning Systems*, *25*(2), 407–417. doi:10.1109/TNNLS.2013.2273537

Benosman, R., Ieng, S.-H., Clercq, C., Bartolozzi, C. & Srinivasan, M. (2012). Asynchronous frameless event-based optical flow. *Neural Networks*, *27*, 32–37. doi:10.1016/j.neunet.2011.11.001

Bi, G.-Q. & Poo, M.-M. (1998). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of Neuroscience*, *18*(24), 10464–10472. doi:10.1523/JNEUROSCI.18-24-10464.1998

Bi, G.-Q. & Poo, M.-M. (2001). Synaptic modification by correlated activity: Hebb's postulate revisited. *Annual Review of Neuroscience*, *24*(1), 139–166. doi:10.1146/annurev.neuro.24.1.139

Bichler, O., Querlioz, D., Thorpe, S. J., Bourgoin, J.-P. & Gamrat, C. (2012). Extraction of temporally correlated features from dynamic vision sensors with spike-timing-dependent plasticity. *Neural Networks*. Selected Papers from IJCNN 2011, *32*, 339–348. doi:10.1016/j.neunet.2012.02.022

Bienenstock, E. L., Cooper, L. N. & Munro, P. W. (1982). Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex. *Journal of Neuroscience*, *2*(1), 32–48. doi:10.1523/JNEUROSCI.02-01-00032.1982

Bing, Z., Meschede, C., Huang, K., Chen, G., Rohrbein, F., Akl, M. & Knoll, A. (2018). End to end learning of spiking neural network based on R-STDP for a lane keeping vehicle. In *2018 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 1–8). doi:10.1109/ICRA.2018.8460482

Bing, Z., Baumann, I., Jiang, Z., Huang, K., Cai, C. & Knoll, A. (2019). Supervised learning in SNN via reward-modulated spike-timing-dependent plasticity for a target reaching vehicle. *Frontiers in Neurorobotics*, *13*. doi:10.3389/fnbot.2019.00018

Bing, Z., Meschede, C., Chen, G., Knoll, A. & Huang, K. (2020). Indirect and direct training of spiking neural networks for end-to-end control of a lane-keeping vehicle. *Neural Networks*, *121*, 21–36. doi:10.1016/j.neunet.2019.05.019

Bing, Z., Meschede, C., Röhrbein, F., Huang, K. & Knoll, A. C. (2018). A survey of robotics control based on learning-inspired spiking neural networks. *Frontiers in Neurorobotics*, *12*. doi:10.3389/fnbot.2018.00035

Bohte, S. M., Kok, J. N. & La Poutré, H. (2002). Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, *48*(1), 17–37. doi:10.1016/S0925-2312(01)00658-0

Borst, A., Haag, J. & Reiff, D. F. (2010). Fly motion vision. *Annual Review of Neuroscience*, *33*(1), 49–70. doi:10.1146/annurev-neuro-060909-153155

Bouvier, M., Valentian, A., Mesquida, T., Rummens, F., Reyboz, M., Vianello, E. & Beigne, E. (2019). Spiking neural networks hardware implementations and challenges: a survey. *J. Emerg. Technol. Comput. Syst. 15*(2), 22:1–22:35. doi:10.1145/3304103

Brandli, C., Berner, R., Yang, M., Liu, S. & Delbruck, T. (2014). A 240 × 180 130 dB 3 µs latency global shutter spatiotemporal vision sensor. *IEEE Journal of Solid-State Circuits*, *49*(10), 2333–2341. doi:10.1109/JSSC.2014.2342715

Brosch, T. & Neumann, H. (2016). Event-based optical flow on neuromorphic hardware. In *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (Formerly BIONETICS)* (pp. 551–558). BICT'15. doi:10.4108/eai.3-12-2015.2262447

Brosch, T., Tschechne, S. & Neumann, H. (2015). On event-based optical flow detection. *Frontiers in Neuroscience*, *9*. doi:10.3389/fnins.2015.00137

Burda, Y., Edwards, H., Storkey, A. & Klimov, O. (2018). Exploration by random network distillation. *arXiv:1810.12894 [cs, stat]*. Retrieved May 2, 2019, from http://arxiv.org/abs/1810.12894

Camus, T. (1997). Real-time quantized optical flow. *Real-Time Imaging*, *3*(2), 71–86. doi:10.1006/rtim.1996.0048

Caporale, N. & Dan, Y. (2008). Spike timing–dependent plasticity: a Hebbian learning rule. *Annual Review of Neuroscience*, *31*(1), 25–46. doi:10.1146/annurev.neuro.31.060407.125639

Chahl, J. S., Srinivasan, M. V. & Zhang, S. W. (2004). Landing strategies in honeybees and applications to uninhabited airborne vehicles. *The International Journal of Robotics Research*, *23*(2), 101–110. doi:10.1177/0278364904041320

Clawson, T. S., Ferrari, S., Fuller, S. B. & Wood, R. J. (2016). Spiking neural network (SNN) control of a flapping insect-scale robot. In *2016 IEEE 55th Conference on Decision and Control (CDC)* (pp. 3381–3388). doi:10.1109/CDC.2016.7798778

Conradt, J. (2015). On-board real-time optic-flow for miniature event-based vision sensors. In *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)* (pp. 1858–1863). doi:10.1109/ROBIO.2015.7419043

Conroy, J., Gremillion, G., Ranganathan, B. & Humbert, J. S. (2009). Implementation of wide-field integration of optic flow for autonomous quadrotor navigation. *Autonomous Robots*, *27*(3), 189. doi:10.1007/s10514-009-9140-0

Constantinescu, A. O., O'Reilly, J. X. & Behrens, T. E. J. (2016). Organizing conceptual knowledge in humans with a gridlike code. *Science*, *352*(6292), 1464–1468. doi:10.1126/science.aaf0941

Davies, M., Srinivasa, N., Lin, T., Chinya, G., Cao, Y., Choday, S. H., … Wang, H. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro*, *38*(1), 82–99. doi:10.1109/MM.2018.112130359

de Croon, G. C. H. E. (2016). Monocular distance estimation with optical flow maneuvers and efference copies: a stability-based strategy. *Bioinspiration & Biomimetics*, *11*(1), 016004. doi:10.1088/1748-3190/11/1/016004

de Croon, G. C. H. E., Ho, H. W., De Wagter, C., van Kampen, E., Remes, B. & Chu, Q. P. (2013). Optic-flow based slope estimation for autonomous landing. *International Journal of Micro Air Vehicles*, *5*(4), 287–297. doi:10.1260/1756-8293.5.4.287

de Croon, G., Dartel, M. F. V. & Postma, E. O. (2005). Evolutionary learning outperforms reinforcement learning on non-Markovian tasks. In *Workshop on Memory and Learning Mechanisms in Autonomous Robots, 8th European Conference on Artificial Life*.

de Croon, G., de Clercq, K., Ruijsink, R., Remes, B. & de Wagter, C. (2009). Design, aerodynamics, and vision-based control of the DelFly. *International Journal of Micro Air Vehicles*, *1*(2), 71–97. doi:10.1260/175682909789498288

di Castro, D., Volkinshtein, D. & Meir, R. (2009). Temporal difference based actor critic learning - convergence and neural implementation. In *Advances in neural information processing systems 21* (pp. 385–392). Curran Associates, Inc. Retrieved February 25, 2019, from http://papers.nips.cc/paper/3517-temporal-difference-based-actor-critic-learning-convergence-and-neural-implementation.pdf

Diehl, P. U. & Cook, M. (2014). Efficient implementation of STDP rules on SpiNNaker neuromorphic hardware. In *2014 International Joint Conference on Neural Networks (IJCNN)* (pp. 4288–4295). doi:10.1109/IJCNN.2014.6889876

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S. & Pfeiffer, M. (2015). Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International Joint Conference on Neural Networks (IJCNN)* (pp. 1–8). doi:10.1109/IJCNN.2015.7280696

Diehl, P. U. & Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience*, *9*. doi:10.3389/fncom.2015.00099

Doya, K. (2000). Reinforcement learning in continuous time and space. *Neural Computation*, *12*(1), 219–245. doi:10.1162/089976600300015961

Eichner, H., Joesch, M., Schnell, B., Reiff, D. F. & Borst, A. (2011). Internal structure of the fly elementary motion detector. *Neuron*, *70*(6), 1155–1164. doi:10.1016/j.neuron.2011.03.028

Eppler, J. M., Helias, M., Muller, E., Diesmann, M. & Gewaltig, M.-O. (2009). PyNEST: a convenient interface to the NEST simulator. *Frontiers in Neuroinformatics*, *2*. doi:10.3389/neuro.11.012.2008

Expert, F. & Ruffier, F. (2015). Flying over uneven moving terrain based on optic-flow cues without any need for reference frames or accelerometers. *Bioinspiration & Biomimetics*, *10*(2), 026003. doi:10.1088/1748-3182/10/2/026003

Faghihi, F., Moustafa, A. A., Heinrich, R. & Wörgötter, F. (2017). A computational model of conditioning inspired by drosophila olfactory system. *Neural Networks*, *87*, 96–108. doi:10.1016/j.neunet.2016.11.002

Farries, M. A. & Fairhall, A. L. (2007). Reinforcement learning with modulated spike timing–dependent synaptic plasticity. *Journal of Neurophysiology*, *98*(6), 3648–3665. doi:10.1152/jn.00364.2007

Fleet, D. J. & Jepson, A. D. (1990). Computation of component image velocity from local phase information. *International Journal of Computer Vision*, *5*(1), 77–104. doi:10.1007/BF00056772

Floreano, D., Dürr, P. & Mattiussi, C. (2008). Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, *1*(1), 47–62. doi:10.1007/s12065-007-0002-4

Florian, R. V. (2007). Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity. *Neural Computation*, *19*(6), 1468–1502. doi:10.1162/neco.2007.19.6.1468

Foderaro, G., Henriquez, C. & Ferrari, S. (2010). Indirect training of a spiking neural network for flight control via spike-timing-dependent synaptic plasticity. In *49th IEEE Conference on Decision and Control (CDC)* (pp. 911–917). doi:10.1109/CDC.2010.5717260

Fogel, D. B. (1997). The advantages of evolutionary computation. In *Biocomputing and Emergent Computation: Proceedings of BCEC97* (pp. 1–11). World Scientific Press. Retrieved December 18, 2019, from http://dl.acm.org/citation.cfm?id=648178.749054

Franceschini, N., Riehle, A. & Le Nestour, A. (1989). Directionally selective motion detection by insect neurons. In *Facets of Vision* (pp. 360–390). Springer Berlin Heidelberg.

Frémaux, N. & Gerstner, W. (2016). Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Frontiers in Neural Circuits*, *9*. doi:10.3389/fncir.2015.00085

Frémaux, N., Sprekeler, H. & Gerstner, W. (2010). Functional requirements for reward-modulated spike-timing-dependent plasticity. *Journal of Neuroscience*, *30*(40), 13326–13337. doi:10.1523/JNEUROSCI.6249-09.2010

Frémaux, N., Sprekeler, H. & Gerstner, W. (2013). Reinforcement learning using a continuous time actor-critic framework with spiking neurons. *PLOS Computational Biology*, *9*(4), e1003024. doi:10.1371/journal.pcbi.1003024

Furber, S. B., Galluppi, F., Temple, S. & Plana, L. A. (2014). The SpiNNaker project. *Proceedings of the IEEE*, *102*(5), 652–665. doi:10.1109/JPROC.2014.2304638

Gallego, G., Delbruck, T., Orchard, G., Bartolozzi, C., Taba, B., Censi, A., … Scaramuzza, D. (2019). Event-based vision: a survey. *arXiv:1904.08405 [cs]*. Retrieved April 25, 2019, from http://arxiv.org/abs/1904.08405

Georgopoulos, A. P., Schwartz, A. B. & Kettner, R. E. (1986). Neuronal population coding of movement direction. *Science*, *233*(4771), 1416–1419. doi:10.1126/science.3749885

Gerstner, W. & Kistler, W. M. (2002). *Spiking neuron models: single neurons, populations, plasticity*. Cambridge University Press.

Gerstner, W., Kistler, W. M., Naud, R. & Paninski, L. (2014). *Neuronal dynamics: from single neurons to networks and models of cognition*. New York, NY, USA: Cambridge University Press.

Gerstner, W., Lehmann, M., Liakoni, V., Corneil, D. & Brea, J. (2018). Eligibility traces and plasticity on behavioral time scales: experimental support of NeoHebbian three-factor learning rules. *Frontiers in Neural Circuits*, *12*. doi:10.3389/fncir.2018.00053

Gerstner, W., van Hemmen, J. L. & Cowan, J. D. (1996). What matters in neuronal locking? *Neural Computation*, *8*(8), 1653–1676. doi:10.1162/neco.1996.8.8.1653

Gewaltig, M.-O. & Diesmann, M. (2007). NEST (NEural simulation tool). *Scholarpedia*, *2*(4), 1430. doi:10.4249/scholarpedia.1430

Gibson, J. J. (1950). *The perception of the visual world*. Boston: Houghton Mifflin Company.

Goodfellow, I., Bengio, Y. & Courville, A. (2016). *Deep learning*. MIT Press.

Goodman, D. F. M. & Brette, R. (2008). Brian: a simulator for spiking neural networks in python. *Frontiers in Neuroinformatics*, *2*. doi:10.3389/neuro.11.005.2008

Gullapalli, V. (1990). A stochastic reinforcement learning algorithm for learning real-valued functions. *Neural Networks*, *3*(6), 671–692. doi:10.1016/0893-6080(90)90056-Q

Haessig, G., Berthelon, X., Ieng, S.-H. & Benosman, R. (2019). A spiking neural network model of depth from defocus for event-based neuromorphic vision. *Scientific Reports*, *9*(1), 3744. doi:10.1038/s41598-019-40064-0

Haessig, G., Cassidy, A., Alvarez, R., Benosman, R. & Orchard, G. (2018). Spiking optical flow for event-based sensors using IBM's TrueNorth neurosynaptic system. *IEEE Transactions on Biomedical Circuits and Systems*, *12*(4), 860–870. doi:10.1109/TBCAS.2018.2834558

Hafting, T., Fyhn, M., Molden, S., Moser, M.-B. & Moser, E. I. (2005). Microstructure of a spatial map in the entorhinal cortex. *Nature*, *436*(7052), 801. doi:10.1038/nature03721

Harris, C. & Stephens, M. (1988). A combined corner and edge detector. In *In Proc. Fourth Alvey Vision Conference* (pp. 147–152).

Hassenstein, B. & Reichardt, W. (1956). Systemtheoretische analyse der zeit-, reihenfolgen- und vorzeichenauswertung bei der bewegungsperzeption des rüsselkäfers chlorophanus. *Zeitschrift für Naturforschung B*, *11*(9), 513–524. doi:10.1515/znb-1956-9-1004

Hazan, H., Saunders, D. J., Khan, H., Patel, D., Sanghavi, D. T., Siegelmann, H. T. & Kozma, R. (2018). BindsNET: a machine learning-oriented spiking neural networks library in python. *Frontiers in Neuroinformatics*, *12*. doi:10.3389/fninf.2018.00089

Hebb, D. O. (1949). *The organization of behavior: a neuropsychological theory*. New York: John Wiley & Sons, Inc.

Herissé, B., Hamel, T., Mahony, R. & Russotto, F. (2012). Landing a VTOL unmanned aerial vehicle on a moving platform using optical flow. *IEEE Transactions on Robotics*, *28*(1), 77–89. doi:10.1109/TRO.2011.2163435

Ho, H. W. & de Croon, G. C. H. E. (2016). Characterization of flow field divergence for MAVs vertical control landing. In *AIAA Guidance, Navigation, and Control Conference*. doi:10.2514/6.2016-0106

Ho, H. W., de Croon, G. C. H. E., van Kampen, E., Chu, Q. P. & Mulder, M. (2018). Adaptive gain control strategy for constant optical flow divergence landing. *IEEE Transactions on Robotics*, *34*(2), 508–516. doi:10.1109/TRO.2018.2817418

Hodgkin, A. L. & Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, *117*(4), 500–544. doi:10.1113/jphysiol.1952.sp004764

Horn, B. K. P. & Schunck, B. G. (1981). Determining optical flow. *Artificial Intelligence*, *17*(1), 185–203. doi:10.1016/0004-3702(81)90024-2

Hull, C. L. (1932). The goal-gradient hypothesis and maze learning. *Psychological Review*, *39*(1), 25–43. doi:10.1037/h0072640

Hull, C. L. (1943). *Principles of behavior: an introduction to behavior theory*. New York, USA: Appleton-Century.

Izhikevich, E. M. (2007). Solving the distal reward problem through linkage of STDP and dopamine signaling. *Cerebral Cortex*, *17*(10), 2443–2452. doi:10.1093/cercor/bhl152

Izhikevich, E. M. & Desai, N. S. (2003). Relating STDP to BCM. *Neural Computation*, *15*(7), 1511–1523. doi:10.1162/089976603321891783

Izzo, D. & de Croon, G. C. H. E. (2012). Landing with time-to-contact and ventral optic flow estimates. *Journal of Guidance, Control, and Dynamics*, *35*(4), 1362–1367. doi:10.2514/1.56598

Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., … Michalewski, H. (2019). Model-based reinforcement learning for Atari. *arXiv:1903.00374 [cs, stat]*. Retrieved November 21, 2019, from http://arxiv.org/abs/1903.00374

Karásek, M., Muijres, F. T., Wagter, C. D., Remes, B. D. W. & de Croon, G. C. H. E. (2018). A tailless aerial robotic flapper reveals that flies use torque coupling in rapid banked turns. *Science*, *361*(6407), 1089–1094. doi:10.1126/science.aat0350

Kendoul, F., Fantoni, I. & Nonami, K. (2009). Optic flow-based vision system for autonomous 3d localization and control of small aerial vehicles. *Robotics and Autonomous Systems*, *57*(6), 591–602. doi:10.1016/j.robot.2009.02.001

Keshavan, J., Gremillion, G., Escobar-Alvarez, H. & Humbert, J. S. (2014). A analysis-based, controller-synthesis framework for robust bioinspired visual navigation in less-structured environments. *Bioinspiration & Biomimetics*, *9*(2), 025011. doi:10.1088/1748-3182/9/2/025011

Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S. J. & Masquelier, T. (2018). STDP-based spiking deep convolutional neural networks for object recognition. *Neural Networks*, *99*, 56–67. doi:10.1016/j.neunet.2017.12.005

Killian, N. J., Jutras, M. J. & Buffalo, E. A. (2012). A map of visual space in the primate entorhinal cortex. *Nature*, *491*(7426), 761–764. doi:10.1038/nature11587

Kistler, W. M., Gerstner, W. & Hemmen, J. L. v. (1997). Reduction of the Hodgkin-Huxley equations to a single-variable threshold model. *Neural Computation*, *9*(5), 1015–1045. doi:10.1162/neco.1997.9.5.1015

Klopf, A. H. (1972). *Brain function and adaptive systems: a heterostatic theory* (No. AFCRL-SR-133). AIR FORCE CAMBRIDGE RESEARCH LABS HANSCOM AFB MA. Retrieved May 3, 2019, from https://apps.dtic.mil/docs/citations/AD0742259

Klopf, A. H. (1982). *The hedonistic neuron: a theory of memory, learning, and intelligence*. Washington, DC, USA: Hemisphere Pub. Corp.

Kober, J., Bagnell, J. A. & Peters, J. (2013). Reinforcement learning in robotics: a survey. *The International Journal of Robotics Research*, *32*(11), 1238–1274. doi:10.1177/0278364913495721

Kober, J. & Peters, J. R. (2009). Policy search for motor primitives in robotics. In *Advances in neural information processing systems 21* (pp. 849–856). Curran Associates, Inc. Retrieved May 4, 2019, from http://papers.nips.cc/paper/3545-policy-search-for-motor-primitives-in-robotics.pdf

Laud, A. & DeJong, G. (2002). Reinforcement learning and shaping: encouraging intended behaviors. In *Proceedings of the Nineteenth International Conference on Machine Learning* (pp. 355–362). ICML '02. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. Retrieved March 12, 2019, from http://dl.acm.org/citation.cfm?id=645531.656003

Laud, A. & DeJong, G. (2003). The influence of reward on the speed of reinforcement learning: an analysis of shaping. In *Proceedings of the Twentieth International Conference on Machine Learning* (pp. 440–447). ICML'03. AAAI Press. Retrieved March 12, 2019, from http://dl.acm.org/citation.cfm?id=3041838.3041894

Lee, D. N. (1976). A theory of visual control of braking based on information about time-to-collision. *Perception*, *5*(4), 437–459. doi:10.1068/p050437

Lee, D. N., Davies, M. N. O., Green, P. R. & van der Weel, F. R. (1993). Visual control of velocity of approach by pigeons when landing. *Journal of Experimental Biology*, *180*(1), 85–104. Retrieved March 5, 2019, from http://jeb.biologists.org/content/180/1/85

Legenstein, R., Pecevski, D. & Maass, W. (2008). A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback. *PLOS Computational Biology*, *4*(10), e1000180. doi:10.1371/journal.pcbi.1000180

Lichtsteiner, P., Posch, C. & Delbruck, T. (2008). A 128 × 128 120 dB 15 µs latency asynchronous temporal contrast vision sensor. *IEEE Journal of Solid-State Circuits*, *43*(2), 566–576. doi:10.1109/JSSC.2007.914337

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., … Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv:1509.02971 [cs, stat]*. Retrieved April 26, 2019, from http://arxiv.org/abs/1509.02971

Lin, C., Wild, A., Chinya, G. N., Cao, Y., Davies, M., Lavery, D. M. & Wang, H. (2018). Programming spiking neural networks on Intel's Loihi. *Computer*, *51*(3), 52–61. doi:10.1109/MC.2018.157113521

Longuet-Higgins, H. C. & Prazdny, K. (1980). The interpretation of a moving retinal image. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, *208*(1173), 385–397. doi:10.1098/rspb.1980.0057

Lucas, B. D. & Kanade, T. (1981). An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence* (Vol. 2, pp. 674–679). IJCAI'81. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. Retrieved March 4, 2019, from http://dl.acm.org/citation.cfm?id=1623264.1623280

Ma, K. Y., Chirarattananon, P., Fuller, S. B. & Wood, R. J. (2013). Controlled flight of a biologically inspired, insect-scale robot. *Science*, *340*(6132), 603–607. doi:10.1126/science.1231806

Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, *10*(9), 1659–1671. doi:10.1016/S0893-6080(97)00011-7

Markram, H., Lübke, J., Frotscher, M. & Sakmann, B. (1997). Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science*, *275*(5297), 213–215. doi:10.1126/science.275.5297.213

Martin, S. J., Grimwood, P. D. & Morris, R. G. M. (2000). Synaptic plasticity and memory: an evaluation of the hypothesis. *Annual Review of Neuroscience*, *23*(1), 649–711. doi:10.1146/annurev.neuro.23.1.649

Mataric, M. J. (1994). Reward functions for accelerated learning. In *Machine learning proceedings 1994* (pp. 181–189). doi:10.1016/B978-1-55860-335-6.50030-1

McCulloch, W. S. & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*(4), 115–133. doi:10.1007/BF02478259

Mead, C. (1990). Neuromorphic electronic systems. *Proceedings of the IEEE*, *78*(10), 1629–1636. doi:10.1109/5.58356

Mead, C. (1989). *Analog VLSI and neural systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., … Modha, D. S. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, *345*(6197), 668–673. doi:10.1126/science.1254642

Mikaitis, M., Pineda García, G., Knight, J. C. & Furber, S. B. (2018). Neuromodulated synaptic plasticity on the SpiNNaker neuromorphic system. *Frontiers in Neuroscience*, *12*. doi:10.3389/fnins.2018.00105

Minsky, M. (1961). Steps toward artificial intelligence. *Proceedings of the IRE*, *49*(1), 8–30. doi:10.1109/JRPROC.1961.287775

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv:1312.5602 [cs]*. Retrieved April 30, 2019, from http://arxiv.org/abs/1312.5602

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., … Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. doi:10.1038/nature14236

Moldovan, T. M. & Abbeel, P. (2012). Safe exploration in Markov decision processes. *arXiv:1205.4810 [cs]*. Retrieved May 4, 2019, from http://arxiv.org/abs/1205.4810

Montague, P. R., Dayan, P. & Sejnowski, T. J. (1996). A framework for mesencephalic dopamine systems based on predictive Hebbian learning. *Journal of Neuroscience*, *16*(5), 1936–1947. doi:10.1523/JNEUROSCI.16-05-01936.1996

Morrison, A., Aertsen, A. & Diesmann, M. (2007). Spike-timing-dependent plasticity in balanced random networks. *Neural Computation*, *19*(6), 1437–1467. doi:10.1162/neco.2007.19.6.1437

Moser, E. I., Kropff, E. & Moser, M.-B. (2008). Place cells, grid cells, and the brain's spatial representation system. *Annual Review of Neuroscience*, *31*(1), 69–89. doi:10.1146/annurev.neuro.31.061307.090723

Mozafari, M., Ganjtabesh, M., Nowzari-Dalini, A. & Masquelier, T. (2019). SpykeTorch: efficient simulation of convolutional spiking neural networks with at most one spike per neuron. *arXiv:1903.02440 [cs, q-bio]*. Retrieved March 13, 2019, from http://arxiv.org/abs/1903.02440

Neftci, E. O. & Averbeck, B. B. (2019). Reinforcement learning in artificial and biological systems. *Nature Machine Intelligence*, *1*(3), 133–143. doi:10.1038/s42256-019-0025-4

Ng, A. Y., Harada, D. & Russell, S. J. (1999). Policy invariance under reward transformations: theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on*

Machine Learning (pp. 278–287). ICML '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. Retrieved March 12, 2019, from http://dl.acm.org/citation.cfm?id=645528.657613

Ng, A. Y. & Russell, S. J. (2000). Algorithms for inverse reinforcement learning. In *in Proc. 17th International Conf. on Machine Learning* (pp. 663–670). Morgan Kaufmann.

Nichols, E., McDaid, L. J. & Siddique, N. (2013). Biologically inspired SNN for robot control. *IEEE Transactions on Cybernetics*, *43*(1), 115–128. doi:10.1109/TSMCB.2012.2200674

O'Doherty, J., Dayan, P., Schultz, J., Deichmann, R., Friston, K. & Dolan, R. J. (2004). Dissociable roles of ventral and dorsal striatum in instrumental conditioning. *Science*, *304*(5669), 452–454. doi:10.1126/science.1094285

O'Keefe, J. & Dostrovsky, J. (1971). The hippocampus as a spatial map: preliminary evidence from unit activity in the freely-moving rat. *Brain Research*, *34*, 171–175. doi:10.1016/0006-8993(71)90358-1

Olshausen, B. A. & Field, D. J. (2004). Sparse coding of sensory inputs. *Current Opinion in Neurobiology*, *14*(4), 481–487. doi:10.1016/j.conb.2004.07.007

Orchard, G., Benosman, R., Etienne-Cummings, R. & Thakor, N. V. (2013). A spiking neural network architecture for visual motion estimation. In *2013 IEEE Biomedical Circuits and Systems Conference (BioCAS)* (pp. 298–301). doi:10.1109/BioCAS.2013.6679698

Orchard, G. & Etienne-Cummings, R. (2014). Bioinspired visual motion estimation. *Proceedings of the IEEE*, *102*(10), 1520–1536. doi:10.1109/JPROC.2014.2346763

Orchard, G., Jayawant, A., Cohen, G. K. & Thakor, N. (2015). Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in Neuroscience*, *9*. doi:10.3389/fnins.2015.00437

Paredes-Vallés, F., Scheper, K. Y. W. & de Croon, G. C. H. E. (2019). Unsupervised learning of a hierarchical spiking neural network for optical flow estimation: from events to global motion perception. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1–1. doi:10.1109/TPAMI.2019.2903179

Pavlov, I. P. (1927). *Conditioned reflexes: an investigation of the physiological activity of the cerebral cortex*. London, England: Oxford University Press.

Pfeiffer, M. & Pfeil, T. (2018). Deep learning with spiking neurons: opportunities and challenges. *Frontiers in Neuroscience*, *12*. doi:10.3389/fnins.2018.00774

Pfister, J.-P., Toyoizumi, T., Barber, D. & Gerstner, W. (2006). Optimal spike-timing-dependent plasticity for precise action potential firing in supervised learning. *Neural Computation*, *18*(6), 1318–1348. doi:10.1162/neco.2006.18.6.1318

Pijnacker Hordijk, B. J., Scheper, K. Y. W. & de Croon, G. C. H. E. (2018). Vertical landing for micro air vehicles using event-based optical flow. *Journal of Field Robotics*, *35*(1), 69–90. doi:10.1002/rob.21764

Polvara, R., Patacchiola, M., Sharma, S., Wan, J., Manning, A., Sutton, R. & Cangelosi, A. (2018). Autonomous quadrotor landing using deep reinforcement learning. *arXiv:1709.03339 [cs]*. Retrieved March 12, 2019, from http://arxiv.org/abs/1709.03339

Posch, C., Matolin, D. & Wohlgenannt, R. (2011). A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS. *IEEE Journal of Solid-State Circuits*, *46*(1), 259–275. doi:10.1109/JSSC.2010.2085952

Posch, C., Serrano-Gotarredona, T., Linares-Barranco, B. & Delbruck, T. (2014). Retinomorphic event-based vision sensors: bioinspired cameras with spiking output. *Proceedings of the IEEE*, *102*(10), 1470–1484. doi:10.1109/JPROC.2014.2346153

Potjans, W., Morrison, A. & Diesmann, M. (2009). A spiking neural network model of an actor-critic learning agent. *Neural Computation*, *21*(2), 301–339. doi:10.1162/neco.2008.08-07-593

Rescorla, R. A. & Wagner, A. R. (1972). A theory of pavlovian conditioning: variations in the effectiveness of reinforcement and nonreinforcement. In *Classical conditioning II: current research and theory* (Vol. Vol. 2, pp. 64–99).

Rodriguez-Ramos, A., Sampedro, C., Bavle, H., Moreno, I. G. & Campoy, P. (2018). A deep reinforcement learning technique for vision-based autonomous multirotor landing on a moving platform. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 1010–1017). doi:10.1109/IROS.2018.8594472

Rodriguez-Ramos, A., Sampedro, C., Bavle, H., de la Puente, P. & Campoy, P. (2019). A deep reinforcement learning strategy for UAV autonomous landing on a moving platform. *Journal of Intelligent & Robotic Systems*, *93*(1), 351–366. doi:10.1007/s10846-018-0891-8

Romo, R. & Schultz, W. (1990). Dopamine neurons of the monkey midbrain: contingencies of responses to active touch during self-initiated arm movements. *Journal of Neurophysiology*, *63*(3), 592–606. doi:10.1152/jn.1990.63.3.592

Rossum, M. C. W. v., Bi, G. Q. & Turrigiano, G. G. (2000). Stable Hebbian learning from spike timing-dependent plasticity. *Journal of Neuroscience*, *20*(23), 8812–8821. doi:10.1523/JNEUROSCI.20-23-08812.2000

Rosten, E. & Drummond, T. (2006). Machine learning for high-speed corner detection. In *Computer Vision – ECCV 2006* (pp. 430–443). Lecture Notes in Computer Science. Springer Berlin Heidelberg.

Roy, K., Jaiswal, A. & Panda, P. (2019). Towards spike-based machine intelligence with neuromorphic computing. *Nature*, *575*(7784), 607–617. doi:10.1038/s41586-019-1677-2

Rueckauer, B. & Delbruck, T. (2016). Evaluation of event-based algorithms for optical flow with ground-truth from inertial measurement sensor. *Frontiers in Neuroscience*, *10*. doi:10.3389/fnins.2016.00176

Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M. & Liu, S.-C. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience*, *11*. doi:10.3389/fnins.2017.00682

Ruffier, F. & Franceschini, N. (2005). Optic flow regulation: the key to aircraft automatic guidance. *Robotics and Autonomous Systems*. Biomimetic Robotics, *50*(4), 177–194. doi:10.1016/j.robot.2004.09.016

Ruffier, F. & Franceschini, N. (2015). Optic flow regulation in unsteady environments: a tethered MAV achieves terrain following and targeted landing over a moving platform. *Journal of Intelligent & Robotic Systems*, *79*(2), 275–293. doi:10.1007/s10846-014-0062-5

Sanket, N. J., Parameshwara, C. M., Singh, C. D., Kuruttukulam, A. V., Fermüller, C., Scaramuzza, D. & Aloimonos, Y. (2019). EVDodge: embodied AI for high-speed dodging on a quadrotor using event cameras. *arXiv:1906.02919 [cs]*. Retrieved June 21, 2019, from http://arxiv.org/abs/1906.02919

Scaramuzza, D. & Fraundorfer, F. (2011). Visual odometry part i: the first 30 years and fundamentals. *IEEE Robotics Automation Magazine*, *18*(4), 80–92. doi:10.1109/MRA.2011.943233

Scheper, K. Y. W. & de Croon, G. C. H. E. (2020). Evolution of robust high speed optical-flow-based landing for autonomous MAVs. *Robotics and Autonomous Systems*, *124*, 103380. doi:10.1016/j.robot.2019.103380

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv:1707.06347 [cs]*. Retrieved May 2, 2019, from http://arxiv.org/abs/1707.06347

Schultz, W. & Romo, R. (1990). Dopamine neurons of the monkey midbrain: contingencies of responses to stimuli eliciting immediate behavioral reactions. *Journal of Neurophysiology*, *63*(3), 607–624. doi:10.1152/jn.1990.63.3.607

Schultz, W. (1998). Predictive reward signal of dopamine neurons. *Journal of Neurophysiology*, *80*(1), 1–27. doi:10.1152/jn.1998.80.1.1

Schultz, W. (2002). Getting formal with dopamine and reward. *Neuron*, *36*(2), 241–263. doi:10.1016/S0896-6273(02)00967-4

Schultz, W., Dayan, P. & Montague, P. R. (1997). A neural substrate of prediction and reward. *Science*, *275*(5306), 1593–1599. doi:10.1126/science.275.5306.1593

Serres, J. R. & Ruffier, F. (2017). Optic flow-based collision-free strategies: from insects to robots. *Arthropod Structure & Development*. From Insects to Robots, *46*(5), 703–717. doi:10.1016/j.asd.2017.06.003

Shrestha, A., Ahmed, K., Wang, Y. & Qiu, Q. (2017). Stable spike-timing-dependent plasticity rule for multilayer unsupervised and supervised learning. In *2017 International Joint Conference on Neural Networks (IJCNN)* (pp. 1999–2006). doi:10.1109/IJCNN.2017.7966096

Shrestha, S. B. & Orchard, G. (2018). SLAYER: spike layer error reassignment in time. In *Advances in neural information processing systems 31* (pp. 1412–1421). Curran Associates, Inc. Retrieved March 13, 2019, from http://papers.nips.cc/paper/7415-slayer-spike-layer-error-reassignment-in-time.pdf

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., … Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, *529*(7587), 484–489. doi:10.1038/nature16961

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., … Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, *362*(6419), 1140–1144. doi:10.1126/science.aar6404

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., … Hassabis, D. (2017). Mastering the game of go without human knowledge. *Nature*, *550*(7676), 354–359. doi:10.1038/nature24270

Skinner, B. F. (1938). *The behavior of organisms: an experimental analysis*. New York, USA: Appleton-Century.

Skinner, B. F. (1958). Reinforcement today. *American Psychologist*, *13*(3), 94–99. doi:10.1037/h0049039

Skinner, B. F. (1963). Operant behavior. *American Psychologist*, *18*(8), 503–515. doi:10.1037/h0045185

Smart, W. D. & Kaelbling, L. P. (2002). Effective reinforcement learning for mobile robots. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)* (Vol. 4, 3404–3410 vol.4). doi:10.1109/ROBOT.2002.1014237

Song, S., Miller, K. D. & Abbott, L. F. (2000). Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. *Nature Neuroscience*, *3*(9), 919–926. doi:10.1038/78829

Sorg, J., Lewis, R. L. & Singh, S. P. (2010). Reward design via online gradient ascent. In *Advances in neural information processing systems 23* (pp. 2190–2198). Curran Associates, Inc. Retrieved May 1, 2019, from http://papers.nips.cc/paper/4146-reward-design-via-online-gradient-ascent.pdf

Spüler, M., Nagel, S. & Rosenstiel, W. (2015). A spiking neuronal model learning a motor control task by reinforcement learning and structural synaptic plasticity. In *2015 International Joint Conference on Neural Networks (IJCNN)* (pp. 1–8). doi:10.1109/IJCNN.2015.7280521

Srinivasan, M. V., Lehrer, M., Kirchner, W. H. & Zhang, S. W. (1991). Range perception through apparent image speed in freely flying honeybees. *Visual Neuroscience*, *6*(5), 519–535. doi:10.1017/S095252380000136X

Srinivasan, M., Zhang, S., Lehrer, M. & Collett, T. (1996). Honeybee navigation en route to the goal: visual flight control and odometry. *Journal of Experimental Biology*, *199*(1), 237–244. Retrieved March 5, 2019, from http://jeb.biologists.org/content/199/1/237

Stanley, K., Bryant, B. & Miikkulainen, R. (2005). Real-time neuroevolution in the NERO video game. *IEEE Transactions on Evolutionary Computation*, *9*(6), 653–668. doi:10.1109/TEVC.2005.856210

Stein, R. B. (1965). A theoretical analysis of neuronal variability. *Biophysical Journal*, *5*(2), 173–194. doi:10.1016/S0006-3495(65)86709-1

Stein, R. B. (1967). Some models of neuronal variability. *Biophysical Journal*, *7*(1), 37–68. doi:10.1016/S0006-3495(67)86574-3

Stimberg, M., Goodman, D. F. M., Benichoux, V. & Brette, R. (2013). Brian 2 - the second coming: spiking neural network simulation in python with code generation. *BMC Neuroscience*, *14*(1), P38. doi:10.1186/1471-2202-14-S1-P38

Sutton, R. S. & Barto, A. G. (1998). *Reinforcement learning: an introduction* (1st). Cambridge, MA, USA: MIT Press.

Sutton, R. S. & Barto, A. G. (2018). *Reinforcement learning: an introduction* (2nd). Cambridge, MA, USA: MIT Press.

Sutton, R. S., Koop, A. & Silver, D. (2007). On the role of tracking in stationary environments. (pp. 871–878). doi:10.1145/1273496.1273606

Takahashi, Y., Schoenbaum, G. & Niv, Y. (2008). Silencing the critics: understanding the effects of cocaine sensitization on dorsolateral and ventral striatum in the context of an actor/critic model. *Frontiers in Neuroscience*, *2*. doi:10.3389/neuro.01.014.2008

Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T. & Maida, A. (2019). Deep learning in spiking neural networks. *Neural Networks*, *111*, 47–63. doi:10.1016/j.neunet.2018.12.002

Thorndike, E. L. (1898). Animal intelligence: an experimental study of the associative processes in animals. *The Psychological Review: Monograph Supplements*, *2*(4), i–109. doi:10.1037/h0092987

Thorndike, E. L. (1911). *Animal intelligence: experimental studies*. New York, USA: The Macmillan Company. Retrieved May 2, 2019, from http://archive.org/details/animalintelligen00thor

Thorpe, S. J., Delorme, A. & Van Rullen, R. (2001). Spike-based strategies for rapid processing. *Neural Networks*, *14*(6), 715–725. doi:10.1016/S0893-6080(01)00083-1

Tschechne, S., Sailer, R. & Neumann, H. (2014). Bio-inspired optic flow from event-based neuromorphic sensor input. In *Artificial Neural Networks in Pattern Recognition* (pp. 171–182). Lecture Notes in Computer Science. Springer International Publishing.

Ullman, S. (1979). *The interpretation of visual motion*. The interpretation of visual motion. Oxford, England: MIT Press.

van Hasselt, H. (2010). Double Q-learning. In *Advances in neural information processing systems 23* (pp. 2613–2621). Curran Associates, Inc. Retrieved April 30, 2019, from http://papers.nips.cc/paper/3964-double-q-learning.pdf

van Hasselt, H., Doron, Y., Strub, F., Hessel, M., Sonnerat, N. & Modayil, J. (2018). Deep reinforcement learning and the deadly triad. *arXiv:1812.02648 [cs]*. Retrieved April 30, 2019, from http://arxiv.org/abs/1812.02648

van Hasselt, H., Guez, A. & Silver, D. (2016). Deep reinforcement learning with double Q-learning. In *Thirtieth AAAI Conference on Artificial Intelligence*. Retrieved April 30, 2019, from https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12389

Vasilaki, E., Frémaux, N., Urbanczik, R., Senn, W. & Gerstner, W. (2009). Spike-based reinforcement learning in continuous state and action space: when policy gradient methods fail. *PLOS Computational Biology*, *5*(12), e1000586. doi:10.1371/journal.pcbi.1000586

Vitay, J., Dinkelbach, H. Ü. & Hamker, F. H. (2015). ANNarchy: a code generation approach to neural simulations on parallel hardware. *Frontiers in Neuroinformatics*, *9*. doi:10.3389/fninf.2015.00019

Ye, C., Mitrokhin, A., Fermüller, C., Yorke, J. A. & Aloimonos, Y. (2018). Unsupervised learning of dense optical flow, depth and egomotion from sparse event data. *arXiv:1809.08625 [cs]*. Retrieved September 17, 2019, from http://arxiv.org/abs/1809.08625

Zambrano, D. & Bohte, S. M. (2016). Fast and efficient asynchronous neural computation with adapting spiking neural networks. *arXiv:1609.02053 [cs]*. Retrieved March 12, 2019, from http://arxiv.org/abs/1609.02053

Zamora, I., Lopez, N. G., Vilches, V. M. & Cordero, A. H. (2016). Extending the OpenAI gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo. *arXiv:1608.05742 [cs]*. Retrieved May 4, 2019, from http://arxiv.org/abs/1608.05742

Zhao, F., Zeng, Y. & Xu, B. (2018). A brain-inspired decision-making spiking neural network and its application in unmanned aerial vehicle. *Frontiers in Neurorobotics*, *12*. doi:10.3389/fnbot.2018.00056

Zhu, A. Z., Yuan, L., Chaney, K. & Daniilidis, K. (2018). EV-FlowNet: self-supervised optical flow estimation for event-based cameras. *Robotics: Science and Systems XIV*. doi:10.15607/RSS.2018.XIV.062

Zhu, A. Z., Yuan, L., Chaney, K. & Daniilidis, K. (2019). Unsupervised event-based learning of optical flow, depth, and egomotion. (pp. 989–997). Retrieved June 21, 2019, from http://openaccess.thecvf.com/content_CVPR_2019/html/Zhu_Unsupervised_Event-Based_Learning_of_Optical_Flow_Depth_and_Egomotion_CVPR_2019_paper.html